THÈSE de DOCTORAT de l'UNIVERSITÉ PARIS 6

Spécialité :

INFORMATIQUE

présentée

par **Bruno BOUZY**

pour obtenir le grade de **DOCTEUR de l'UNIVERSITÉ PARIS 6**

Sujet de la thèse :

MODÉLISATION COGNITIVE DU JOUEUR DE GO.

soutenue le Vendredi 13 Janvier 1995

devant le jury composé de :

Paul BOURGINE	Rapporteur
Pierre COLMEZ	Examinateur
Jean MICHEL	Examinateur
André MOUSSA	Examinateur
Jacques PITRAT	Directeur
Georges STAMON	Examinateur
Bernard VICTORRI	Rapporteur

REMERCIEMENTS

Je remercie Jacques Pitrat, directeur de recherche au CNRS, d'avoir accepté de diriger cette thèse dans un domaine novateur et encore peu connu, à savoir le jeu de Go. Je le remercie pour tous les conseils qu'il m'a donnés au cours de cette thèse, pour sa compétence et pour son ouverture d'esprit.

Je remercie Bernard Victorri, directeur de recherche au CNRS, et Paul Bourgine, ingénieur au CEMAGREF, d'avoir rapporté cette thèse.

Je remercie Georges Stamon, directeur de l'EHEI, professeur à Paris 5, Jean Michel, chargé de recherche au CNRS, une fois champion de France de Go, Pierre Colmez, chargé de recherche au CNRS, six fois vice-champion de France de Go, André Moussa, normalien, enseignant de physique, treize fois champion de France de Go, d'avoir accepté d'être examinateurs.

Je remercie le Ministère de la Recherche et de l'Enseignement Supérieur d'avoir supporté cette recherche dans le domaine des Sciences de la Cognition et le LAFORIA de m'avoir accueilli pendant trois ans et offert un environnement scientifique de qualité.

Je remercie Patrick Ricaud, qui a préparé une thèse en IA sur le jeu de Go en même temps que moi, pour toutes les discussions informelles et fructueuses que nous avons eues à propos de l'Intelligence Artificielle et de la programmation du jeu de Go depuis trois ans. Je remercie Tristan Cazenave pour son programme de Go, Gogol, partenaire et concurrent privilégié de INDIGO, son petit frère en quelque sorte, sans lequel INDIGO n'aurait pas autant progressé pendant le dernier trimestre de 1993. Je le remercie aussi pour les discussions que nous avons eues. Je remercie Jean-Marc Nigro, qui a préparé une thèse en IA exactement en même temps que moi, pour toutes les discussions que nous avons eues. Je remercie aussi les autres thèsards du LAFORIA pour la bonne ambiance qu'ils ont su y entretenir, en particulier les "MR" : Mohammed Ramdani et Maria Rifqi.

Je remercie André Bouzy, Jacques Pitrat, Valérie Sion, Tristan Cazenave et Sylvie Kornman pour les parties ou la totalité de la thèse qu'ils ont bien voulu relire et annoter avec précision.

Je remercie le jeu de Go et tous les joueurs de Go d'avoir été sur mon chemin.

Je remercie infiniment mes parents, André et Anne-Marie, mes grand-parents, Paul et Marguerite, Guy et Marie, et mes "beaux-parents" David et Rosette.

Je remercie spécialement Valérie pour tous ses conseils sans lesquels rien de tout ce qui suit n'aurait été possible, pour l'énergie qu'elle m'a donnée pendant trois ans, indispensable à la réalisation de ce travail.

Je remercie Pablo et Lucas pour leur fraîcheur et leur joie quotidienne.

RÉSUMÉ

Notre travail est né de la synergie existant entre la psychologie cognitive, l'Intelligence Artificielle (IA) et le jeu de Go. Pour comprendre le fonctionnement du système cognitif humain, il est fondamental d'expliciter les connaissances intuitives que l'homme utilise dans le monde réel et de construire un modèle cognitif. Le jeu de Go, simplification du monde réel en gardant les caractéristiques spatiales et temporelles, est un domaine d'expérience adapté. La machine permet de mettre en œuvre le modèle cognitif, par construction d'un modèle computationnel et d'un programme. Ceci oblige à expliciter des connaissances, supposées essentiellement non conscientes chez l'homme, étant donné la différence intriguante de niveau entre la machine et l'homme au Go.

Pour construire notre modèle computationnel du joueur de Go, nous avons utilisé les verbalisations de joueurs et nous avons fait des rapprochements entre le jeu de Go d'une part, et des domaines de l'IA et des mathématiques, d'autre part : l'Intelligence Artificielle Distribuée, la morphologie mathématique, la théorie des jeux, le Méta, la Logique Floue.

Le modèle computationnel est orienté objet, structuré en quatre niveaux : le niveau zéro, le niveau élémentaire, le niveau itératif et le niveau global. Il utilise intensivement les jeux de Conway et un demi-millier de règles. Il reconnait statiquement la vie et la mort des groupes en fonction de propriétés internes et de propriétés d'interaction. Il reconnaît les territoires avec des outils de morphologie mathématique. Il remet à jour ses données incrémentalement. Nous faisons correspondre les concepts explicités dans le modèle computationnel avec ceux du modèle cognitif : regroupement, fraction, intérieur, extérieur, interaction, jeu, multi-échelle, multi-critère, stratégie, morphologie, topologie. La large proportion de connaissances non conscientes dans cette liste nous conforte dans notre travail, basé sur le jeu de Go et la machine, pour expliciter des connaissances intuitives.

Le programme INDIGO, développé en C++, joue une partie complète sur des damiers 9-9, 13-13 ou 19-19. Il joue un coup entre cinq secondes et une minute sur des stations de travail Sun, en utilisant un mécanisme de mise à jour des données incrémental. Le niveau du programme est évalué entre 15 ème et 20 ème kyu. Il a été testé contre différents programmes de Go et joueurs humains.

ABSTRACT

Our work is born from synergy between cognitive psychology, Artificial Intelligence (AI) and the game of Go. To understand human cognitive system, it is fundamental to uncover intuitive knowledge that human being uses in the real world and build a cognitive model. The game of Go, real world's simplification that keeps its space and time features, is a well-adapted experiment domain. Computer allows to build a computational model and a program that correspond to the cognitive model. Then, it permits to explicit knowledge that is supposed to be mainly not conscious. This hypothesis would explain the amazing difference between man and machine in Go.

Cognitive modelling is bootstrapped with verbalizations of Go players and continued with machine implementation. We made usefull links between the game of Go and the neighbouring domains of AI and mathematics : DAI, mathematical morphology, game theory, metaknowledge and fuzzy logic.

Computational model is object oriented, structured in four levels : zéro-level, elementary level, iterative level and global level. It uses primarly Conway's games and involves approximately a half-thousand rules. It includes an algorithm which recognizes statically life and death of groups based on interaction with the local neighbourhood. It uses morphology tools to recognize territory in the manner of good human players. It incrementally updates objects. We revealed concepts we linked with human knowledge : grouping, cutting, inside and outside, interaction, game, multi-scale, strategy, morphology, topology. The large part of non conscious concepts in this list reinforced us in our work, based on the game of Go and the machine, to reveal intuitive knowledge of human beings.

The program INDIGO (Is Now Designed In Good Objects), developped with C++, plays complete games on 9-9, 13-13 and 19-19 boards. It plays a move between five seconds and one minute on a Sunsparcstation. Strength of our program is evaluated between 15th and 20th kyu. We tested it against others Go programs as well as human players.

INTRODUCTION

"En voulant, on se trompe souvent.
En ne voulant pas, on se trompe toujours."

Romain Rolland

Exemple

Le jeu de Go est un jeu d'origine chinoise vieux de 4000 ans. Ses règles sont simples mais sa stratégie est complexe. La figure *Introduction-exemple* donne un exemple de position.

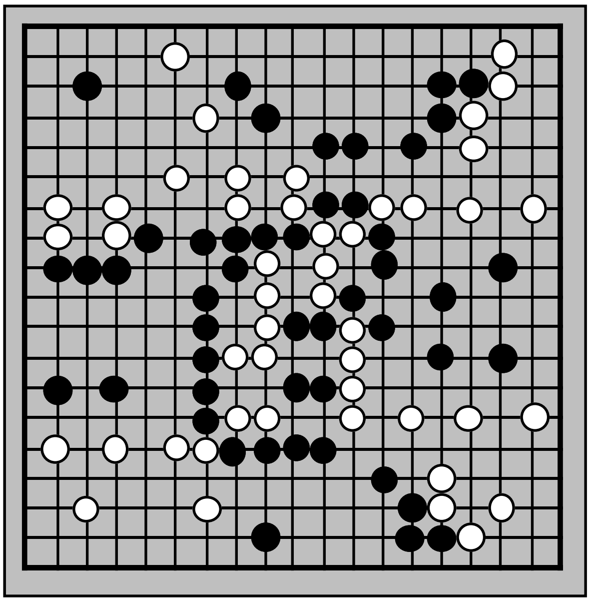


figure Introduction-exemple

Elle paraît confuse pour le novice. Le joueur de Go sait par contre y voir un ordre et choisir un coup. Mais le joueur de Go ne sait pas toujours expliquer clairement la façon dont il joue. Il utilise des connaissances intuitives acquises dans le monde réel. Modéliser la façon dont un être humain joue au Go est une approche intéressante pour expliciter des connaissances intuitives. Mais elle est complexe. Nous avons voulu tout de même l'entreprendre au cours de notre travail de thèse. Le but de ce document est de présenter ce travail.

Structure du document

Dans la première partie, nous expliquons quel a été le but de notre thèse : élaborer un modèle cognitif du joueur de Go, le valider en l'implémentant sous forme de programme, pour expliciter des connaissances intuitives de l'être humain. Nous précisons pourquoi nous avons choisi le jeu de Go comme domaine d'expérience. Nous définissons ce que nous appelons un modèle cognitif.

Dans la deuxième partie, nous présentons la méthode que nous avons utilisée pour construire notre modèle cognitif. Nous tirons les enseignements de notre première version du modèle cognitif du joueur de Go. Nous discutons de l'utilité des expériences basées sur des verbalisations. Nous montrons comment l'implémentation sur machine a permis de valider notre modèle et d'expliciter des connaissances intuitives. Enfin, nous montrons les analogies que nous avons faites entre le jeu de Go d'une part, et des domaines de l'IA et des mathématiques d'autre part - les domaines voisins. Ces analogies nous ont permis de mettre au point des techniques utiles pour notre travail. Nous terminons cette partie par l'état de l'art de la programmation du jeu de Go.

Dans la troisième partie, nous présentons notre modèle et le programme INDIGO, réalisation informatique validant notre modèle cognitif. Les caractéristiques essentielles du modèle sont une conception orientée objet, une structuration en niveaux conceptuels et l'utilisation du concept de jeu, hérité de la théorie de Conway. En particulier, un point fort de notre modèle est un algorithme qui reconnaît statiquement si un groupe est mort en fonction de propriétés internes et de propriétés d'interaction avec le voisinage ennemi du groupe. A chaque concept présent dans notre modèle, nous faisons correspondre des connaissances du joueur humain et nous discutons de son degré de conscience chez l'homme.

Dans la quatrième partie, nous évaluons notre travail. Au premier chapitre, nous donnons les résultats du programme INDIGO et nous discutons de l'aspect pratique et des perspectives envisageables pour ce programme. Au deuxième chapitre, nous présentons les parties de notre modèle que nous estimons être l'équivalent de connaissances intuitives d'un être humain. Enfin, au troisième chapitre, nous présentons notre travail au travers des domaines voisins tels que : le Méta, l'Intelligence Artificielle Distribuée et la Logique Floue.

Enfin, nous concluons.

Conseil au lecteur

D'abord, nous voulons montrer comment ces domaines voisins ont été utilisés dans notre travail. En retour, nous voulons montrer comment notre modélisation du jeu de Go est un exemple enrichissant la liste des applications de ces domaines. Nous avons donc placé l'utilisation des domaines voisins dans la deuxième partie du document relative à la méthode et les exemples illustrant un domaine dans la quatrième partie sur l'évaluation de notre travail. Certains paragraphes sur l'utilisation des domaines voisins pourraient ne pas être compris à la première lecture. Nous conseillons donc d'effectuer plusieurs lectures, les paragraphes s'éclairant mutuellement.

Nous conseillons au non joueur de Go de lire les règles du jeu de Go d'abord. Elles sont présentées en annexe.

Nous ne présentons pas le détail du programme INDIGO mais le modèle qui le supporte. Par conséquent, celui qui ne connaît pas l'informatique peut tout de même lire ce document.

Le programmeur de Go trouvera dans cette thèse matière explicite pour améliorer son programme. Cette thèse présente l'intérieur du programme INDIGO.

PARTIE 1 : BUT DE LA THÈSE

Plan de la partie

Dans le premier paragraphe, nous présentons le but de notre thèse ainsi que nos convictions sans les discuter ni les argumenter.

Ce n'est qu'ensuite que nous développons ces convictions, donnons des exemples et proposons des définitions et des notations.

But et convictions

Avant de démarrer notre thèse, nous avons voulu **élaborer un modèle cognitif dans un domaine complexe, plus simple que le monde réel, où l'homme utilise des connaissances intuitives**.

Nous sommes de formation mathématicienne avec quatre ans d'expérience en informatique au début de notre thèse et nous jouons bien au Go. Nous avons voulu faire une thèse en sciences cognitives et nous avons les convictions suivantes.

Le système cognitif humain est fait pour comprendre le monde réel, qui justifie l'existence d'un organe comme le cerveau. Dans cette optique, l'étude du système cognitif humain doit être faite, en situation d'activité, au sein du monde réel. Celui-ci est très complexe. Le système cognitif humain est composé d'une partie non consciente et d'une partie consciente. Nous pensons que la partie non consciente est faite pour cacher la complexité du monde réel et le faire apparaître sous une forme simplifiée dont la cohérence est accessible à la partie consciente. Le fonctionnement non conscient supportant le fonctionnement conscient, il est plus important d'étudier le fonctionnement non conscient du système cognitif humain que le fonctionnement conscient.

Pour étudier le fonctionnement non conscient du système cognitif humain plusieurs difficultés se présentent. D'abord, le fonctionnement non conscient du système cognitif humain semble inaccessible par définition. Cependant, il est possible d'en atteindre une partie par des travaux de conscientisation basés sur la verbalisation de sujets. Ensuite, le monde réel est trop complexe. Mener des expériences significatives et reproductibles sur des sujets en situation d'activité dans le monde réel est beaucoup trop difficile. Il est préférable de trouver un domaine moins complexe pour faciliter les expériences. Cependant, ce domaine doit préserver les caractéristiques du monde réel. En particulier, sa complexité doit être telle qu'un sujet en activité dans ce domaine utilise des connaissances non conscientes. Le jeu de Go, possèdant des éléments de complexité propres au monde réel, nous l'avons choisi pour réaliser nos expériences.

Par ailleurs, un modèle cognitif doit être validé. Pour ce faire, il est nécessaire d'implémenter le modèle dans un programme tournant sur une machine, sans se contenter d'un modèle "papier". L'objectif de validation du modèle sur une machine peut alors s'inscrire dans un travail d'Intelligence Artificielle (IA).

Le but original de l'IA, créer une intelligence artificielle est loin d'être atteint. Les résultats précoces obtenus en IA ont pu laisser penser le contraire et donner lieu à des spéculations qui n'ont pas été vérifiées aujourd'hui. L'IA a découvert des obstacles. Actuellement, en comparaison avec un être humain, la machine apprend très peu, possède très peu de connaissances perceptives et très peu de connaissances de sens commun sur le monde réel. Le but général, au départ de l'IA s'est donc spécialisé sur des domaines concrets. Cette spécialisation est bonne. L'IA, comme les sciences cognitives, a tout à gagner à s'attaquer à des problèmes complexes ressemblant au monde réel même si ces problèmes sont spécifiques. A ce titre, le jeu de Go est un terrain d'expériences très riche pour l'IA. De fait, les machines sont faibles au jeu de Go. Celui-ci est un défi pour l'IA.

Avec ces convictions, le but initial de notre thèse s'est finalement précisé pour devenir le suivant : **élaborer un modèle cognitif du joueur de Go, le valider en l'implémentant sous forme de programme, pour expliciter des connaissances intuitives de l'être humain**.

Discussions, exemples, définitions et notations

Dans ce paragraphe, nous développons nos idées, donnons des exemples et proposons des définitions. Le plan de ce paragraphe est le suivant :

- Importance du monde réel
- La fonction essentielle du cerveau est non consciente
  - Les degrés de conscience
    - Conscient
    - Conscientisable
    - Intuitif
- Discussion sur l'Intelligence Artificielle
- Le choix du domaine pour la thèse : le jeu de Go
  - Le jeu de Go est analogue au monde réel en plus simple
  - Les connaissances du Go sont non conscientes
  - Au Go, les machines sont très faibles par rapport à l'homme
  - Contrairement à beaucoup de jeux, le jeu de Go résiste à l'informatisation
    - La force brute ne marche pas au Go
    - Simuler le comportement du joueur humain
- Qu'est ce que nous appelons un modèle cognitif ?
  - Des critères pour définir un modèle cognitif
  - Notre définition d'un modèle cognitif

Importance du monde réel

Nous avons écrit que **le système cognitif humain s'est développé pour comprendre le monde réel et à cause du monde réel**.

Évidemment, la polémique existe [Changeux & Connes 1992] :

- le monde réel n'est que le produit de notre cerveau. Après tout, nos perceptions sont dans notre cerveau. Si notre cerveau n'existait pas, nous ne percevrions plus le monde.
- le monde réel existe indépendamment de notre cerveau. Notre cerveau est un outil qui s'est développé à cause du monde réel, qui nous permet de l'approcher et le comprendre.

Il est impossible de prouver logiquement l'une ou l'autre des deux hypothèses. Ce serait peine perdue. Nous disons simplement que nous penchons pour la deuxième hypothèse. Dans cette optique, toute étude sur le fonctionnement du système cognitif humain doit donc s'intéresser au monde réel.

Notation

Pour exprimer une connaissance ou un concept associé à l'homme, nous utilisons des formes arrondies:

figure But-1

La fonction essentielle du cerveau est non consciente

Nous pensons que la fonction essentielle du cerveau humain est de nous faire apparaître le monde réel sous forme cohérente et simplifiée, de nous cacher sa complexité pour ne pas nous submerger. Les connaissances que l'homme utilise sont plus ou moins conscientes.

Les degrés de conscience

Nous allons préciser ce que nous appelons de façon simplifiée et non académique, conscient, conscientisable et intuitif.

Conscient :

Les faits que nous percevons sont conscients pour la plupart : si nous entendons un bruit, nous sommes conscients de ce bruit, si nous voyons une image nous en sommes conscients. Il en est de même des connaissances que nous pouvons formuler au cours d'un raisonnement: si nous effectuons une addition nous en sommes conscients, nous pouvons en parler à un moment donné. Pour exprimer un ensemble de connaissances conscientes de l'homme, nous utilisons le blanc:

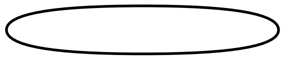


figure But-3

Conscientisable :

Nous pensons qu'une partie de ce qui est non conscient peut être rendu conscient par un travail, dit de conscientisation, par exemple la verbalisation [Vermersch 1991b]. Nous appelons cette partie non consciente, le conscientisable. Une expérience non consciente passée peut être conscientisée. Par contre, il semble difficile de conscientiser la façon dont notre appareil visuel nous fait apparaître les images que nous voyons. Une partie non consciente n'est pas conscientisable. Pour exprimer un ensemble de connaissances conscientisables de l'homme, nous utilisons le grisé clair:

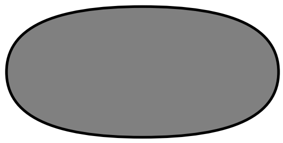


figure But-4

Intuitif :

Ce qui n'est pas conscient, ni conscientisable. Notre appareil perceptif est majoritairement intuitif, nous ne savons pas comment fonctionne notre système visuel, auditif, etc... mais notre appareil perceptif produit des images et des sons dont nous avons conscience. Ce que nous avons l'habitude de faire devient non conscient, nous le faisons de façon automatique sans y réfléchir.

Nous pensons que la plus grosse partie de l'activité de notre cerveau est non consciente ou intuitive. Pour exprimer un ensemble de connaissances intuitives (non conscientisables) de l'homme, nous utilisons le grisé foncé:

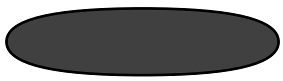


figure But-5

Discussion sur l'Intelligence Artificielle

Le but de l'IA paraît clair a priori : créer une machine intelligente. Nous avons la même définition de l'intelligence que le dictionnaire : l'intelligence est la faculté de comprendre [Larousse 1994]. Cette définition suit l'éthymologie du mot intelligence (**intellegere** = comprendre en latin). Nous supposons qu'elle sous-entend l'existence du monde réel car sans monde à comprendre, pas d'intelligence possible.

L'IA a obtenu très tôt de nombreux succès :
- Le General Problem Solver [Newell & Simon 1957],
- Le programme de Samuel qui a battu un champion américain aux Checkers[1] [Samuel 1959],
- Le système MYCIN qui détecte un agent infectieux dans le sang d'un patient et prescrit un traitement [Shortliffe 1974].

Ces succès précoces ont laissé libre cours aux spéculations [Allis 1994] :
- On a pensé que tous les problèmes seraient résolus à cause du mot "General" contenu dans le nom du programme de Newell, alors que le point fort du programme GPS était de trouver une bonne représentation de certains types de problèmes pour les résoudre "élégamment".
- On a pensé que la machine était devenue meilleure que l'être humain aux Checkers. Mieux même : que ce jeu était résolu. On a même pensé que tous les jeux allaient être résolus puisque le programme de Samuel "apprenait".
- On a pensé qu'on allait bientôt pouvoir remplacer des médecins.

20 à 35 ans plus tard :
- Il n'existe aucun résolveur de problème général.
- Ce n'est que récemment que les meilleurs programmes de Checkers rivalisent avec les meilleurs joueurs humains de Checkers [Schaeffer & al 1992].
- Les systèmes experts ont été limités par le goulot d'étranglement [Feigenbaum 1979] de l'acquisition des connaissances et les médecins peuvent soigner tranquillement.

En 1994, l'IA regroupe plusieurs entités classables par exemple comme suit :
- Construire une intelligence artificielle (le but initial) :
  - Déclarativité, Méta, Réflexivité.
- Imiter des facultés humaines intelligentes :
  - Perception, Sens commun, Apprentissage, Explication.
- Imiter, s'inspirer du monde réel :
  - Représentation Objet, Algorithmes Génétiques, Réseaux de Neurones, Systèmes Multi-Agent.
- Résoudre des problèmes concrets du monde réel :
  - Vision, Reconnaissance de la Parole, Langage Naturel.
- Formaliser, élaborer des théories :
  - Logique Floue, Logiques non-classiques.

Sous cet angle, l'Intelligence Artificielle apparaît en pleine effervescence. Le but général de départ est toujours poursuivi même si la création d'une intelligence artificielle est encore loin d'être faite. Des buts spécifiques, plus concrets sont apparus et nous pensons que cela est fructueux car ils sont en relation avec le monde réel. Il faut continuer de travailler sur des domaines qui permettent à la machine de combler son retard sur l'être humain : perception et apprentissage essentiellement. Le jeu de Go est un point de départ idéal pour inculquer à la machine ces deux facultés.

---

[1]Le jeu de dames, version anglaise, qui se joue sur un damier 8*8.

Notations

Pour exprimer une connaissance[1] ou un concept associé à la machine, nous utilisons des formes carrées :

figure But-2

Pour exprimer un ensemble de connaissances de la machine nous utilisons les rectangles comme sur la figure *But-6.*

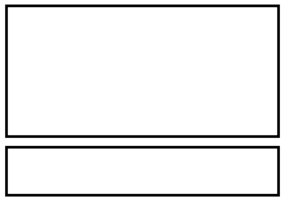


figure But-6

Les rectangles et les ovales symbolisent deux types de modèles. Les ovales symbolisent un **modèle cognitif** et les rectangles un **modèle computationnel**.

[1] Pour simplifier encore les comparaisons entre l'homme et la machine, nous utilisons un langage anthropomorphique bien pratique. Cependant, nous ne pensons pas que les machines aient des connaissances au même titre que l'homme.

Le choix du domaine: le jeu de Go

Le jeu de Go est analogue au monde réel en plus simple

Le jeu de Go est une simplification du monde réel [Langston 1988] et il en garde de nombreuses caractéristiques.

Les caractéristiques

Il possède des dimensions spatiales. Il a une dimension temporelle (la durée de la partie). Chaque point de l'espace (l'intersection) possède une propriété qui prend des valeurs (noir, blanc, vide). Chaque point du goban possède un voisinage. Le goban est perçu à plusieurs échelles de grandeur.

Les simplifications

Il est fini alors que le monde réel nous apparaît infini. Il est discret (19 lignes et 19 colonnes) alors que le monde réel nous apparaît continu. Il possède deux dimensions spatiales contre trois au monde réel. Une intersection possède une seule propriété, contrairement au monde réel où chaque point nous apparaît avec un grand nombre de propriétés (couleur, odeur, son, ...) . Le voisinage d'une intersection est fini : quatre intersections, alors que le voisinage d'un point du monde réel ne l'est pas. Le monde réel est perçu à différentes échelles alors que le goban ne possède essentiellement que deux échelles : l'une dite "locale" et l'autre "globale".

Les connaissances du Go sont non conscientes

L'homme, produit de millions d'années d'évolution, utilise ses connaissances intuitives pour jouer au Go. D'autre part, un joueur de Go qui apprend, transforme ses connaissances conscientes en connaissances non conscientes, comme dans beaucoup de domaines d'ailleurs. C'est pourquoi il est intéressant de faire des expériences sur les joueurs de Go pour tenter d'expliciter ces connaissances.

Une seule expérience de psychologie cognitive existe sur le jeu de Go

La seule expérience sur la nature des connaissances des joueurs de Go [Reitman 1976] a été faite pour retrouver les résultats obtenus sur le jeu d'Échecs [Chase & Simon 1973]. Les joueurs débutants ou confirmés ont les mêmes difficultés à reproduire de mémoire des positions ne ressemblant pas à celles habituellement rencontrées au cours des parties. Mais ils ont des performances très différentes pour reproduire des positions de parties réelles car le joueur confirmé utilise des connaissances sous forme de "chunk" pour mémoriser une position. De nombreuses expériences ont été faites sur les joueurs d'Échecs [De Groot 1946] pour modéliser leurs processus de pensée.

Au Go, les machines sont très faibles par rapport à l'homme

Le niveau d'un joueur de Go s'évalue avec le système des kyus et des dans, comme au Judo. Un débutant est disons 30ème **kyu**. Un joueur très moyen 10ème kyu. Jusqu'à 1er kyu, plus le joueur est fort plus son nombre de kyus est petit. A partir de 1er **dan**, plus le joueur est fort plus il possède de dans. Les meilleurs joueurs français sont 5ème dan. Les meilleurs joueurs européens sont 7ème dan. En Europe tous les joueurs sont amateurs. Sur l'échelle de niveau amateur, un kyu ou un dan d'écart correspond à un coup d'avance au début de la partie. On dit une "pierre" de handicap.

En Asie, il existe des joueurs professionnels évalués sur le même principe des kyus et des dans. L'échelle professionnelle est décalée par rapport à l'échelle amateur. Un premier dan professionnel correspond à un 6ème dan amateur. Trois dans d'écart sur l'échelle professionnelle correspondent à une pierre de handicap. Les professionnels ont généralement été formés depuis leur plus jeune âge. Les meilleurs joueurs professionnels participent à de grands tournois, comme au Tennis. Ils sont 9ème dan professionnel.

Le niveau des machines est très faible par rapport à celui des êtres humains. Une machine débutante commence ∞ kyu. Il est assez facile de faire un programme 30ème kyu moyennant un effort de programmation de quelques mois. Il existe quelques logiciels en freeware 30ème à 40ème kyu: Wally et Gnugo. Il est plus difficile de faire un programme 20ème kyu. Seul Goliath, le meilleur programme du monde, est 10ème kyu.

Contrairement à beaucoup de jeux, le jeu de Go résiste à l'informatisation

La force brute ne marche pas au Go

Dans sa thèse [Allis 1994], Allis passe les jeux de la "liste olympique" au crible de la complexité et donne le résultat actuel de la programmation de ces jeux. La "liste olympique" est une liste de jeux qui ont été jugés "intéressants" aux olympiades d'ordinateurs [Levy & Beal 1989] [Levy & Beal 1991]. Pour "montrer" que la force brute ne marche pas au Go, nous nous limitons à quatre jeux de la liste olympique : Othello, les Checkers, les Echecs et le Go.

La complexité de ces jeux

Allis définit précisément deux types de complexité. La complexité de l'espace des états et la complexité de l'arbre du jeu. Nous donnons ici des approximations de ces définitions suffisantes pour "montrer" que la force brute ne marche pas. La complexité E de l'espace des états est le nombre d'états légaux possibles au sens de la règle du jeu. La complexité A de l'arbre du jeu est le nombre de nœuds feuilles de l'arbre de recherche de la solution à partir de la position initiale. On calcule ce nombre avec B^L, où L est la longueur moyenne des parties réelles et B, une estimation du facteur de branchement de l'arbre (ou nombre de coups moyens engendrés sur une position). Pour les jeux de la liste olympique, Allis donne les complexités de ces jeux :

	Othello	Les Checkers[1]	Les Echecs	Go[2]
E	$3^{64} \approx 10^{30}$	10^{18}	10^{43}	$3^{361} \approx 10^{172}$
A	10^{58}	$2.8^{70} \approx 10^{31}$	$35^{80} \approx 10^{67}$	$200^{250} \approx 10^{575}$

Ces nombres sont trop grands pour nous parler. Ils "montrent" en première approximation que ces jeux sont de "complexité" croissante[3] et que le Go est le plus complexe.

En fait, cette argumentation est discutable. Sur 7-7, on trouverait $E = 10^{23}$ et $A = 30^{40} \approx 10^{59}$, on trouverait que le jeu de Go sur 7-7 est aussi complexe que Othello. Pourtant, il n'existe aucun programme de Go qui surclasse les joueurs humains comme c'est le cas à Othello. En prenant, le Go-moku[4] qui se joue sur 15-15, on pourrait argumenter que celui-ci, joué sur 19-19 est très complexe. Or ce jeu n'est pas aussi complexe puisqu'il est résolu et que la machine peut trouver une séquence gagnante [Allis 1994]. L'argumentation basée sur des chiffres ne suffit pas. Le jeu

[1] Ces chiffres prennent en compte le fait que beaucoup de coups sont forcés [Schaeffer & al 1992]
[2] Ces chiffres sont les nôtres. Nous pensons qu'une partie sur 19-19 dure en moyenne 250 coups et que 200 coups sont possibles en moyenne. Ce qui est faux au début et à la fin de la partie, mais vrai en moyenne.
[3] Othello semble plus complexe que les Checkers car, pour les Checkers Allis tient compte des coups forcés, et pour Othello les chiffres sont pris à l'état brut.
[4] Le jeu du morpion à quelques variantes près.

de Go est complexe non seulement à cause de la combinatoire, mais surtout par la difficulté pour trouver et mettre en œuvre une fonction d'évaluation précise et rapide.

Le résultat de la programmation de ces jeux

Othello

En 1994, plusieurs programmes d'Othello dépassent clairement le meilleur joueur humain. Le meilleur programme est Logistello de Michael Buro. Les programmes d'Othello utilisent de l'alpha-bêta, une base d'ouvertures, une recherche en fin de partie qui trouve le résultat du jeu après environ 36 coups.

Les Checkers

Le meilleur programme de Checkers est Chinook [Schaeffer & al 1992] qui est du niveau du champion du monde Marion Tinsley avec lequel il a presque fait jeu égal (2 victoires, 33 nuls, 4 défaites). Chinook utilise de l'alpha-bêta (20 coups à l'avance), une fonction d'évaluation, une base d'ouvertures et de fins de partie.

Les Echecs

Le programme qui a le meilleur classement ELO est Deep Thought (2550), ce qui le place entre la centième et la cent cinquantième place au monde. Les parties jouées entre Garry Kasparov, le champion du monde actuel, et Deep Thought ont toutes été gagnées par Kasparov. Les concepteurs du programme, qui travaillent sur une version parallèle : Deep Blue, pensent battre Kasparov. Deep Thought utilise de l'alpha-bêta (10 coups de profondeur), une fonction d'évaluation, un générateur de coups intégré au hardware, une base d'ouvertures [Anantharaman & al 1989]. En 1994, dans un tournoi de parties rapides, il est déjà arrivé qu'un programme (Chess Genius 2) batte Kasparov ou un très grand champion. Toujours en 1994, un tournoi, opposant six anciens champions des Etats Unis à huit logiciels, a donné l'avantage aux joueurs humains, par 29.5 points à 18.5. Mais il faut retenir l'excellente performance du programme WChess, invaincu, qui a obtenu 6 points (4 victoires et 2 nulles).

Go[1]

Les meilleurs programmes de Go, Goliath et Go Intellect sont 10ème kyu [Boon 1991] [Chen 1990]. Ce qui est un niveau très moyen sur l'échelle humaine. Ces programmes utilisent un savant équilibre entre les connaissances et la recherche arborescente. Ils s'inspirent tous du comportement humain. Il est impossible de faire autrement actuellement. La complexité du jeu de Go rend impossible une approche basée uniquement sur la recherche arborescente.

La seule approche envisageable actuellement est de s'inspirer du comportement du joueur humain

Les programmeurs de Go doivent s'inspirer du comportement humain pour développer leurs programmes, même si cela est difficile, et même si dans le cas des Echecs cela a donné des résultats [Pitrat 1977] [Wilkins 1980] moins bons que la force brute. L'IA doit continuer de développer des techniques d'acquisition de connaissances. Pour l'Intelligence Artificielle, le Go est à la fois un défi [Bradley 1979] et un domaine d'expérimentations idéal [Bouzy & Victorri 1992].

[1]Un état de l'art de la programmation du jeu de Go est présenté plus loin, avant la description de notre modèle.

Qu'est ce que nous appelons un modèle cognitif ?

Ce paragraphe présente des **critères** permettant de définir un modèle cognitif. Ensuite, nous présentons **notre définition** d'un modèle cognitif avec les approximations computationnelles faites.

Des critères pour définir un modèle cognitif

Critère "source" :

Un modèle est cognitif si la source qui a permis de créer le modèle est l' **être humain**.

Critère du "quoi" :

Il est possible de définir un modèle cognitif par le "quoi", c'est-à-dire par **ce que fait le modèle**. Si le résultat est identique à celui d'un être humain placé dans les mêmes conditions nous dirons que c'est un modèle cognitif suivant le critère du "quoi". Comme le souligne David Marr, il est fondamental de savoir ce que fait le modèle. En théorie de la vision [Marr 1982], l'auteur commence son livre par une question : quoi ? et il y répond immédiatement :*"What does it mean, to see? The plain man's answer would be, to know what is where by looking."*

Critère du "comment" :

Il est possible de définir un modèle cognitif par le "comment", c'est-à-dire par **la manière ou la façon de faire du modèle**. Si le modèle et l'être humain donnent les mêmes explications lorsqu'ils sont dans les mêmes conditions nous disons que le modèle est cognitif. Cela suppose qu'un modèle cognitif au sens du "comment" possède un module explicatif afin de comparer les explications du modèle avec celles des êtres humains [Ericsson & Simon 1980]. Le module explicatif le plus sommaire est la trace de l'exécution du programme qui implémente le modèle.

Critère du "combien" :

Un critère pratique important est le **temps de réponse du modèle**. En combien de temps le modèle fournit-il son résultat ? Si le modèle fournit son résultat en un temps trop long, comment le valider ? Un modèle qui donne le bon résultat en un temps "infini" est-il un modèle acceptable ? Dans des domaines complexes où le nombre d'états possibles est très grand, nous pensons que le nombre de modèles qui décrivent ce domaine est encore plus grand et qu'il est encore plus difficile de juger un tel modèle. La limitation sur le temps de réponse offre l'avantage d'être un critère concret et d'évaluer les résultats.

Ce critère a l'inconvénient de considérer certaines idées qui seraient bonnes si le support sur lequel s'exécute le modèle était plus puissant, comme mauvaises. Par exemple, en linguistique la théorie de Chomsky sur les grammaires n'est validée sur aucun support [Chomsky 1965], par contre celle de Winograd [Winograd 1972], plus concrète, a donné naissance a un programme informatique. Pourtant les deux théories apportent chacune une part déterminante à la connaissance que nous avons en linguistique et sur le traitement des langues naturelles. En théorie de la vision, David Marr [Marr 1982] pense clairement que le niveau théorique est fondamental : *"Although algorithms and mechanisms are empirically more accessible, it is the top level of computational theory, which is critically important from an information-processing point of view."* Nous le pensons aussi.

Critère du "support" :

La puissance du support

Il est fondamental de valider un modèle pour vérifier que les critères ci-dessus sont respectés. en le faisant tourner sur un support. La puissance de ce support montre que les critères du "combien" et du "support" influent sur le "quoi" et le "comment" du modèle.

Prenons un exemple de modèle simple dans le cas du Go. Supposons que le "quoi" soit le niveau en dan ou kyu du modèle créé, que le "comment" du modèle soit : on utilise une fonction d'évaluation simple du goban : +1 (respectivement -1) si l'intersection est noire (resp. blanche) ou vide mais entourée de voisines noires (resp. blanches). Imaginons que nous ayons une machine infiniment puissante pour faire de la recherche arborescente comme aux Echecs. En théorie, ce modèle aurait un "quoi" qui serait le meilleur niveau qui soit : disons 20 ème dan professionnel et un "comment" très simple.

Le problème est que nous ne disposons pas d'une machine infiniment puissante et que nous ne pouvons pas vérifier nos hypothèses théoriques. Avec les machines actuelles, en supposant qu'elles aient démarré à la naissance de l'univers, nous n'aurions exploré qu'une infime parcelle de l'arbre de recherche. Les critères du "combien" et du "support" de 1994 montrent que ce modèle est utopique. Il faut réduire nos ambitions et accepter de faire un modèle qui ne soit pas 20 ème dan mais plutôt 20 ème kyu : on influence le "quoi". Pour ce faire, il faut changer de méthode : on influence le "comment". Ainsi, la différence en pratique entre la théorie et la pratique est plus grande que la différence en théorie entre la théorie et la pratique...

La nature du support

La nature d'un modèle dépend aussi de **la nature du support sur lequel il tourne**. Un cerveau, dont le fonctionnement est un mystère, ne ressemble pas à une machine. Ce mystère entraîne nécessairement qu'un modèle qui tourne sur un cerveau, modèle cognitif au sens strict, soit différent d'un modèle qui tourne sur une machine actuelle ou modèle computationnel.

Nous pensons que la force du système cognitif humain est le niveau d'intégration qui existe entre la théorie du monde réel et son support : les neurones du cerveau. Nous pensons que la théorie du monde réel qui tourne dans notre cerveau est indissociable du cerveau. Les neurones et la théorie du monde réel qui tourne dessus sont nés ensemble et ont grandi ensemble. De ce point de vue, nous nous positionnons selon l'approche connexionniste des sciences cognitives. Malheureusement, nous pensons qu'il est impossible de valider empiriquement une telle affirmation (et nous n'avons pas cherché à le faire dans le cadre de notre thèse).

Si l'on veut continuer d'associer la définition d'un modèle cognitif à l'être humain, il faut dire aussi que le support du modèle doit être identique à celui du système cognitif humain : en l'occurrence le cerveau ! Mais nous ne disposons pas de cerveau pour valider notre modèle, seulement de machines[1] !

Comment faire ?

[1]stations de travail Sun.

Notre définition d'un modèle cognitif

Finalement, **le modèle cognitif doit** :

prendre sa source sur l'être humain,

produire des résultats comparables à ceux d'êtres humains,

fournir des explications comparables à celles données par des êtres humains,

avoir un temps de réponse suffisamment court.

Modèle cognitif ou computationnel ?

Nous dirons que le modèle sera valide lorsque son "quoi", son "comment" et son "combien" approcheront suffisamment le "quoi", le "comment" et le "combien" d'un être humain. Nous abandonnons l'idée de valider le modèle en relation avec la façon dont il tourne sur un cerveau humain.

Pour valider le modèle cognitif, nous utiliseront la machine et nous élaborerons un modèle computationnel qui l'approchera.

Notre volonté de valider notre modèle cognitif en utilisant une machine, afin de travailler concrètement, peut donner au lecteur l'impression que notre conviction se rapproche de l'approche cognitiviste [Fodor 1976] : analogie entre le fonctionnnement de l'ordinateur et celui du système cognitif humain avec la dualité esprit - cerveau analogue à la dualité logiciel - matériel. L'analogie existe mais il ne faut pas déduire trop de similarités entre le fonctionnement du système cognitif humain et celui de l'ordinateur. Par exemple, l'indépendance entre les états mentaux et l'activité neuronale à l'image de l'indépendance entre un logiciel et le matériel n'est qu'une analogie mais pas une réalité.

Notations

Le modèle cognitif sera présenté visuellement avec des ovales et le modèle computationnel avec des rectangles. Le rapprochement entre le modèle cognitif et le modèle computationnel sera fait à l'aide de flèches de correspondance comme le montre la figure *But-cognitif-computationnel.*

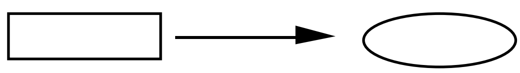


figure But-cognitif-computationnel

Bibliographie

[Allis 1994] - L. V. Allis - Searching for Solutions in Games and Artificial Intelligence - PhThesis - Vrije Universitat Amsterdam - Maastricht - Septembre 1994

[Anantharaman & al 1989] - Anantharaman T.S., Campbell M.S., Hsu F.H. - Singular extensions : Adding selectivity to brute-force searching - Artificial Intelligence, vol. 43, n°1, pp. 99-109 - 1989

[Boon 1991] - M. Boon - Overzicht van de ontwikkeling van een Go spelend programma - Afstudeer scriptie informatica onder begeleiding van prof. J. Bergstra - 1991

[Bouzy & Victorri 1992] - B. Bouzy, B. Victorri - Go et Intelligence Artificielle - Revue de l'AFIA n. 10 - Juillet 1992

[Bradley 1979] - M.B. Bradley - The game of Go, the ultimate programming challenge ? - Creative computing, 5, 3, p 89-99 - Mars 1979

[Chase & Simon 1973] - Chase W. Simon H. - Perception in Chess - Cognitive Psychology. - 4, 55-81 - 1973

[Changeux & Connes 1992] - J.P. Changeux, A. Connes - Matière à penser - Editions Odile Jacob - 1992

[Chen 1990] - K. Chen - Move decision process of Go intellect - Computer Go 14, spring 90

[Chomsky 1965] - N. Chomsky - Aspects of the Theory of Syntax - Cambridge, Mass. - MIT Press - 1965

[De Groot 1946] - A. D. De Groot - Psychological studies - tome 4 - Thought and choice in chess - Mouton the hague Paris - Traduction avec additions de la thèse de 1946.

[Ericsson & Simon 1980]- K.A. Ericsson H.A. Simon - Protocols analysis, verbal reports as data - MIT Press - Cambridge, Massachussetts, London, England

[Feigenbaum 1979] - E.A. Feigenbaum - Themes and Case Studies of Knowledge Engineering - Expert Systems in the Micro-electronic age (ed. D. Michie) - pp. 3-25, Edinburgh University Press, Edinburgh, Scotland

[Fodor 1976] - J. Fodor - Langage of thought - 1976

[Langston 1988] - R. Langston - Perception in Go as a problem AI - Computer Go 6, spring 88

[Larousse 1994] - Petit Larousse - 1994

[Levy & Beal 1989] - D.N.L. Levy, D.F. Beal (eds.) - Heuristic programming in Artificial Intelligence : the first computer olympiad - Ellis Horwood, Chichester, England - 1989

[Levy & Beal 1991] - D.N.L. Levy, D.F. Beal (eds.) - Heuristic programming in Artificial Intelligence : the second computer olympiad - Ellis Horwood, Chichester, England - 1991

[Marr 1977] - D. Marr - A personal view - Artificial Intelligence, vol. 9, pp. 37-48 - 1977

[Marr 1982] - D. Marr - Vision - San Fransisco, Freeman & co - 1982

[Newell & al 1957] - A. Newell, J.C. Shaw, H.A. Simon - Preliminary description of General Problem Solving Program-I (GPS-I) - Report CIP Working Paper 7

[Pitrat 1977] - J. Pitrat - A chess combination program which uses plans - Artificial intelligence 8, 275-321 - 1977

[Reitman 1976] - Reitman J.S. - Skilled perception in Go: deducing memory structures from inter-response times - Cognitive Psychology 8, 3 - 1976

[Samuel 1959] - Samuel A.L. - Some studies in machine learning using the game of checkers - IBM Journal of research and development - Vol. 3, n° 3 - 1959

[Schaeffer & al 1992] - J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, D. Szafron - A World Championship Caliber Checkers Program - Artificial Intelligence, vol. 53, pp. 273-289 - 1992

[Shortliffe 1974] - E.H. Shortliffe - MYCIN : Computer-based Medical Consultations. PhD thesis, Stanford University, Stanford, CA, 1974

[Vermersch 1991b]- Vermersch P. - Les connaissances non conscientes de l'homme au travail - Le journal des psychologues 84 - 1991

[Wilkins 1980] - D. Wilkins - Using plans and pattern in chess - Artificial intelligence 14, 165-203 - 1980

[Winograd 1972] - T. Winograd - Understanding Natural Language - New York - Academic Press - 1972

PARTIE 2 : MÉTHODE

Dans cette partie nous présentons l'approche générale utilisée au cours du travail de recherche pour aboutir à notre modèle.

Le plan de cette partie est le suivant :

Les enseignements tirés de notre première modélisation
Les verbalisations
L'implémentation sur machine
L'utilisation de domaines voisins
L'état de l'art de la programmation du jeu de Go

LES ENSEIGNEMENTS TIRÉS DE NOTRE PREMIÈRE MODÉLISATION

Les résultats

Nous avons construit notre modèle cognitif en analysant les idées que nous avions en tant que joueur de Go et celles qui sont écrites dans les livres de Go :

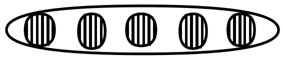


figure Méthode-1

Nous avons formalisé et implémenté ces idées sur une machine pour élaborer un modèle computationnel :

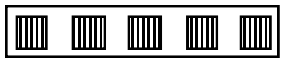


figure Méthode-2

Nous nous sommes rendu compte que les résultats étaient très mauvais :

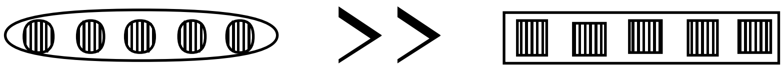


figure Méthode-3

Pourquoi la machine, dans laquelle nous avons l'impression d'avoir mis les connaissances de joueurs de Go, n'a-t-elle pas les mêmes résultats que l'homme ? Comment expliquer la dramatique inégalité de la figure *Méthode-3*.

Les enseignements

Les résultats de notre première modélisation ont renforcé nos convictions sur le degré de conscience qu'un être humain a de ses connaissances. L'être humain possède :

des connaissances de haut niveau, conscientes,
des connaissances acquises par l'expérience, non conscientes mais conscientisables,
des connaissances intuitives non conscientisables.

Ce qui est représenté par la figure *Méthode-4*.

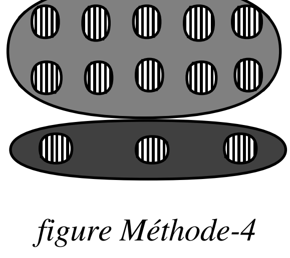


figure Méthode-4

Nous pensons que ceci est la cause principale du mauvais résultat de la figure *Méthode-3* que nous préférons alors remplacer par la figure *Méthode-5*:

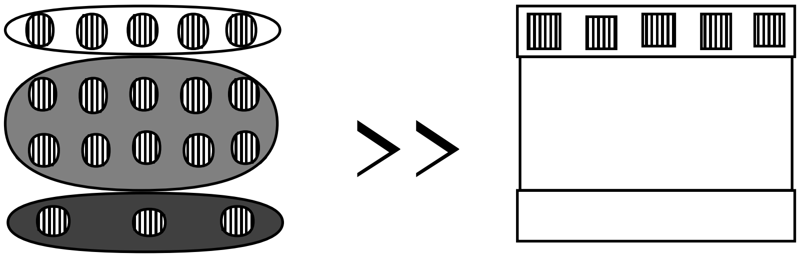


figure Méthode-5

Dans notre première modélisation représentée par la partie droite de la figure *Méthode-5*, la machine ne possède aucun équivalent de connaissances non conscientes du joueur de Go et ses résultats sont logiquement inférieurs.

Pour travailler nous avons fait deux hypothèses sur la forme globale des modèles cognitif et computationnel que nous cherchions. Il nous fallait traduire le fait que les mots, utilisés dans les livres de Go ou dans l'expression de nos idées, ont une "profondeur" cachée, contiennent des implicites et que leur évidence apparente est illusion [Wittgenstein 1951] .

Les connaissances du Go explicitées et implémentées sur machine se décomposent en plusieurs niveaux d'abstraction :

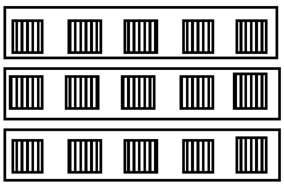


figure Méthode-6

Au niveau 0, il existe des concepts élémentaires.
Au niveau 1, il existe des concepts élaborés avec le niveau 0.
Au niveau n, il existe des concepts élaborés avec le niveau n-1.

Les connaissances du Go d'un niveau sont reliées par des relations de voisinage. Pour élaborer une connaissance du niveau N+1, nous utilisons la connaissance homonyme du niveau N et ses connaissances "voisines" du niveau N.

Un concept Y(n+1) du niveau n+1 est un affinage d'un concept Y(n), (il porte le même nom) avec les concepts "voisins" de X(n), Z(n) comme sur la figure *Méthode-7*.

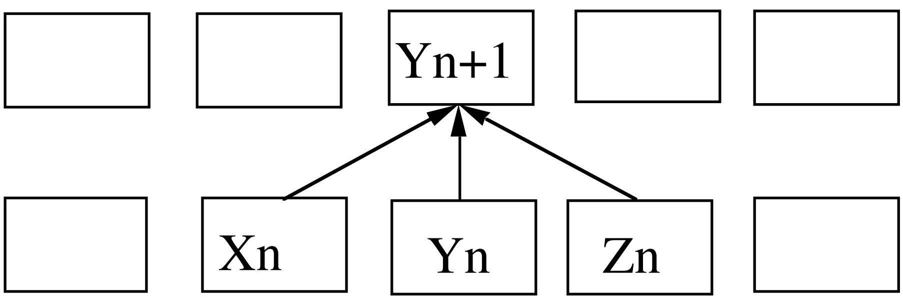


figure Méthode-7

Ces deux hypothèses sur la forme des modèles cognitif et computationnel que nous cherchons sont guidées par l'approche cognitiviste classique en sciences cognitives [Fodor 19] qui utilise largement l'analogie entre l'homme et la machine. L'informaticien peut interpréter les figures *Méthode-6* et *Méthode-7* comme une architecture logicielle avec un découpage en modules qui s'utilisent les uns les autres.

Exemple

Pour comprendre cet exemple nous conseillons au lecteur de se reporter au chapitre qui décrit les niveaux "zéro" et "élémentaire" de notre modèle pour avoir un exemple de ce que l'on peut appeler le jeu de la chaîne et jeu de la connexion.

Supposons que:

X(0) soit la construction d'une chaîne
Y(0) soit la définition statique de la connexion par une base de règles
X(1) soit le jeu de la chaîne
Y(1) soit le jeu de la connexion de deux chaînes
X(2) soit le jeu de la chaîne qui utilise la possibilité de la chaîne de se connecter avec une chaîne voisine.
Y(2) soit la connexion de deux chaînes avec la possibilité de capturer une chaîne ennemie pour se connecter.

X(1) utilise X(0)
Y(1) utilise X(0) et Y(0)
X(2) utilise X(1) et Y(1)
Y(2) utilise X(1) et Y(1)

X(0) et Y(0) sont "voisins".

Les définitions ci-dessus sont des exemples. Le lecteur peut imaginer beaucoup d'autres possibilités pour définir des jeux de la chaîne et des jeux de la connexion plus ou moins évolués. En tout cas, il faut comprendre que l'on ne peut pas définir brutalement un concept ex-nihilo indépendamment du reste. La définition d'un concept se fait par étapes en utilisant des concepts voisins de niveau inférieur.

LES VERBALISATIONS

Introduction

Pour effectuer des expériences de psychologie cognitive, on peut mesurer différents types de données issues de ces expériences. A chaque fois ces données sont des observables externes: les temps de réponse [Bonnet 1988], les mouvements oculaires [Levy-Schoen 1988] ou les verbalisations [Caverni 1988]. Dans ce paragraphe nous discutons seulement des verbalisations. D'abord, nous présentons un état de l'art sur l'utilisation des verbalisations en psychologie cognitive . Ensuite nous montrons que dans un domaine complexe comme le jeu de Go où le modèle cognitif est très abstrait et structuré, les verbalisations, qui expriment les choses à plat, peuvent servir de point de départ mais cachent l'abstraction et donc ne suffisent pas pour élaborer le modèle.

L'état de l'art

Les expériences de psychologie cognitive utilisant les verbalisations ont été nombreuses dans les années soixante et soixante-dix. Mais l'utilisation de verbalisations a été explicitement contestée en psychologie cognitive [Nisbett & Wilson 1977]. Nous citons les objections faites aux verbalisations selon l'état de l'art en psychologie cognitive [Caverni 1988]. Puis nous citons le modèle de Ericsson et Simon [Ericsson & Simon 1980]. Ensuite nous citons dans quelles conditions les verbalisations peuvent servir d'observables au fonctionnement cognitif humain [Caverni 1988] [Vermersch 1991a].

Les objections aux verbalisations en psychologie cognitive

En 1977, un article de Nisbett et Wilson a créé une polémique en psychologie cognitive [Nisbett & Wilson 1977]. Les auteurs ont montré que l'interprétation des résultats d'expériences basées sur des verbalisations est très difficile. Elle dépend des conditions imposées par l'expérimentateur aux sujets et peut amener facilement à des conclusions contraires. En réponse à leur article, plusieurs articles [Smith & Miller 1978] [White 1980] démontrant la position inverse sont parus dans les années qui suivirent.

Caverni [Caverni 1988] pose clairement la question :*"Dans quelle mesure et à quelle condition peut-on rendre compte de l'activité cognitive d'un sujet dans une tâche ou une situation données, à partir d'une verbalisation du sujet ?"*

La verbalisation modifie l'exécution de la tâche et de la performance. Gagne et Smith ont montré sur des expériences sur la tour de Hanoi que la verbalisation améliore la performance [Gagne & Smith 1962]. Karpf que la verbalisation ne produit pas d'effet [Karpf 1973]. Ericsson que la verbalisation diminue la performance [Ericsson 1975]. Pour expliquer ces trois résultats apparemment différents, la conjecture est la suivante : si la procédure s'exécute en code verbal, la verbalisation "met en place" la procédure et améliore la performance, sinon, la verbalisation perturbe la procédure.

Les processus cognitifs ne sont pas accessibles par verbalisation. Les protocoles verbaux ne sont pas directement exploitables et sont mis en forme par l'expérimentateur. L'objectivité de cette mise en forme est contestée. Quand les verbalisations sont correctes, Nisbett et Wilson pensent que les verbalisations ne sont pas nécessairement le fruit d'une introspection mais plutôt de l'utilisation de "théorie causales a priori" que les sujets connaissent grâce à la culture ambiante [Nisbett & Wilson 1977].

Le modèle de fonctionnement cognitif et le modèle de production verbale de Ericsson et Simon

Ericsson et Simon pensent que **la validité des verbalisations doit se référer à un modèle préexistant à l'expérience** [Ericsson & Simon 1980]. Ce modèle est fait de deux parties : un modèle cible concernant la tâche considérée effectuée par les sujets et un modèle de la production verbale. Ils supposent que le modèle utilise trois types de mémoire : des mémoires sensorielles (MS), des mémoires à court terme (MCT), des mémoires à long terme (MLT) et deux types de processus : des processus automatiques et des processus contrôlés.

L'avantage des verbalisations.

Les années soixante et soixante-dix ont été l'époque du foisonnement d'expériences utilisant les verbalisations. Le doute mis par Nisbett et Wilson en 1977 a permis d'avoir du recul sur les verbalisations et de connaître les précautions d'emploi de celles-ci. Toutefois, cela ne veut pas dire qu'il faille abandonner les verbalisations. Bien au contraire ! Il serait impossible de travailler sans elles. Par approximations successives, les verbalisations permettent d'obtenir des informations cruciales et précises sur le déroulement d'une tâche effectuée par un sujet. Par un questionnement perfectionné, on peut aider un sujet à expliciter des connaissances qu'il a utilisées pour effectuer une tâche [Vermersch 1991a], même si ces connaissances sont non conscientes [Vermersch 1991b].

Conclusion

Beaucoup d'objections sont faites à l'utilisation de protocoles verbaux [Nisbett & Wilson 1977]. Ericsson et Simon ont montré qu'il est nécessaire d'avoir un modèle a priori que les verbalisations viennent conforter [Ericsson & Simon 1980]. L'objection principale est l'impossibilité d'extraire un modèle cognitif directement à partir des verbalisations [Caverni 1988]. Toutefois, les verbalisations associées à des questions appropriées offrent des renseignements précieux sans lesquels, il serait sûrement impossible d'extraire un modèle cognitif [Vermersch 1991a]. Nous sommes globalement d'accord avec ces appréciations sur les verbalisations. Nous le montrons au paragraphe suivant au travers de notre travail.

Les verbalisations dans notre travail

Dans ce paragraphe, nous présentons **notre point de vue** sur les verbalisations en nous appuyant sur deux exemples qui ont été la source de verbalisations de joueurs de Go :

la **capture d'une pierre** sur la deuxième, troisième ou quatrième ligne du goban,
la **connexion** et la **séparation**.

Ensuite, nous montrons comment les verbalisations permettent de **conscientiser des connaissances**, cela suivant le niveau des joueurs débutants ou confirmés et en utilisant la notation picturale sur le degré de conscience d'une connaissance introduite précédemment.

Deux exemples

La capture d'une pierre sur la deuxième, troisième ou quatrième ligne du goban

Pour fixer les idées, supposons que nous ayons trois joueurs de Go sous la main, un débutant, un joueur moyen et un joueur fort et que nous puissions les interroger sur des positions de Go. Nous les interrogeons sur les positions des figures *Méthode-verbe-A*, *Méthode-verbe-B* et *Méthode-verbe-C* pour savoir si la pierre blanche en atari est capturée ou non si le Blanc commence (si Noir commence, il prend la pierre blanche en un coup évidemment).

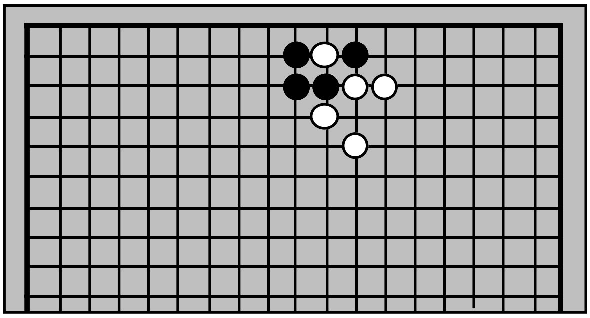


figure Méthode-verbe-A

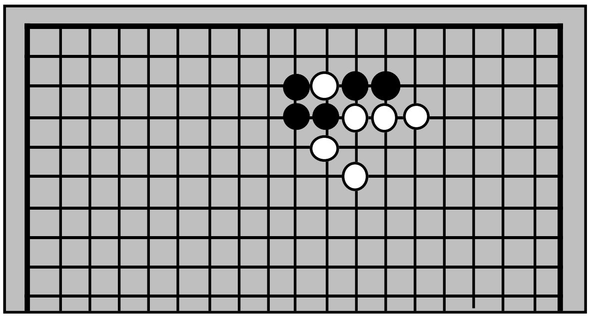


figure Méthode-verbe-B

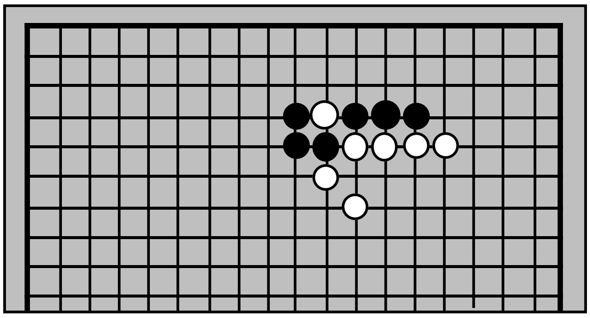


figure Méthode-verbe-C

Le joueur débutant dit : *"Sur A, la pierre blanche est capturée, sur B et C, elle ne l'est pas."*
Le joueur moyen dit : *"Sur A et B, la pierre blanche est capturée, sur C, elle ne l'est pas."*
Le joueur fort dit : *"Sur A, B et C, la pierre blanche est capturée."*

Comment sans rien connaître du jeu de Go, un expérimentateur va-t-il pouvoir engendrer un modèle cognitif à partir de ces seules verbalisations ?

Cela paraît impossible.

Nous pensons que les seules verbalisations ne suffisent pas à construire le modèle.

Si l'expérimentateur connaît le jeu de Go, il peut créer un modèle. Si l'on reprend la troisième hypothèse sur la forme de notre modèle cognitif, à savoir qu'un modèle est une hiérarchie de concepts où chaque concept est un raffinement du concept du dessous en fonction des concepts voisins, on peut supposer que des modèles possibles engendrés par notre expérimentateur seront les suivants :

Modèle du débutant :

X(0) la construction d'une chaîne Y3(1) le jeu de la capture d'une chaîne tel que la chaîne est stable si elle a 3 libertés au moins.

Modèle du joueur moyen :

X(0) la construction d'une chaîne Y4(1) le jeu de la capture d'une chaîne tel que la chaîne est stable si elle a 4 libertés au moins.

Modèle du joueur fort :

X(0) la construction d'une chaîne Y4(1) le jeu de la capture d'une chaîne tel que la chaîne est stable si elle a 4 libertés au moins. Z(1) le jeu de la connexion de deux chaînes X(2) la construction d'un groupe avec des chaînes reliées par des connexions Z(1) X(3) la construction d'un groupe avec des groupes X(2) reliés par des chaînes capturées Y(2). T(4) la qualification d'un groupe[1] Y(5) le jeu de la capture d'un groupe

Les trois modèles sont inclus les uns dans les autres.

Imaginons que l'expérimentateur soit plus paresseux et n'ait fait que 3 expériences au lieu des 9 précédentes : il n'interroge le débutant que sur A, le joueur moyen que sur B et le joueur fort que sur C. Il obtient des verbalisations identiques : *"La pierre blanche est capturée."*.

A priori, nous pensons que dans le cas du Go, la verbalisation cache l'abstraction. Le modèle sur les groupes du joueur fort, abstraction du modèle sur les chaînes, produit la même verbalisation que le modèle sur les chaînes du joueur débutant ou moyen. La verbalisation est en surface et le modèle est en profondeur. Évidemment, nous pouvons reprocher à l'expérimentateur d'être paresseux. Nous supposons maintenant que l'expérimentateur est moins paresseux. Il peut demander des explications aux joueurs. Sur la position A, le débutant, le joueur moyen et le joueur fort donnent la même explication car le modèle du débutant suffit. *"Parce que si Blanc joue 1 dans la figure Méthode-parce-que-A, la séquence 1-2-3-4 capture la chaîne blanche."*

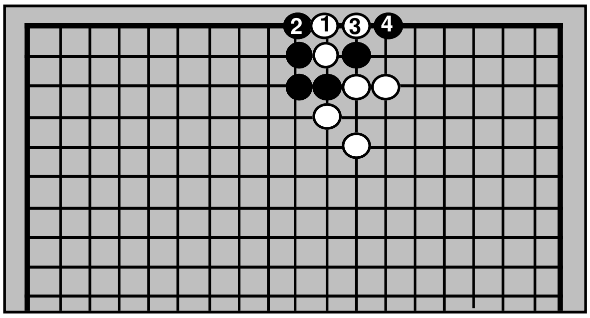


figure Méthode-parce-que-A

Sur la position B, le joueur moyen et le joueur fort donnent la même explication car le modèle du joueur moyen suffit. *"Parce que si Blanc joue 1 dans la figure Méthode-parce-que-B, la séquence 1-2-3-4-5-6-7-8 capture la chaîne blanche."*

[1]Nous ne rentrons pas dans les détails.

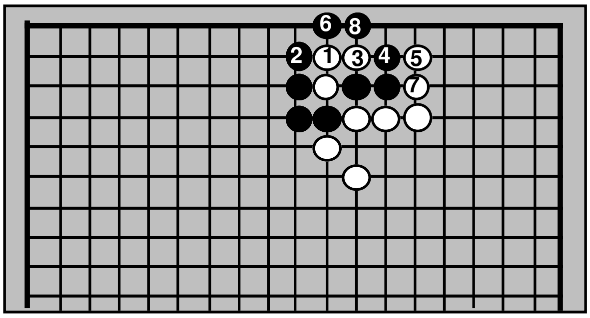


figure Méthode-parce-que-B

Sur la position B, le débutant donne l'explication suivante : *"Quand Blanc joue 1 sur la figure Méthode-parce-que-B, la chaîne blanche a 3 libertés elle est stable"*. Sur la position C, le joueur fort donne l'explication suivante. *"Parce que si Blanc joue 1 dans la figure Méthode-parce-que-C, Noir peut jouer la séquence de 1 à 26 qui capture la chaîne blanche."*

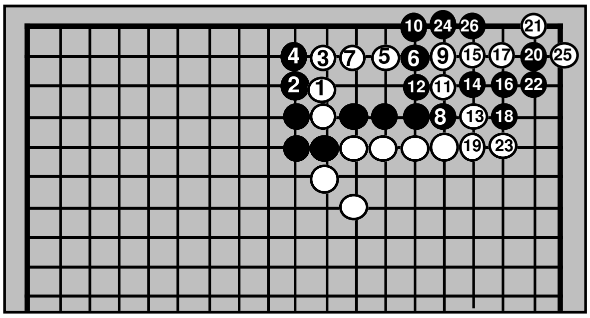


figure Méthode-parce-que-C

Sur la position C, le joueur moyen donne l'explication suivante. *"Parce que si Blanc joue 1 et 3 dans la figure Méthode-parce-que-3, la chaîne blanche a 4 libertés, elle est stable"* Avec ces explications et en associant un concept à chaque mot, l'expérimentateur identifie déjà une première liste de concepts tels que *capture*, *stabilité.*

En questionnant le joueur fort :
"Pourquoi joue-t-on 5 qui n'est pas au contact de la chaîne blanche ?"
"Pour s'étendre et occuper l'espace."
"Pourquoi 6 est-il joué loin de la chaîne blanche 1-3 ?"
"Parce que 1, 3, 5 constituent un groupe et qu'il faut capturer le groupe en entier."
"Un groupe ?"
"La pierre 5 est indéconnectable de la chaîne 1, 3."
On peut expliciter d'autres concepts : *proximité*, *occupation de l'espace*, *groupe*, *connexion.*

Les verbalisations servent de point de départ pour identifier des concepts.

Nous pensons que les verbalisations permettent de dégager grossièrement les principaux concepts utilisés par les sujets au cours de la tâche exécutée. En faisant une simple correspondance entre les mots et les concepts du modèle, on obtient déjà une première liste grossière du modèle. Par contre il est difficile d'aller beaucoup plus loin dans l'utilisation des verbalisations dans un domaine aussi complexe que le Go. Pour élaborer un modèle, il ne suffit pas d'expliciter une liste de concepts mais il faut placer les concepts les uns par rapport aux autres.

Il est très difficile de trouver l'agencement des concepts à partir des seules verbalisations.

Enfin, les verbalisations sont ici des commentaires de parties explicatifs. Qu'ils soient formulés après coup[1] ou concomitants, nous pensons que le mécanisme explicatif est différent du mécanisme moteur des actions du joueur de Go (le choix des coups du joueur). On ne peut pas créer un modèle d'un domaine complexe en transformant les explications en règles qu'un moteur utiliserait. Par exemple, si un joueur de Go dit : “J’ai joué ici parce que ce groupe était encerclé et que je voulais le faire vivre”. Cela ne signifie pas nécessairement qu’il existe dans le modèle qui lui correspond une règle du type: “SI un groupe est encerclé ALORS le faire vivre”.

L’explication et l’action sont deux choses distinctes.

Nous pensons que le mécanisme moteur engendrant l’action est autre. Nous pensons, en plus, que ce mécanisme moteur nous est caché.

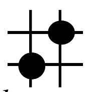
La connexion et la séparation[2] de l'adversaire

Ce paragraphe montre comment à partir de 3 dessins comportant 4 intersections seulement, un cogniticien très curieux (trop?), ENFANDELEFAN, et un joueur de Go, CROCODILE, très sûr de lui (trop?), peuvent arriver à découvrir, non sans mal, un concept caché - le concept de séparation - derrière un autre - celui de connexion. Pour faciliter la compréhension de l'entretien, un mathématicien qui sait déjà tout, SERPENPITON, nous donne amicalement son avis. Les verbalisations sont imaginées. Le concept de séparation caché derrière celui de connexion est par contre un phénomène bien réel.

Notations :

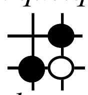
"CROCODILE s'exprime comme il peut."
ENFANDELEFAN pose des questions à propos de 3 dessins.
(SERPENPITON pense sans rien dire)

Une histoire comme ça[3]



kosumi

Que vois-tu ?
*"C'est **connecté** !"*
(Il faut entendre connecté au sens du jeu de la connexion, pas 4-connecté)
***Connecté** ?*
"Oui parce que si Blanc joue d'un côté, Noir joue de l'autre"
(Il a raison)
" et réciproquement".
(Il l'air de connaître quelques rudiments de mathématiques!)
(Avec le kosumi seul, il est difficile d'arriver à quelque chose. Dessin suivant, SVP)



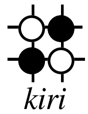
hane

Et là, que vois-tu ?

---

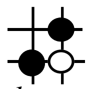
[1]Par un modèle de production verbale selon le schéma Ericssonien.
[2]Le paragraphe sur la morphologie mathématique permet de montrer au lecteur ce que nous appelons séparation et son utilité vis-à-vis de la connexion. Nous conseillons au lecteur de s'y reporter.
[3]Nous nous sommes inspiré de l'histoire de l'enfant d'éléphant [Kipling]

"On peut connecter !".
(Le concept de séparation est décidément bien caché.)
Peut ?
"..."
*(Il pose la mauvaise question, il risque de s'engager loin. Il suffirait de dire * avec la théorie de Conway et ce n'est pas cela qui nous intéresse ici)*
On ?
(Toujours mauvais)
"Ca dépend à qui c'est de jouer, si blanc joue, il peut couper"
Couper ?
(Bravo! C'est le bon chemin. Ils vont mettre le temps mais il vont y arriver...)
"Ben oui déconnecter quoi"
(Aïe! Ouillouillouille, on fait marche arrière. Vite un autre dessin.)



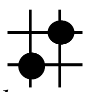
kiri

Et là ?
*"C'est foutu, c'est **coupé**."*
(Le concept de séparation est toujours caché, mais ça chauffe...)
***Coupé** ?*
"Ben oui complètement déconnecté quoi"
(déconnecté, déconnecté il ne sait dire que ça...)
Complètement ?
*"Ben oui, quoiqu'il arrive, il est **séparé**."*
***Séparé** ?*
(Il a dit le mot, c'est encourageant, mais le concept n'est pas encore identifié)
"Oui déconnecté."
(On va tourner en rond)
- silence -
*"**Blanc** aussi est déconnecté"*
*Faites excuses ! Vous avez dit **Blanc** ? Ainsi les deux pierres noires qui étaient connectées sur la première diapo ne seraient pas connectées et les blanches non plus ? Comme c'est rigolo !*
(Tirez mon jeune ami, tirez !)
*"Ben oui, **les pierres noires séparent** les pierres blanches, quoi, c'est évident."*



hane

*Évident ? Mais alors sur ce dessin, que font l**es pierres noires** ?*
(Ca chauffe de plus en plus)
*"Ben, elles **séparent** toujours."*
*Elles **séparent** toujours quoi ?*
"La pierre blanche de l'autre intersection."
L'autre intersection ?
*"Oui, même s'il n'y a **rien** sur l'autre intersection, ça sépare quand même."*
***Rien** ?*
"Pas de pierre noire ni blanche."



kosumi

Et alors sur ce dessin qu'est-ce qu'elles font les pierres noires ?
*"Ben elles **séparent** en deux, aussi."*
T'aurais pu le dire plus tôt !
(C'est ainsi que de la confusion générale, surgit le Concept de Séparation)
"Je ne m'en rendais pas compte!"

Commentaires :

Dans cet exemple, il est intéressant de noter que le chemin "kosumi" -> "hane" -> "kiri" -> "hane" -> "kosumi" semble nécessaire pour faire expliciter au joueur de Go le concept de séparation. Le joueur de Go est incapable d'expliciter le concept de séparation sur le "kosumi" seul, bien que le concept de séparation y soit présent. Le "kosumi" est associé trop fortement au concept de connexion.

Cette fiction montre **les difficultés pour découvrir un concept caché à partir de verbalisations**. Précisons que :

Les dessins sont choisis après avoir trouvé et en fonction du résultat, pour expliquer au lecteur.

Il y a seulement 3 dessins de 4 intersections. L'information contenue est très faible.

ENFANDELEFAN pose de bonnes questions. L'histoire pourrait tenir sur plusieurs pages sans aboutir à quoi que ce soit.

Dans la réalité c'est encore plus difficile :

On ne sait pas ce que l'on cherche, puisqu'on ne l'a pas trouvé.

Il y a beaucoup de dessins possibles avec beaucoup d'intersections.

On pose les questions au hasard.

Dans le cas de la découverte du concept de séparation caché derrière celui de connexion, nous pensons que la connaissance de domaines voisins tels que la morphologie mathématique nous ont plus aidé que les verbalisations. Les verbalisations arrivent après coup comme une illusion d'explication.

Conscientiser des connaissances sur le jeu de Go

La figure *Méthode-4* montre visuellement la première hypothèse que nous avons faite sur notre modèle cognitif suivant le degré de conscience des connaissances utilisées. Dans cette hypothèse, les interrogations de joueurs de Go sont des expériences de conscientisation [Vermersch 1991b]. Le but de ce paragraphe est de discuter sur l'évolution du degré de conscience des connaissances au fur et à mesure des expériences et quels types de connaissances sont explicités par des expériences sur des joueurs débutants et sur des joueurs confirmés. Avant de le faire, nous renforcons le fait qu'un joueur humain utilise majoritairement des connaissances sous forme d'automatismes en faisant des remarques sur les parties rapides.

Observation de parties rapides

Les joueurs de parties rapides sont dans la position de la figure *Méthode-rapide* :

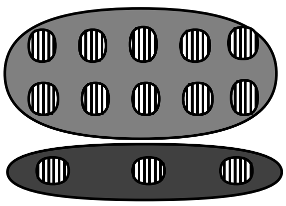


figure Méthode-rapide

Ils utilisent beaucoup moins de connaissances conscientes qu'en partie lente. Pourtant leur niveau ne baisse que de quelques dans ou kyus (2 ou 3). Cette remarque renforce l'idée que les joueurs utilisent des automatismes non conscients conscientisables.

Observation de joueurs débutants

Les débutants sont dans la position suivante:

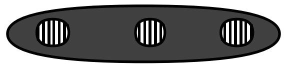


figure Méthode-8

Ils ne disposent pas de connaissance sur le jeu de Go mais ont des connaissances intuitives sur le monde réel. Ecouter les explications à voix haute de débutants en train de jouer permet de suivre leur apprentissage :

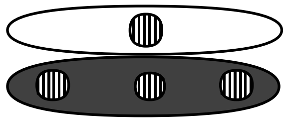


figure Méthode-9

Les joueurs débutants acquièrent très vite des automatismes sur le jeu de Go:

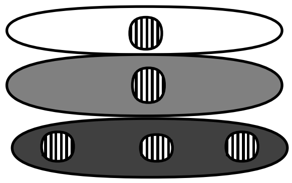


figure Méthode-10

L'apprentissage est rapide. Les débutants ne restent pas longtemps débutants. Les concepts rendus apparents par les expériences deviennent des concepts de niveau de plus en plus élevé.

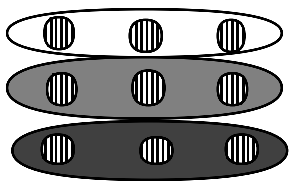


figure Méthode-11

Les expériences sur les débutants offrent l'avantage d'expliciter des connaissances de niveau suffisamment bas. Sur la figure *Méthode-12*, le bas du rectangle blanc du milieu se remplit peu à peu (le haut était déjà rempli par notre première modélisation) :

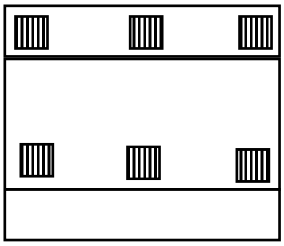


figure Méthode-12

Les débutants sont très intéressants à étudier car ils permettent d'expliciter des connaissances de bas niveau au fur et à mesure qu'ils acquièrent leurs premières connaissances. Par contre, la durée de vie du débutant est courte. En moyenne, après une dizaine de parties sur 9-9 à 4 ou 5 pierres de handicap contre un joueur confirmé qui lui commente chaque partie, le débutant complet (30 ème kyu) passe à 3 pierres et n'est plus un débutant complet. S'il est capable de gagner à 3 pierres sur 9-9, il est conseillé de passer sur un goban 13-13, et ainsi de suite en diminuant le handicap et en augmentant la taille du goban. Quand le joueur débutant a fait quelques parties sur 19-19, il n'est plus vraiment débutant mais un joueur faible (20 ème kyu). Les débutants sont précieux mais éphémères.

Observation de joueurs confirmés

Les joueurs confirmés ont l’avantage de disposer de beaucoup de connaissances sur le jeu de Go Par contre, ils ont déjà rendu non conscientes la plupart de ces connaissances :

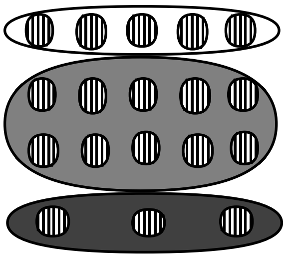


figure Méthode-13

Les expériences sur les joueurs confirmés, faites à haute voix, ont permis de dévoiler des connaissances du niveau juste inférieur comme sur la figure *Méthode-14* :

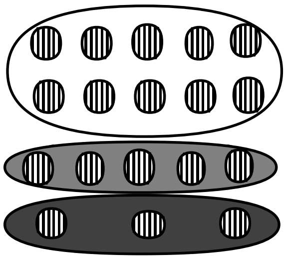


figure Méthode-14

Il faut préciser que cette découverte est difficile et nécessite un travail de conscientisation [Vermersch 1991a]. L'expérimentateur est important. Il doit poser des questions au sujet pour l'aider à parler de concepts intéressants. Mais il ne doit pas être trop directif. Des questions vides de sens, qui ont pour seul but de replacer le sujet dans des conditions proches de celles où il a opéré, ou la répétition du dernier mot cité par le sujet aident le sujet en le dirigeant le moins possible. Ces expériences ont permis de remplir le haut du rectangle blanc du milieu de la figure *Méthode-15* :

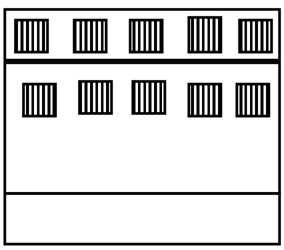


figure Méthode-15

Ces expériences sont difficiles pour les sujets qui doivent faire un effort pour expliquer plus qu’ils ne le voudraient. D’autre part, après des expériences répétées, les sujets jouent moins bien au Go car ils perdent une partie de leurs automatismes.

Conclusion

Pour expliquer les mauvais résultats d'une première modélisation, nous avons renforcé nos convictions :

> Les connaissances du joueur de Go humain ont des degrés de conscience différents que nous classifions en trois catégories :

> les connaissances conscientes,
> les connaissantes non conscientes conscientisables,
> les connaissances intuitives.

Et nous avons recherché un type de modèle cognitif particulier où :

> Les concepts explicités se répartissent en niveaux d'abstraction.

> Un concept d'un niveau est un raffinement du concept homonyme du niveau d'en dessous avec les concepts "voisins" de ce concept.

L'utilisation de **verbalisations** est controversée en psychologie cognitive, et nous sommes d'accord avec les objections qui leur sont faites. Dans le cas du jeu de Go, domaine complexe, les verbalisations cachent l'abstraction et ne servent pas pour agencer les concepts du modèle. Cependant, elles servent de point de départ indispensable et permettent de **conscientiser** des connaissances non conscientes.

Bibliographie

[Bonnet 1988] - C. Bonnet - Les temps de traitement dans la perception visuelles des formes - Psychologie Cognitive, modèles et méthodes - 1988

[Caverni 1988] - J.P. Caverni - La verbalisation comme source d'observables pour l'étude du fonctionnement cognitif - Psychologie Cognitive, modèles et méthodes - 1988

[Ericsson 1975] - K.A. Ericsson - Instruction to verbalize as a means to study problem solving processes with the eight puzzle: a preliminary study - Stockholm University, Department of Psychology - report n° 458 - 1975

[Ericsson Simon 1980]- K.A. Ericsson H.A. Simon - Protocols analysis, verbal reports as data - MIT Press - Cambridge, Massachussetts, London, England

[Gagne & Smith 1962] - R.H. Gagne and E.C. Smith - A study of the effects of verbalisation on problem solving - Journal of experimental psychology - 63, 12-18 - 1962

[Karpf 1973] - D.A. Karpf - Thinking aloud in humain discrimination learning - Dissertation Abstracts International - 33, 6111 B (University Microfilms International n° 73-13625) - 1973

[Kipling] - R. Kipling - Histoires comme ça.

[Levy-Schoen 1988] - Levy-Schoen - Les mouvements des yeux comme indicateurs des processus cognitifs - Psychologie Cognitive, modèles et méthodes - 1988

[Nisbett Wilson 1977] - R.E. Nisbett T.D. Wilson - Telling more than we can know: Verbal reports on mental processes - Psychological Review 84, 231-259 - 1977

[Smith & Miller 1978] - E.R. Smith, F.S. Miller - Limits on perception of cognitive process : a reply to Nisbett and Wilson - Psychological Review 85, 355-362 - 1978

[Vermersch 1991a] - Vermersch P. - Questionner l'action: l'entretien d'explicitation - Psychologie française, numéro spécial "Anatomie de l'entretien", 35-3, 227-235 - 1991

[Vermersch 1991b]- Vermersch P. - Les connaissances non conscientes de l'homme au travail - Le journal des psychologues 84 - 1991

[White 1980] - P.A. White - Limitations on verbal reports of internal events : a refutation of Nisbett and Wilson and of Bem - Psychological Review 87, 105-112 - 1980

[Wittgenstein 1951] - L. Wittgenstein - de la certitude - Gallimard - Septembre 1987

L'IMPLÉMENTATION DU MODÈLE SUR MACHINE

Ce paragraphe présente brièvement les avantages de l'implémentation du modèle sur machine.

Validation du modèle

C'est plus qu'un avantage, c'est un impératif. Nous pensons que pour valider un modèle cognitif, il faut implémenter le modèle sous forme de programme sur machine et ne pas se contenter d'une théorie "papier". A ce titre, le modèle qui sous-tend le programme est computationnel. Dans la modélisation d'un domaine complexe une idée "bonne théoriquement" (dans la tête ou sur le papier) peut être très mauvaise pratiquement.

Nous avons procédé par itérations. Au début nous sommes parti de verbalisations et nous avons conçu un premier modèle. Nous l'avons implémenté. Nous avons comparé les résultats du programme aux résultats de joueurs humains. Au début ils étaient très mauvais. Tant que les résultats n'étaient pas identiques à ceux de joueurs humains, nous avons recommencé. Peu à peu nous avons abandonné les verbalisations pour les raisons citées précédemment. Une itération devenait la succession des phases suivantes : éventuellement conception[1] , programmation, tests. Les tests pouvaient être le résultat de parties contre d'autres joueurs ou l'analyse du résultat sur des sous-problèmes du jeu de Go.

Extraction de connaissances non conscientes de l'homme par différence

Après quelques cycles conception-implémentation-tests du modèle sur machine, nous étions en train d'implémenter des idées que nous n'avions pas au départ. Ces idées nous sont venues en voyant pratiquement les résultats produits par le modèle. Dans la modélisation d'un domaine complexe, nous pensons que la machine est nécessaire non seulement pour valider ou invalider des idées mais surtout pour donner d'autres idées au concepteur du modèle.

L'avantage des machines pour les sciences cognitives est qu'elles n'ont pas l'équivalent de nos connaissances intuitives. Par différence de résultat entre la machine et l'homme on peut tenter de caractériser l'activité non consciente de l'homme. Essayer de faire tourner des modèles (nécessairement conscientisés : si le modèle existe sur une machine ou du papier c'est qu'il a été rendu conscient) sur des machines peut nous faire percevoir la part des connaissances non conscientes et conscientes de l'homme qui joue au Go.

Conclusion

Le résultat d'une idée a priori dans un domaine complexe est imprévisible. Pour valider un modèle cognitif, il est impératif de passer par un modèle computationnel et de l'**implémenter sur machine**. D'autre part, la machine qui ne possède l'équivalent d'aucune connaissance du domaine d'un joueur humain au départ, permet d'**expliciter l'équivalent de connaissances non conscientes** du joueur de Go, par différence.

---

[1]Si les résultats étaient trop différents de ceux d'un joueur humain, nous devions revoir la conception générale de notre modèle.

L'UTILISATION DE DOMAINES VOISINS

La richesse du jeu de Go le place au carrefour de plusieurs domaines. En effectuant des analogies entre ces domaines, nous avons porté un regard nouveau sur notre travail. Ces analogies ont rendu ces domaines voisins de notre travail et nous les appelons ainsi dans la suite. Il est clair que les domaines voisins ont été des sources indispensables à notre travail. C'est souvent la combinaison de plusieurs domaines voisins qui fait découvrir un concept intéressant et utile à notre travail. Le but de ce chapitre est de présenter ces domaines.

Dans ce chapitre nous présentons les domaines voisins suivants.

La **Théorie des jeux** car le Go est un jeu !

l'**Intelligence Artificielle Distribuée** à cause de la nature distribuée du jeu de Go.

la **Vision** à cause de la nature visuelle du jeu de Go. Plus précisément :
- la **Géométrie Fractale** pour 2 raisons:
  1) à cause des 3 échelles principales: intersection, locale, globale
  2) à cause de l'idée de fraction,
- la **Morphologie mathématique** à cause de la nature du goban,
- **David Marr** à cause de la hiérarchie de son modèle computationnel de la vision.

La **Logique floue**
- à cause du flou inhérent à des concepts visuels comme les territoires.
- à cause d'imprécisions nécessaires pour décrire un domaine complexe.

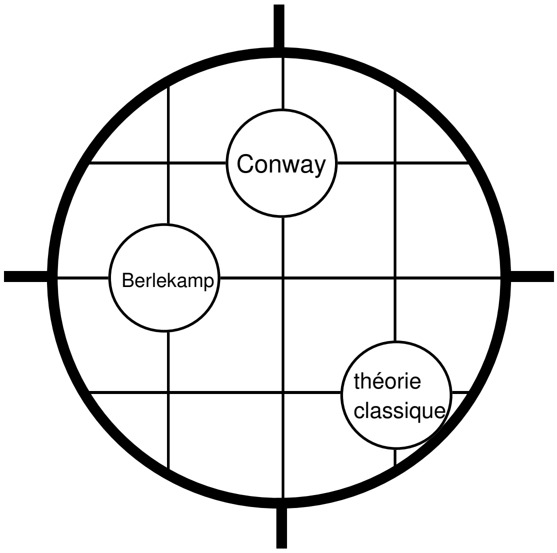
Le **Méta** pour deux raisons:
- A cause de la nécessité de décomposer le jeu de Go en sous-jeux. Chaque jeu manipulant d'autres jeux (au lieu de manipuler des coups) est considéré comme un métajeu.
- A cause de la possibilité d'utiliser un langage de règles dont les prémisses sont des patterns, de la possibilité de générer ces patterns avec des métapatterns de spécialisation.

Le **concept de territoire est typique de la synergie qui existe entre ces domaines**. D'abord la géométrie fractale nous a fait faire une analogie entre des "fjords" de la croissance fractale et les territoires au Go. Ensuite, la morphologie mathématique symbolique a permis de formaliser la notion de territoire grossièrement avec l'opérateur de fermeture. Enfin la numérisation de cette morphologie mathématique a permis de stabiliser la reconnaissance des territoires dans des situations réelles où le "bruit" est présent. . Nous avons choisi d'exposer dans ce chapitre, le concept de territoire présent dans INDIGO, au lieu de le faire dans le chapitre sur la présentation du modèle.

Ces domaines sont présentés soit dans ce chapitre, soit dans le chapitre sur l'évaluation de notre travail, soit dans les deux chapitres. Le plan de ce chapitre est le suivant :

La théorie des jeux
L'Intelligence Artificielle Distribuée
La vision
La logique floue

La théorie des jeux



Introduction :

Ce paragraphe décrit l'état de l'art dans ce que nous appelons la théorie des jeux. Nous procédons en plusieurs étapes. D'abord, nous réglons le cas de la théorie des jeux "économiste". Ensuite, nous décrivons un jeu au sens *classique* tel qu'il est perçu depuis que la programmation du jeu d'Échecs est populaire en Intelligence Artificielle, c'est-à-dire de façon monolithique avec un but unique et une recherche arborescente pour l'atteindre. Puis, nous décrivons un jeu au sens de Conway tel qu'il est maintenant utilisé dans la programmation du jeu de Go. Nous citons également les travaux de Berlekamp. Enfin, nous récapitulons notre situation au sein de cet état de l'art en théorie des jeux.

La théorie des jeux "économiste"

Nous n'entendons pas "théorie des jeux" au sens utilisé par les économistes [Von Neumann & Morgensteen 1944]. De cette "théorie des jeux", deux informations sont importantes pour situer notre discours. La première décrit des comportements de joueurs "psychologiques". Ceux-ci visent à jouer sur la psychologie de l'adversaire. Cette théorie permet de *jouer l'adversaire* plutôt que jouer tout court. Cela n'entre pas dans notre discours. Par contre, cette théorie des jeux fut la première à formaliser clairement le principe du minimax que nous citons plus loin.

La théorie classique

Préliminaires :

La théorie classique comprend toute la littérature sur les jeux depuis une cinquantaine d'années. Les idées dont la communauté scientifique bénéficie actuellement ont mûri lentement et en parallèle. Il est donc difficile d'attribuer la paternité d'un concept donné à un ou des auteurs bien précis. Souvent une idée nouvelle apparaît pour la première fois dans un article sous forme confuse pour un lecteur non averti. La même idée apparaît plus claire dans un état de l'art postérieur. Cette remarque nous a permis de parcourir de nombreux articles originels et de nous intéresser aux articles de synthèse sur le domaine pour dresser un panorama dans un style prosaïque.

Il est également frappant de constater comment un algorithme est dit "optimal" selon les auteurs au moment de la parution de l'article, puis comment un article postérieur présente un nouvel algorithme meilleur que l'algorithme "optimal"... Cet aspect n'est sans doute pas spécifique du domaine de la recherche arborescente mais il nous a paru suffisamment caricatural pour que nous nous permettions de le préciser.

Généralités :

L'idée générale est de considérer un jeu à deux joueurs à information complète et de le représenter sous une forme arborescente où chaque noeud de l'arbre correspond à une position du jeu. Chaque position possède une valeur que l'on peut déterminer statiquement avec une fonction d'évaluation. Aux Échecs, une "bonne" fonction d'évaluation consiste à attribuer une valeur algébrique à chaque pièce (10 pour la Reine, 5 pour la Tour, 3 pour le Fou ou le Cavalier, 1 pour le Pion) et de faire la somme des valeurs pour chaque pièce pour trouver la valeur de la position. La recherche du "meilleur" coup consiste alors à effectuer une recherche de la meilleure valeur dans cet arbre.

Le Minimax :

L'idée de départ : la recherche arborescente par minimax, est attribuée à la théorie "économiste" des jeux [Von Neumann & Morgensteen 1944]. Le principe de cette méthode est de balayer tout l'arbre jusqu'aux feuilles et de "remonter" les valeurs terminales en prenant alternativement le maximum ou le minimum des valeurs des noeuds fils du noeud considéré.

L'Alpha-Bêta :

Ce n'est que plus tard qu'une nouvelle recherche arborescente apparaît progressivement. Samuel en parle dans son article sur les Checkers [Samuel 1959], mais cette idée lui paraît moins importante que les autres idées d'apprentissage présentées dans cet article. Cette nouvelle méthode reçut le nom de recherche arborescente par "alpha-bêta" lorsque McCarthy nomma deux variables de son programme alpha et bêta pour désigner des valeurs "optimiste" et "pessimiste" du jeu. L'avantage de l' alpha-bêta sur le minimax est de trouver une solution optimale sans balayer tout l'arbre. L'algorithme coupe les branches de l'arbre dès qu'il sait qu'un noeud aboutira à une valeur minimax moins bonne que ce qu'il a déjà parcouru avant. L'inconvénient de cette méthode est de ne pas trouver toutes les solutions. Nous verrons que cet inconvénient est gênant dans le cas du jeu de Go. Un point commun de minimax et de alpha-bêta est de rendre la recherche arborescente indépendante du jeu considéré.

L'Alpha-Bêta pruning :

Au début des années soixante, les programmes d'Échecs utilisent le "alpha-bêta pruning", technique qui vise à ordonner l'exploration des coups générés dans une position donnée pour obtenir le plus de coupes alpha-bêta dans l'arbre et réduire la taille de la recherche. Cette technique qui classe les coups dans un certain ordre est dépendante du jeu considéré. Une description claire figure dans [Samuel 1967]. Une formalisation de la méthode figure dans [Nillson 1971] et [Knuth 1975]. Pendant une dizaine d'années cette technique s'est stabilisée. Des résultats concrets sont apparus à l'image de la progression des programmes d'Échecs.

L'algorithme B* :

Parallèlement, l'algorithme B* [Berliner 1975] est apparu, bénéficiant de la combinaison des recherches sur l'alpha-bêta et de l'algorithme A* [Nillson 1971]. Ce dernier algorithme recherche une solution d'un problème représenté par un graphe d'états par application d'heuristiques. L'algorithme B* reprend l'idée d'affecter une valeur optimiste et une valeur pessimiste à chaque noeud de l'arbre. L'algorithme B* dispose donc de deux fonctions d'évaluation : une optimiste <u>et</u> une pessimiste.

L'optimisation de l'alpha-bêta :

Dans le début des années 80, la nouvelle question fut de savoir comment mener la recherche alpha-bêta, en largeur ou en profondeur d'abord ? En effet, jusqu'à ce moment là, les recherches étaient menées en profondeur d'abord. L'idée de rechercher en largeur d'abord apparut avec l'algorithme SSS* (Search Space State) [Stockman 1979]. Cette idée était séduisante à l'époque car son auteur parlait de "parallélisation" de l'alpha-bêta. La réponse à la question est sans doute ni l'un ni l'autre comme le montre la méthode Depth First Iterative Deepening [Korf 1985] qui optimise la recherche en termes de temps, d'espace mémoire et de coût du chemin trouvé. Une comparaison des algorithmes des années 80 se trouve dans [Campbell & Marsland 1985]. Elle montre que l'optimisation de l'alpha-bêta dépend de critères non étudiés jusque là : l'importance de l'ordre des noeuds fils d'un noeud, la forme et l'équilibre de l'arbre de recherche. Les arbres correspondant aux jeux étudiés ne sont généralement pas équilibrés et les coups sur une position sont généralement plus ou moins prioritaires ou urgents.

Les algorithmes récents :

Les recherches menées depuis la fin des années 80 visent à trouver comment poursuivre la recherche arborescente de manière indépendante du jeu considéré. La technique des nombres conspirant [MacAllester 1988] développe les noeuds terminaux de l'arbre en fonction du nombre de noeuds nécessaires pour faire changer la valeur minimaxée de la racine de l'arbre. Cette idée a été implémentée [Schaeffer 1990] mais ne donne de bons résultats que sur les positions tactiques des Échecs. L'idée a été reprise et travaillée avec l'algorithme B* pour donner la méthode PN-search (Proof Number Search) [Allis 1994] qui a permis de résoudre de nombreux jeux (Awari, Go-moku).

En parallèle de ces améliorations, la recherche de quiescence s'était développée depuis longtemps [Shannon 1950]. Shannon montre comment la procédure minimax peut être de profondeur variable, s'arrêtant seulement à des positions calmes, et comment on peut rejeter certains coups. La technique s'est généralisée [Beal 1989]. Elle compare une position atteinte par un coup à la position atteinte par le coup "passe". Les résultats de l'alpha-bêta et de la recherche de quiescence sont équivalents [Beal 1989]. Dans les meilleurs programmes actuels, les deux techniques sont utilisées de façon complémentaire : alpha-bêta jusqu'à un profondeur donnée puis recherche de quiescence.

Conclusion :

Il est important de comprendre que toute la recherche sur l'alpha-bêta a été menée a priori de façon indépendante du jeu considéré. Évidemment, le jeu d'Échecs a été le terrain d'application favori de ces techniques et les résultats actuels concrétisent cet effort : les meilleurs programmes d'Échecs ont le niveau correspondant à celui de Grand Maître International [Anantharaman & al 1989] et Gary Kasparov a récemment perdu une partie semi-rapide contre Chess Genius 2. Ces résultats sont très satisfaisants pour l'Intelligence Artificielle qui offre avec l'alpha-bêta et ses dérivés des outils a priori généraux indépendants du jeu considéré. Que rêver de mieux ?

La théorie de Conway

Introduction :

Quand nous nous sommes intéressés à la théorie des jeux de Conway [Conway 1982] et au jeu de Go, nous avons découvert que la notion de jeu telle que nous la percevions via la théorie classique présentée ci-dessus était incomplète. Avec la théorie des jeux de Conway et le jeu de Go nous avons compris qu'un jeu possède un "intérieur" et un "extérieur". Aux Echecs et aux jeux que l'on programme essentiellement par de la recherche arborescente, l'"extérieur" du jeu correspond au but du jeu : gagner. Pour ces jeux, l'"intérieur" correspond à la manière d'atteindre le but : la recherche arborescente. Dans ces "monojeux", le joueur ne se pose pas la question de savoir s'il faut donner un autre aspect extérieur au jeu. Il a raison d'ailleurs, puisqu'il faut gagner et seulement gagner. **La théorie classique a beaucoup étudié l'intérieur d'un jeu mais a négligé son extérieur**.

La théorie de Conway propose un aspect "extérieur" au jeu constitué de propriétés et d'opérations. Cette notion d'"extérieur" convient bien au jeu de Go. En effet, la structure distribuée du jeu de Go oblige à modéliser le jeu de Go en le décomposant en sous-jeux. Pour chacun de ces sous-jeux, le joueur "joue" à ce sous-jeu au sens classique : il essaie de gagner. S'il gagne, ce jeu n'est pas pour autant terminé. Le joueur doit "gérer" le jeu pour conserver le gain de ce jeu. Si la gestion est trop coûteuse, le joueur peut abandonner ce jeu et faire autre chose pendant ce temps. Le choix change l'aspect "extérieur" de ce sous-jeu. Pour simplifier : tout ce qui est relatif au fait de jouer correspond à l'intérieur du jeu et ce qui est relatif à la gestion du jeu correspond à l'extérieur du jeu. **La théorie de Conway offre des outils mathématiques pour étudier l'extérieur d'un jeu**. Nous présentons de façon simplifiée les notions de base de la théorie de Conway qui nous ont servi à construire notre modèle sur le jeu de Go.

Propriétés d'un jeu :

Avec des concepts mathématiques complexes et un formalisme adapté, la théorie de Conway permet de symboliser les propriétés d'un jeu. Ces propriétés sont beaucoup plus fines que le simple résultat d'un jeu (gagné, perdu) dont nous avons l'habitude et que nous héritons du jeu d'Échecs.

Un jeu oppose deux joueurs Gauche et Droite. Les joueurs jouent à tour de rôle. Le jeu se termine quand on ne peut plus jouer. Pour exprimer que le joueur Gauche a m coups possibles et Droite a n coups possibles, on note:
$J = \{ Jg_1, \dots, Jg_m / Jd_1, \dots, Jd_n \}$

Etat statique

Un jeu a un état statique défini par les règles du jeu : Gagné, Perdu, Autre.

Partie gauche, partie droite

On suppose d'abord que Gauche commence et que les deux joueurs jouent de façon optimale, on obtient la *partie gauche Pg* du jeu avec la valeur minimaxée. On suppose que le jeu est calculable c'est-à-dire que cette valeur est Gagné ou Perdu, mais jamais Autre. On suppose ensuite que Droite commence, on refait la même chose pour obtenir la *partie droite Pd* du jeu.

Etat dynamique

En fonction des valeurs prises par la partie gauche et la partie droite du jeu on définit l'*état dynamique E* d'un jeu.

"<"

Si Pg = Pd = Perdu alors Ed = "<"
Cela signifie que le jeu est perdu pour Gauche même si Gauche commence. C'est le cas aux Échecs si Gauche est Mat.

">"

Si Pg = Pd = Gagné alors Ed = ">"
Cela signifie que le jeu est gagné pour Gauche même si Droite commence. C'est le cas aux Échecs si Droite est Mat.

""*

Si Pg = Gagné et Pd = Perdu alors Ed = "*"
Le jeu est gagné pour le premier qui joue.

"0"

Si Pg = Perdu et Pd = Gagné alors Ed = "0"
Le jeu est perdu pour le premier qui joue. C'est le cas du seki au Go.

"?"

Les calculs de parties gauche et droite du jeu peuvent donner aussi la valeur Autre. Il faut un nouveau symbole pour l'état dynamique.
Si Pg = Autre ou Pd = Autre alors Ed = "?"
En pratique, ce cas peut se produire très souvent : si un calcul dure trop longtemps et qu'il est arrêté.

Arbre et jeu

Un jeu est représentable sous forme d'un arbre dont les feuilles désignent un gain, une perte ou une égalité. On peut jouer un coup, ce qui fait déplacer l'état de la partie d'un noeud de l'arbre vers un autre, ou bien passer, ce qui ne change pas l'état de la partie sauf le trait.

Température

Elle mesure la différence entre la valeur de gain et la valeur de perte. Cette notion est surtout importante pour engendrer les coups du jeu global à partir des jeux du niveau immédiatement inférieur : les groupes, les territoires, les espaces vides.

Valeur principale

Elle mesure la valeur vers laquelle se dirige le jeu si les deux joueurs jouent bien. Cette notion est surtout importante pour générer les coups du jeu global à partir des jeux du niveau immédiatement inférieur : les groupes, les territoires, les espaces vides.

Opérations sur des jeux :

Conway s'est intéressé à une multitude de jeux et il en a inventé beaucoup d'autres pour illustrer sa théorie. Il s'est intéressé au jeu de Nim où le but est de ramasser des allumettes. Le joueur qui ne peut plus ramasser d'allumette a perdu. Il a notamment eu l'idée d' additionner des jeux pour en créer un nouveau. Par exemple, étant donné deux jeux, on peut jouer au jeu somme de ces deux jeux en jouant un coup dans l'un ou l'autre des deux jeux à chaque coup. Plus généralement, il a eu l'idée d'effectuer des opérations sur les jeux.

L'intérêt et le principe général de la théorie de Conway

La théorie de Conway décrit un jeu par des opérations et des propriétés. Ce qui est important dans cette théorie est aussi de déduire les propriétés d'un jeu, composé de sous-jeux, à partir des propriétés de chacun des sous-jeux sans utiliser d'outil de recherche arborescente comme l'alpha-bêta. En d'autres termes, un jeu de Conway est décrit. Sa description est utilisée pour connaître la description d'autres jeux construits avec ce jeu.

Au Go, la complexité liée au nombre de coups possibles à chaque position rend impossible une recherche arborescente telle quelle. Pour cette raison, il est nécessaire de découper le jeu global en sous-jeux, de découper un sous-jeu en d'autres sous-jeux, de relier les jeux entre eux, etc. C'est-à-dire d'identifier des jeux, de les décrire et d'effectuer des opérations sur des jeux. Le lecteur comprendra donc que la théorie des jeux de Conway et le jeu de Go se tiennent par la main : le jeu de Go bénéficiera des résultats de cette théorie si l'on sait le modéliser et le mouler dans le cadre de cette théorie. Le jeu de Go, avec ses spécificités, peut enrichir à son tour cette théorie.

La théorie de Berlekamp

Berlekamp a adapté la théorie de Conway à la fin de partie du jeu de Go. Les résultats sont excellents et même un "cauchemar pour les joueurs professionnels" : la théorie mathématique de Berlekamp [Berlekamp 1991] montre que les joueurs professionnels se trompent de 1 ou 2 points sur des positions de fin de parties [Berlekamp & Wolfe 1994].

Mais la théorie de Berlekamp a des limitations. Les positions sont des positions de tout petit yose[1] (toute fin de partie). L'information en entrée de cette théorie doit être prétraitée. La position doit être découpée en *sous-jeux indépendants* sans quoi la théorie ne s'applique pas. La théorie ne tient pas compte de la vie et la mort des groupes, qui est la difficulté majeure de la modélisation du jeu de Go. Les séquences de la théorie ne sont calculées qu'en fonction du yose et si, par malchance, une séquence modifie la vie et la mort d'un groupe, le résultat est complètement faux. Ces limitations empêchent l'utilisation de cette théorie, telle quelle, pour jouer une partie de Go complète. Berlekamp dit lui-même au début de son livre que sa théorie ne peut faire progresser un joueur de Go, qui lirait son livre en entier, que dans la limite de un point en fin de partie !

Berlekamp a montré que des adaptations de la théorie de Conway sont possibles. Le problème est de trouver une adaptation qui soit utile pour traverser une partie de Go complète.

La notion de jeu dans INDIGO

Introduction :

Nous avons cherché à définir un concept de jeu qui s'applique à toutes les phases de la partie. De nombreux chercheurs étudient cette voie prometteuse [Müeller 1993] pour l'appliquer au jeu de Go dans sa globalité. Les difficultés pour adapter la théorie de Conway sont les suivantes. Les jeux de la décomposition sont de types hétérogènes. Les jeux peuvent être dépendants. Les jeux sont reliés par des relations de types hétérogènes.

Nous présentons les notions que nous avons ajoutés à la théorie de Conway spécifiquement pour le jeu de Go. La liste des jeux que nous avons identifiés est présentée dans la partie 3 du document.

Propriété d'un jeu :

Hiérarchie père-fils

Un jeu est le père d'autres jeux si son résultat dépend d'une combinaison logique de ces jeux. Les fils sont des sous-jeux et le père est un sur-jeu. A la racine de la hiérarchie se trouve le jeu global et selon le type de hiérarchie que l'on voudra donner au jeu de Go aux feuilles se trouvent les jeux de l'intersection.

[1] Le yose signifie la fin de partie. Le petit yose signifie la toute fin de partie.

Propriétés supplémentaires d'un jeu :

Source

Un sous-jeu du jeu de Go est construit à partir d'un objet reconnu sur le goban. Il correspond à une action que les joueurs effectuent sur cet objet. Par exemple, pour une chaîne de pierres sur le goban, le jeu de la chaîne est le jeu qui consiste à capturer la chaîne. Nous appelons *source* du jeu l'ensemble des intersections sur lesquelles l'objet repose.

Lieu statique, Lieu dynamique

Une des particularités du jeu de Go est qu'il se déroule en un lieu : le goban. Par voie de conséquence les sous-jeux de la hiérarchie se déroulent aussi sur un sous-ensemble d'intersections. Nous appelons le *lieu statique* du jeu, l'ensemble minimal d'intersections que la règle de ce jeu utilise pour générer les coups de ce jeu dans une position donnée. Nous appelons *lieu dynamique* du jeu la réunion de tous les lieux statiques du jeu pour toutes les positions rencontrées au cours d'un calcul, si calcul il y a. Par abus de langage, nous écrirons simplement *lieu* au lieu de lieu dynamique. A titre d'indication, la source d'un jeu est incluse dans le lieu statique d'un jeu qui est inclus dans le lieu.

Exemple:

Par exemple, sur la figure *Jeu-source-lieu,* la source du jeu de la chaîne noire est exactement la chaîne noire, le lieu statique est constitué des intersections noires, blanches ou libertés de la chaîne noire ou libertés d'une chaine blanche voisine de la chaîne noire (si on suppose que le générateur de coups du jeu de la chaîne engendre les coups qui sont des libertés de chaîne noire ou de chaînes voisines de la chaîne noire).

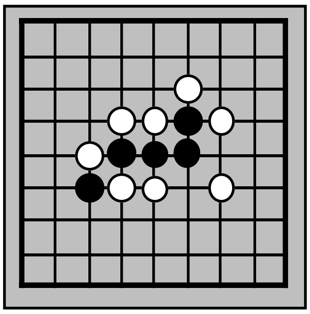


figure Jeu-source-lieu

Le lieu dynamique est la réunion de tous les lieux statiques rencontrés dans la recherche arborescente. Par exemple, il contient le lieu statique de la chaîne noire sur position de la figure *Jeu-lieu-dynamique.*

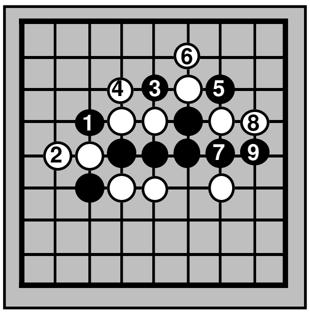


figure Jeu-lieu-dynamique

Indépendances entre jeux voisins

Deux jeux sont plus ou moins indépendants selon que leurs sources ou leurs lieux ont une intersection vide ou non.

Dépendance du résultat

Un jeu A est dépendant d'un jeu B si le résultat de B est utilisé pour connaître le résultat de A.

Règles de recomposition

Si deux jeux sont indépendants, on peut utiliser des règles de recomposition de résultats de jeux pour connaître le résultat du surjeu obtenu par disjonction ou conjonction.

Exemple:

La base de vie d'un groupe dépend du nombre et de l'état des yeux. Le jeu de la base de vie dépend du jeu de l'œil. On peut utiliser des règles de recomposition avec des approximations que nous discutons au paragraphe Métajeu de la partie 4 de ce document. La figure *Jeu-règles-de-recomposition* donne deux exemples de règles de recomposition.

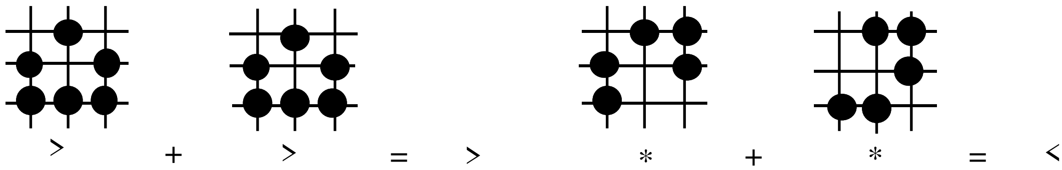


figure Jeu-règles-de-recomposition

Taille

Elle sert pour engendrer les coups du jeu global à partir des jeux du niveau immédiatement inférieur : les groupes, les territoires, les espaces vides.

Coup gote, coup sente

Un coup gote est un coup sur un jeu * qui le fait passer dans l'état > ou <.
Un coup sente est un coup sur un jeu > tel que si l'autre ne répond pas, le jeu n'est plus >

Règles générales d'utilisation des jeux

Ne pas essayer de gagner un jeu <.
Ne pas gagner un jeu déjà >.
Jouer les coups gote (les jeux *).
Jouer des coups sente pour en tirer avantage dans d'autre jeux voisins.
Jouer sur les jeux les plus importants.

Conclusion

Nous avons donné un aperçu de la théorie classique des jeux et de la théorie de Conway. Ce paragraphe nous positionne par rapport à ces théories.

Situation par rapport à la théorie classique, l'intérieur d'un jeu :

Contrairement à ce que laisse croire son apparence de généralité, la théorie classique s'est développée avec, par et pour la programmation du jeu d'Échecs, l'un et l'autre s'aidant mutuellement. Le jeu d'Échecs ayant juste la complexité qu'il faut pour être étudié par cette théorie. La théorie classique a développé des techniques adaptées à l'intérieur d'un jeu[1] mais elle laisse sous silence l'extérieur d'un jeu[2]. Même si notre travail sur le jeu de Go s'attache plus à l'extérieur d'un jeu qu'à son intérieur, nous ne pouvons ignorer l'intérieur d'un jeu et les techniques qui s'y rattachent. Pour manipuler l'intérieur d'un jeu nous avons utilisé des techniques simples qui nous suffisaient et qui dépendaient du type de jeu (son extérieur...): *pour étudier la partie droite et la partie gauche d'un jeu élémentaire nous avons utilisé du* ***alpha-bêta en profondeur d'abord***. Cela nous suffisait, d'une part, mais était nécessaire car, d'autre part, il faut aller au bout de la séquence pour calculer si un shicho marche ou non. *Pour étudier une situation globale nous avons utilisé la recherche de quiescence* (même si pour des raisons pratiques celle-ci est souvent réduite à une simple analyse statique sans arborescence !)

Situation par rapport à la théorie de Conway, l'extérieur d'un jeu :

Nous avons vu que nous utilisons une *hiérarchie* de jeux. Cette hiérarchie est expérimentale et en perpétuelle évolution. Nous avons des relations de *filiations* entre les jeux. Un jeu se déroule sur un *lieu*. Nous avons défini la *dépendance* entre deux jeux. Nous utilisons des *règles de recomposition* de résultats de jeu. La relation de filiation entre le jeu global (la racine de la hiérarchie) est spéciale car elle fait intervenir la *taille* d'un jeu.

Bibliographie

[Allis 1994] - Louis Victor Allis - Searching for Solutions in Games and Artificial Intelligence - Ph.Thesis - Vrije Universitat Amsterdam - Maastricht - Septembre 1994

[Anantharaman & al 1989] - Anantharaman T.S., Campbell M.S., Hsu F.H. - Singular extensions : Adding selectivity to brute-force searching - Artificial Intelligence, vol. 43, n°1, pp. 99-109 - 1989

[Beal 1989] - Don F. Beal - A Generalised Quiescence Search Algorithm - Artificial Intelligence, vol. 43, n°1, pp. 85-98 - 1989

[Berlekamp 1991] - Elwin Berlekamp - Introductory overview of mathematical go endgames - Proceedings of symposia in applied mathematics - 43 - 1991

[Berlekamp & Wolfe 1994] - E.R. Berlekamp, D. Wolfe - Mathematical Go Endgames - Nightmares for the Professional Go Player - Ishi Press International - San Jose, London, Tokyo - 1994

[Berliner 1979] - Berliner H.J. - The B* Tree search algorithm : the best-first proof procedure - Artificial Intelligence, vol. 12, pp. 23-40 - 1979

[Campbell & Marsland 1983] - Campbell M.S., Marsland T.A. - A comparison of minimax tree search algorithms - Artificial Intelligence, vol. 20, n°4, pp. 347-367 - 1983

[Conway 1982] - J. Conway, E.R. Berlekamp, R. Guy - Winnings ways - tome 1 & 2 - Academic press - 1982

[High 1990] - R. High - Mathematical Go - Computer Go 15, fall 90

[1]Recherche arborescente.

[2]Comment utiliser les propriétés d'un jeu dans une position donnée au sein d'un multi-jeu plus vaste qui comprend ce jeu.

[Knuth 1975] - Knuth D.E., Moore R.W. - An analysis of alpha-beta pruning - Artificial Intelligence, vol. 6, n° 4, pp. 293-326 - 1975

[Korf 1985] - Korf R.E. - Depth-First Iterative-Deepening: an optimal admissible tree search - Artificial Intelligence, vol. 27, pp. 97-109 - 1985

[McAllester 1988] - McAllester D.A. - Conspiracy Numbers for Min-Max search - Artificial Intelligence, vol. 35, pp. 287-310 - 1988

[Müeller 1993] - Martin Müeller - Game Theories and Computer Go - Cannes Workshop Computer Go - 1993

[Nillson 1971] - Nillson N.J. - Problem solving methods in Artificial Intelligence - McGraw-Hill, New York - 1971

[Samuel 1959] - Samuel A.L. - Some studies in machine learning using the game of checkers - IBM Journal of research and development - Vol. 3, n° 3 - 1959

[Samuel 1967]- Samuel A.L. - Some studies in machine learning using the game of checkers II - IBM Journal of research and development - Vol. 11, n° 6 - 1967

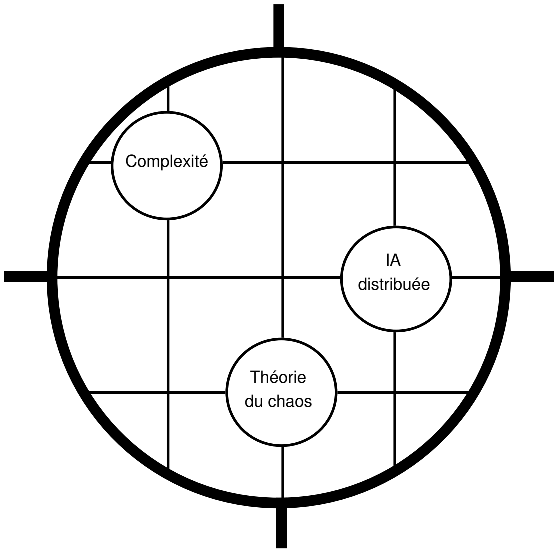
[Schaeffer 1990] - Schaeffer J. - Conspiracy Numbers - Artificial Intelligence, vol. 43, n° 1, pp. 67-84 - 1990

[Shannon 1950] - C. E. Shannon - Programming a computer to play Chess - Philos. Mag. - n° 41, pp 256-275 - 1950

[Stockman 1979] - Stockman G. - A minimax algorithm better than alpha-beta ? - Artificial Intelligence, vol. 12, pp. 179-196 - 1979

[Von Neumann & Morgensteen 1944] - Theory of games and economic behavior - Princeton University Press - Princeton - 1944

L'Intelligence Artificielle Distribuée



Introduction

Nous avons été influencé par des domaines tels que la physique des systèmes complexes, la théorie du chaos et l'Intelligence Artificielle Distribuée. Le but de ce paragraphe est de montrer notre travail suivant la sensibilité de ces domaines. Nous définissons ce que nous appelons un système "naturel". Puis nous donnons une liste d'exemples "naturels" identifiés au Go.

Caractéristiques d'un système "naturel"

Beaucoup d'éléments

Les phénomènes que nous observons dans la nature sont souvent liés au fait que les systèmes naturels sont composés d'un grand nombre d'éléments. Combien de flocons de neige dans sur une pente de montagne en hiver ? Combien d'arbres dans une forêt ?

Des phénomènes chaotiques

L'étude de ce type de système est difficile et pratiquement imprévisible car un petit changement de ses conditions initiales peut produire des effets très différents. Un bruit peut déclencher une avalanche sur une pente neigeuse de montagne. Une allumette peut déclencher un incendie de forêt.

Des phénomènes en chaîne

Souvent, quand l'ampleur de l'effet est très grand devant l'ampleur de la cause, ceci est dû à une succession de phénomènes en chaîne. Une avalanche se produit parce qu'une plaque de neige presque décrochée se décroche et tombe sur la plaque de neige du dessous. Celle-ci se décroche à son tour et ainsi de suite... Un feu de forêt se produit parce qu'un arbre prend feu et enflamme ses voisins, et ainsi de suite...

Des effets à des échelles de grandeur différentes

Souvent, l'effet constaté ne se constate pas à l'échelle des éléments qui constituent le système mais à une échelle supérieure. L'avalanche de neige produit un effet à l'échelle de toute la montagne. Un feu de forêt détruit toute une forêt.

Le bruit de scintillement

Souvent, les effets se produisent à intervalles de temps irréguliers. Des études ont montré que la fréquence des catastrophes d'amplitude A est inversement proportionnel à A^b, où b est une constante positive qui dépend du type du système. Quand b vaut 1, on dit que le signal d'amplitude A est le bruit de scintillement, sans doute par analogie avec le scintillement des étoiles.

Les systèmes à états critiques auto-organisés

Ces caractéristiques souvent observées dans la nature seraient la signature de ce qu'on appelle les systèmes à états critiques auto-organisés, des systèmes en équilibre car des forces opposées s'annulent, mais en équilibre instable au sens ou une petite perturbation détruit complètement l'équilibre, auto-organisés dans le sens où chaque élément possède un comportement simple qui produit un effet imprévisible à l'échelle du système global.

De nombreuses études et analogies [Bak & Chen 1991] [Bak 1992] ont déjà été faites entre divers domaines, les tas de sable, les neurones du cerveau, les feux de forêt, l'activité en bourse, les avalanches de neige, les tremblement de terre, les conflits sociaux, les guerres, etc... Nous citons quelques phénomènes observés dans une partie de Go.

Des exemples "naturels" rencontrés au jeu de Go

Nous avons répertorié de nombreux cas pour lesquels un effet local produit un effet global par une succession d'application de règles simples. Le tableau ci-dessous est le résumé des exemples que nous avons choisis pour illustrer ce point.

CAS	EFFET LOCAL	EFFET GLOBAL
Règle de connexion	Deux pierres connectées	Création d'une chaîne
Règle de capture	Une pierre enlevée	Suppression d'une chaîne
Shicho	Atari-Sortie	Motif diagonal
Faux encerclement	Atari sur voisin	Sortie de l'encerclement
Instabilité globale	Poser une pierre	Tout pour un joueur

La règle de connexion

La position de la figure *IAD-connexion-cas* est un exemple de pierres disposées sur un goban.

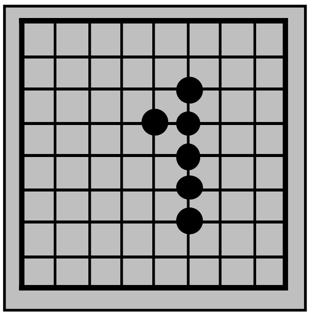


figure IAD-connexion-cas

Effet local

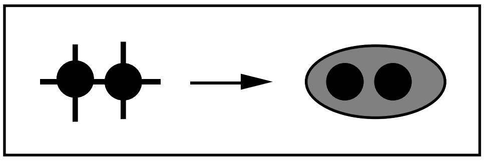


figure IAD-connexion-local

La règle de connexion au Go dit que deux pierres voisines sont connectées et forment une chaîne comme le représente la figure *IAD-connexion-local*.

Effet global

Une succession d'applications de la règle de connexion donne un objet "global", la chaîne, comme le montre la figure *IAD-connexion-global*.

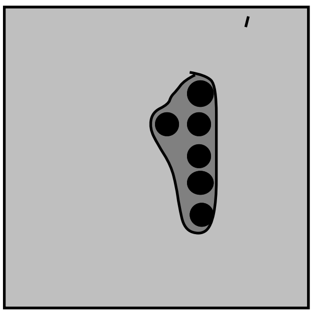


figure IAD-connexion-global

La règle de capture

La règle de capture dit que si l'on supprime la dernière liberté d'une chaîne, on l'enlève du goban. Sur la figure *IAD-capture-cas* la dernière liberté de la chaîne noire a été supprimée.

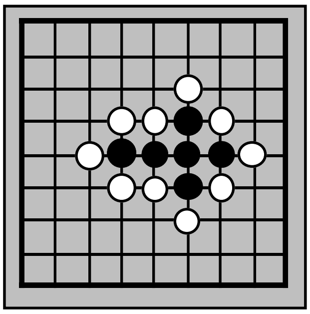


figure IAD-capture-cas

Effet local

La règle de la figure *IAD-capture-local* est utilisée pour enlever une pierre.

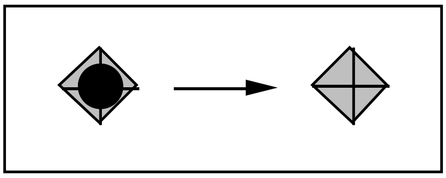


figure IAD-capture-local

Effet global

L'application successive de la règle locale à chaque pierre du goban produit un effet global[1] : le retrait de la chaîne du goban comme le montre la figure *IAD-capture-global* .

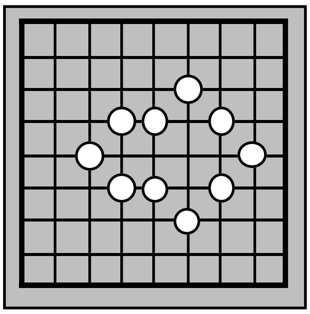


figure IAD-capture-global

[1]Dans cet exemple une pierre est "locale" et une chaîne est "globale".

Le shicho

La figure *IAD-shicho-cas* montre une position au Go qui peut être source d'un motif répété et fréquent dans les parties.

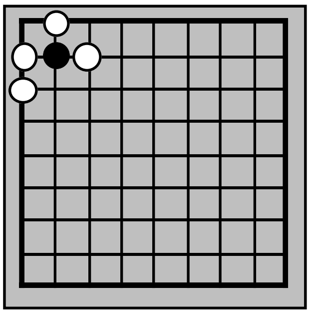


figure IAD-shicho-cas

Si Noir veut sauver sa pierre en atari, il peut la sortir pour la défendre. Mais Blanc pourra faire atari pour essayer de la capturer.

Effet local

L'effet local est traduit par la figure *IAD-shicho-local*.

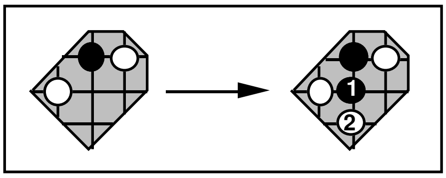


figure IAD-shicho-local

Noir joue 1 pour sortir et avoir 2 libertés mais Blanc lui refait atari.

Effet global

La figure *IAD-shicho-global* montre le motif diagonal qui apparaît par application successive de la règle de la figure *IAD-shicho-local*.

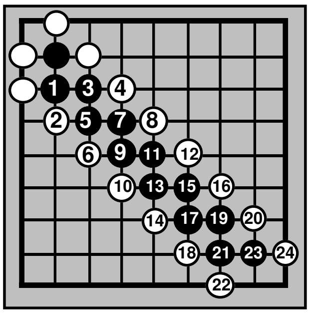


figure IAD-shicho-global

Cela arrive explicitement dans les parties si l'un des deux joueurs insiste pour sauver (respectivement capturer) la chaîne alors que c'est impossible. Implicitement, les shichos se produisent très souvent dans la tête des joueurs.

Si le goban était infini, le motif partirait à l'infini. Cela fait penser aux fourmis généralisées qui repeignent des cases de couleur d'un damier infini [Stewart 1994]. Le théorème de Cohen-Kong montre que les fourmis généralisées n'ont pas de trajectoire bornée. Pratiquement, les simulations montrent qu'elles partent à l'infini selon une diagonale après un nombre d'itérations imprévisible.

Le faux encerclement

La figure *IAD-faux-encerclement-cas* montre une position où la chaîne noire n'a que deux libertés et ne peut en avoir 3 en un coup. Noir veut que cette chaîne obtienne 3 libertés. Blanc veut capturer la chaîne noire. La chaîne noire semble encerclée. Mais cet encerclement est faux comme nous allons le voir.

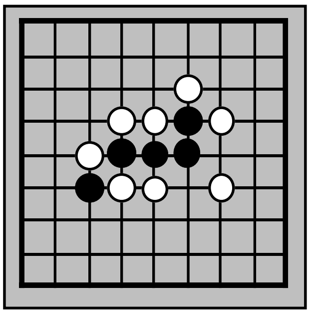


figure IAD-faux-encerclement-cas

<u>Effet local</u>

Les deux joueurs utilisent chacun un effet local tant qu'aucun des deux joueurs n'a atteint son but. Noir utilise la règle de la figure *IAD-faux-encerclement-local-noir* et Blanc utilise la règle de la figure *IAD-faux-encerclement-local-blanc*.

Si atari possible sur chaîne voisine → Faire atari.

figure IAD-faux-encerclement-local-noir

Si chaîne voisine en atari → Sortir la chaîne voisine.

figure IAD-faux-encerclement-local-blanc

Effet global

La figure *IAD-faux-encerclement-global* montre l'effet global produit par une succession d'applications des deux règles précédentes tant que la chaîne ne peut obtenir 3 libertés ou être capturée.

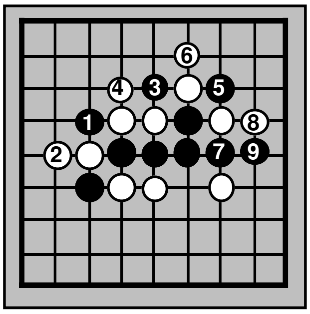


figure IAD-faux-encerclement-global

D'autres successions de couples atari-sortie sont possibles mais ne mènent pas à un état où la chaîne noire a 3 libertés. La succession de couples atari-sortie qui "marche" est circulaire : elle commence sur un voisin de la chaîne et se propage vers le voisin suivant, etc... La chaîne noire qui semblait encerclée sur la figure *IAD-faux-encerclement-cas* ne l'est plus sur la figure *IAD-faux-encerclement-global.*

L'instabilité globale

La position de la figure *IAD-exemple* est instable globalement.

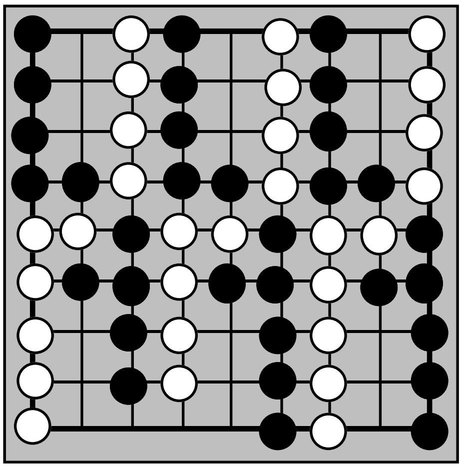


figure IAD-exemple

Si Noir joue comme sur la figure *IAD-noir*,

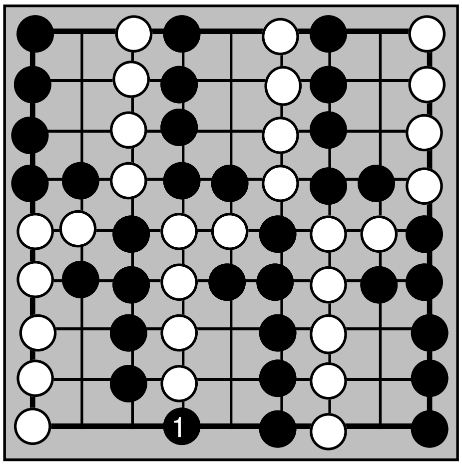


figure IAD-noir

Noir contrôle tout le goban comme le montre la figure *IAD-noir-tout*.

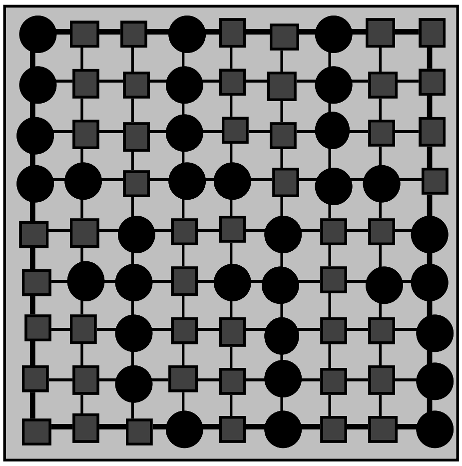


figure IAD-noir-tout

Le groupe blanc situé au côté du coup noir ne possède désormais que deux libertés alors que les groupes voisins en ont au moins trois. Le groupe blanc est donc mort. Un groupe noir fusionne autour de lui. Les groupes voisins blancs de ce groupe noir deviennent plus faibles que lui et meurent à leur tour... et ainsi de suite jusqu'à la formation d'un groupe noir qui recouvre tout le goban. Dans la partie 4 sur l'évaluation de notre modèle cognitif, nous montrons plus en détail comment celui-ci interprète ce type de position.

Et si Blanc joue comme sur la figure *IAD-blanc*,

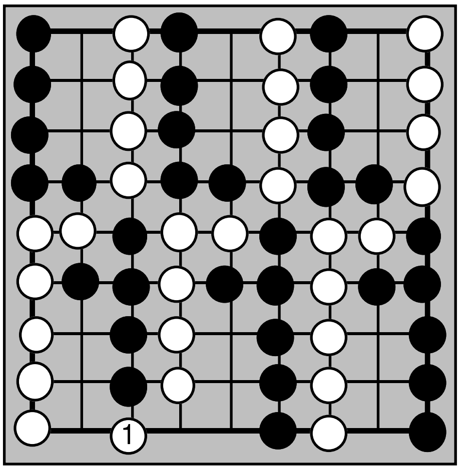


figure IAD-blanc

Blanc contrôle tout le goban comme le montre la figure *IAD-blanc-tout*.

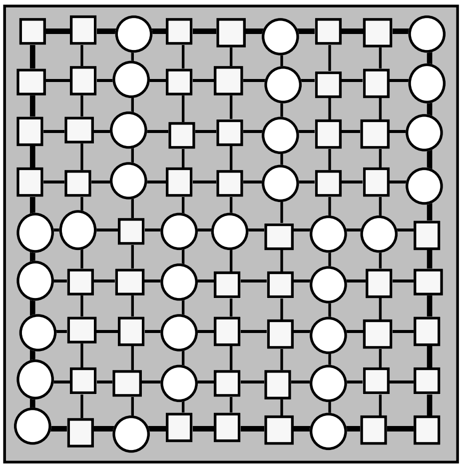


figure IAD-blanc-tout

Effet local

L'effet local est la pose d'une pierre.

Effet global

L'effet global est le contrôle total du goban. Après la présentation du modèle d'INDIGO, nous verrons comment INDIGO interprète ce genre de position instable globalement par une succession de morts de groupe.

Conclusion

Le jeu de Go est distribué et possède des caractéristiques naturelles. Nous montrerons dans la partie sur l'évaluation de notre modèle comment notre modèle interprète une position du type de celles des figures *IAD-noir* et *IAD-blanc*.

Bibliographie

[Bak & Chen 1991] - P. Bak, K. Chen - Les systèmes critiques auto-organisés - Pour la science - Septembre 1991

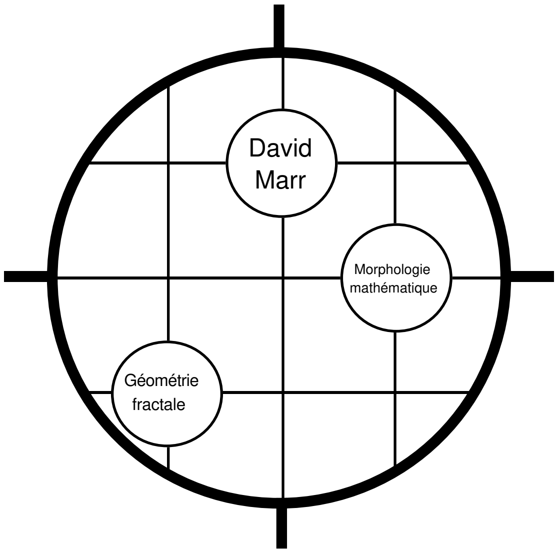
[Bak 1992] - P. Bak - Physica A 191, 41 - 1992

[Stewart 1994] - I. Stewart - La diagonale de la fourmi - Pour la science n° 203, pp. 94-97 - Septembre 1994

La vision

Le jeu de Go est visuel.

Nous montrons comment nous avons été influencé par la Géométrie fractale de Mandelbrot, les travaux de David Marr en théorie computationnelle de la vision et par la Morphologie Mathématique. Nous avons choisi de présenter ces trois domaines sous le même thème Vision :



La Géométrie fractale

Ce paragraphe montre en quoi la géométrie fractale nous a inspiré pour modéliser le jeu de Go.

Pour modéliser les territoires

La modélisation actuelle des territoires n'est pas étrangère à une conférence de B. Mandelbrot à l'Ecole Polytechnique où il parlait des facultés inconscientes de l'oeil. Il disait que tout ce qu'il découvrait, il le faisait avec ses yeux en regardant des images fractales que ses informaticiens avaient engendrées. Ce n'est que plus tard que les démonstrations de ses découvertes étaient faites, par lui-même ou par d'autres. Pendant cette conférence, il montrait des photos ou images d'objets fractals et notamment d'objets créés par croissance fractale, un problème actuel en Physique: des particules avec un mouvement aléatoire viennent s'agréger les unes contre les autres pour former des objets complexes aux particularités diverses. Les outils de la géométrie fractale permettent de mieux comprendre la structure de ces objets physiques. Les images de ces objets fractals les montraient sous des points de vue différents. Notamment, une image qui nous reste, est celle d'un objet fractal vu sous le point de vue que Mandelbrot appelait le point de vue des "fjords". Il appelait "fjord" une zone d'ombre dans laquelle les particules aléatoires avaient très peu de chance d'aller à cause de la forme existante de l'objet déjà agrégé. Sur l'image avec le point de vue des fjords, mon oeil n'a pas vu un objet créé par croissance fractale[1] mais plutôt un goban avec des groupes et des territoires. Les objets croissants étaient les groupes et les fjords étaient les

[1] A ce sujet, il important de remarquer qu'une partie de Go est une expérience de croissance fractale puisqu'une pierre est posée à chaque coup.

territoires. Cette image nous est restée. Dès lors, nous avons vu les territoires comme des zones d'ombre de groupes de pierres et notre cerveau s'est inconsciemment mis en marche de la modélisation de zones d'ombre. Nous le présenterons au paragraphe sur la morphologie mathématique.

Pour modéliser les fractions

Dans son livre The fractal geometry of nature [Mandelbrot 1982], Mandelbrot reprend l'étymologie du mot algèbre: *al jabara*: **le lien** en arabe. Les mathématiques ont longtemps privilégié la continuité des choses et les liens que l'on peut faire entre les choses. Mandelbrot dit que la notion de fractale va à l'inverse de cette idée de liaison entre les choses: on les coupe, les casse, les fractionne sans cesse.

Quand on modélise le Go, on peut choisir de prendre les intersections comme élément de base et de les regrouper en "groupe", c'est-à-dire avoir une approche ascendante des choses : on relie les choses entre elles. On peut aussi choisir de prendre le goban dans sa globalité comme point de départ et de le découper en fragments indépendants : les fractions. Nous ne savons pas si cette deuxième approche est la bonne mais nous savons que la syllabe **frac** contenue dans le mot fractale est présente à notre esprit. Le chapitre qui présente notre modèle donne des exemples de fractions de goban.

Pour modéliser les échelles de grandeurs

Les fractales sont généralement auto-similaires à différentes échelles de grandeur[1]. Il est clair que la notion d'échelle de grandeur nous a occupé l'esprit et que la notion de similarité entre des échelles de grandeur est une notion que nous aimerions retrouver dans le Go. Dans la nature on dit que l'on passe d'une échelle à une autre lorsqu'un système est constitué de milliers, de millions (voire beaucoup plus) d'éléments. Le problème est que le goban est trop petit avec ses 19 lignes et 19 colonnes, il n'y a essentiellement que trois (deux?) échelles: celle de l'intersection, la "locale" (avec les groupes et les territoires) et la "globale". Nous voudrions trouver des analogies entre l'échelle des groupes et l'échelle des pierres.

Nous présentons ici une similarité entre le modèle sur les groupes et les pierres du goban.

<u>Une similarité entre le modèle des groupes et les pierres du goban</u>

Voici une figure quelconque, courante au jeu de Go:

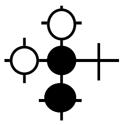
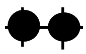
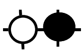
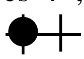


figure 14

A première vue, nous voyons des pierres de Go. Mais que se passe-t-il si nous regardons la figure 14 à l'échelle des groupes, c'est-à-dire si nous imaginons que chaque rond représente non pas une pierre mais un groupe de pierres ?

A cette échelle, le trait qui relie deux ronds représente alors une interaction entre deux groupes.

Nous identifions alors des groupes , des interactions amies , des interactions ennemies  et des interactions vides  comme dans notre modèle. La figure 14 peut être vue indifféremment à l'échelle des pierres ou à l'échelle des groupes. Nous observons donc une similarité entre deux échelles caractéristiques du jeu de Go : l'échelle des pierres et l'échelle des

[1]Un objet est fractal, non pas s'il est similaire à différentes échelles, mais si sa dimension de Besicovitch-Haussdorf est strictement inférieure à sa dimension topologique.

groupes. Le chapitre qui présente notre modèle définit ce que nous appelons les groupes et montre comment nous avons classifié les propriétés d'un groupe en étant guidé par cette similarité d'échelle. Une propriété caractéristique de nombreux systèmes physiques ou naturels soumis à des contraintes est d'évoluer vers des états critiques où l'on peut observer des similarités entre différentes échelles [Wilson 75].

Cette similarité nous conforte dans l'idée que les concepts identifiés par notre méthode expérimentale d'approximations successives sont ceux que nous devions identifier.

La dimension fractale des objets posés sur le goban :

Mandelbrot donne des moyens empiriques d'approcher la dimension fractale D des objets naturels à partir de relations entre le volume, la surface et la longueur d'un objet:

$$\text{volume}^{1/3} \approx \text{surface}^{1/D}$$
$$\text{surface}^{1/2} \approx \text{longueur}^{1/D}$$

Avec la deuxième équation, nous avons mesuré expérimentalement ces grandeurs pour avoir une approximation de D, la dimension fractale des clusters au Go: un groupe avec ses zones d'ombre. Pour des parties de Shusaku [Power] nous avons mesuré $D \approx 1.8$. Mais nous pensons que cette mesure est biaisée par le fait que le goban est trop petit. Une publication existe à ce sujet [Yang & Yao 1991]. Les auteurs trouvent 1.82 et se félicitent. Mais ce nombre est-il "sensiblement inférieur" à 2 par la taille trop petite du goban (19-19) ou par une réelle dimension fractale des objets au Go ? Il faudrait mesurer sur des gobans très grands (100-100 au moins). Il faut que l'humanité joue au Go encore longtemps avant de maîtriser ce qui ce passe sur ce type de goban et que l'on puisse alors faire des mesures significatives.

Bibliographie

[Mandelbrot 1982] - B. Mandelbrot - The fractal geometry of Nature - San Fransisco, Freeman, 1982

[Yang & Yao 1991] - Z.J. Yang, J. Yao - Cluster dimensionality in the game of go - Physica A 176, 447-453, North-Holland - 1991

[Wilson] - Kenneth Wilson - Les phénomènes de physique et les échelles de longueur - Ordre et Chaos - Pour la science

David Marr

Les travaux effectués en vision par David Marr ont fortement influencé notre travail [Marr 1982]. Nous présentons seulement l'influence visible de Marr sur notre travail : la structuration du modèle en niveaux conceptuels.

Les niveaux conceptuels de David Marr

Image

Les points (pixels)

Primal sketch

"Zero-crossings", "blobs", terminaisons et discontinuités, segments de bord, lignes virtuelles, groupes, organisation curviligne, frontières.

2.5 sketch

Orientation locale des surfaces, distance au sujet, discontinuités en profondeur, discontinuités en orientation de surface

3D model representation

Modèle à 3 dimensions arrangé hiérarchiquement.

Les niveaux conceptuels dans INDIGO

Les niveaux conceptuels de Marr ont contribué à la structuration de notre modèle en niveaux :

Niveau zéro

Les intersections et les chaînes

Niveau élémentaire

Les connexions, séparations, yeux, contacts, dilatations, points, formes mortes et vivantes

Niveau itératif

Les groupes, territoires, espaces vides, fractions

Niveau global

Le score

Bibliographie

[Marr 1982] - D. Marr, Vision, San Francisco, Freeman & co, 1982

Morphologie mathématique

Introduction générale :

Ce paragraphe sur la morphologie mathématique comporte plusieurs aspects :

Comment utiliser des opérateurs de morphologie mathématique pour modéliser le territoire et l'influence.
Comment la morphologie mathématique nous conforte dans l'utilisation du concept de séparation vis-à-vis de celui de connexion.
Comment implémenter les opérateurs de dilatation et d'érosion de façon efficace en C.

Utiliser des opérateurs de morphologie mathématique pour modéliser le territoire et l'influence

Introduction

Au paragraphe sur la géométrie fractale, nous avons vu comment la notion de zone d'ombre dans la croissance fractale nous a poussé à modéliser les territoires comme des "zones d'ombre". Comment modéliser une zone d'ombre ?

L'importance des opérations de dilatation et d'érosion pour modéliser des concepts de la règle du jeu (les libertés par exemple sont obtenues en utilisant entre autres la dilatation) nous a poussé à regarder quels outils étaient disponibles en morphologie mathématique [Schmitt 1989] et lesquels pourraient servir.

Il est clair que le modèle de Zobrist [Zobrist 1969] basé sur des dilatations modélise l'influence des groupes au Go. De même les recherches de Pierre Aroutcheff sur l'"aire" d'une intersection [Aroutcheff 1992] utilisent sans le dire le principe de la dilatation morphologique.

En observant que chercher les 4 voisines d'une intersection correspondait à la dilatation morphologique d'une intersection avec pour élément structurant une croix, nous avons pensé que la morphologie mathématique devait être source d'inspiration.

Nous avons repris les opérateurs de base de la morphologie mathématique: dilatation, érosion, ouverture, fermeture. Nous avons essayé de les appliquer à des ensembles d'intersections présents sur un goban: l'ensemble des intersections vides, noires, blanches, occupées, etc... et nous avons regardé ce que cela donnait.

Dans ce paragraphe nous suivons un ordre volontairement didactique illustré par de nombreux exemples.

1er intermède sur la morphologie mathématique

Nous appelons **dilatation morphologique D** l'opérateur qui associe à un ensemble d'intersections I l'ensemble d'intersections constitué par la réunion de I avec l'ensemble des intersections voisines d'une intersection de I au moins.

Nous appelons **érosion morphologique E** l'opérateur qui associe à un ensemble d'intersections I l'ensemble d'intersections constitué par les intersections de I qui ne sont voisines d'aucune intersection de l'ensemble complémentaire de I.

Il existe deux types de libertés

Sur la figure *MM-intérieur* est représentée une chaîne de pierres noires.

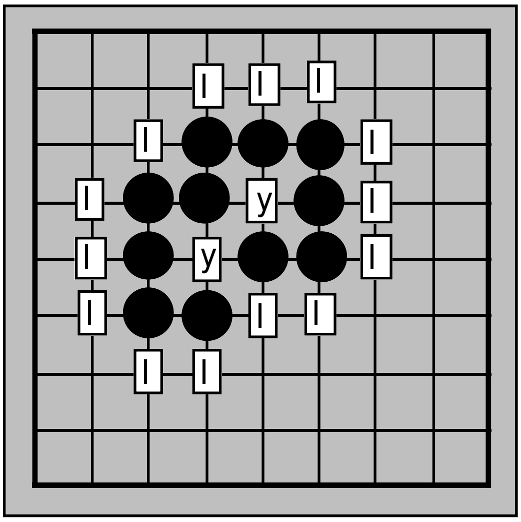


figure MM-intérieur

Cette chaîne possède des libertés ('l' et 'y' sur la figure *MM-intérieur*.). Un joueur de Go les trouve aisément en identifiant les voisines vides de la chaîne. En utilisant le jargon de morphologie mathématique, l'ensemble des libertés est obtenu comme étant l'intersection de l'ensemble des intersections vides et de l'ensemble dilaté de l'ensemble des intersections de la chaîne.

Les joueurs de Go ne donnent pas les mêmes propriétés à toutes ces libertés. Ils appellent les deux libertés indiquées par un 'y' sur la figure des "yeux". (Un oeil est important car une chaîne de pierres qui possède deux yeux est vivante inconditionnellement.)

Si la reconnaissance est naturelle pour un joueur de Go, l'explication est généralement difficile pour lui. Un joueur de Go qui essaye d'expliquer en quoi les libertés 'y' sont différentes des libertés 'l' expliquera que si l'adversaire y joue il n'aura pas de libertés. Ou bien dira que les libertés 'y' sont "intérieures" et les libertés "l" sont extérieures. C'est la dualité **intérieur-extérieur** qui est déterminante et nous intéresse ici.

Quel outil de morphologie mathématique permet de trouver l'ensemble des libertés 'y' à partir de l'ensemble des intersections de la chaîne ?

réponse: L'opérateur de fermeture morphologique

En effet, si l'on applique l'opérateur de dilatation morphologique suivi de l'opérateur d'érosion morphologique (en clair, l'opérateur de fermeture morphologique) et que l'on prend l'intersection avec l'ensemble des intersections vides, on obtient les deux libertés 'y' et seulement elles. Magique, non ?

En fait, cette nuance entre les libertés "intérieures" ('y' sur la figure *MM-intérieur*) et "extérieures" ('l' sur la figure *MM-intérieur*) est très importante. Elle va se retrouver au niveau de ce que les joueurs de Go appellent le "territoire" ou l'"influence".

Le territoire et l'influence

Sur la figure *MM-territoire-1* , l'ensemble représenté par des 'y' matérialise un territoire et l'ensemble représenté par des 'l', l'influence des pierres noires.

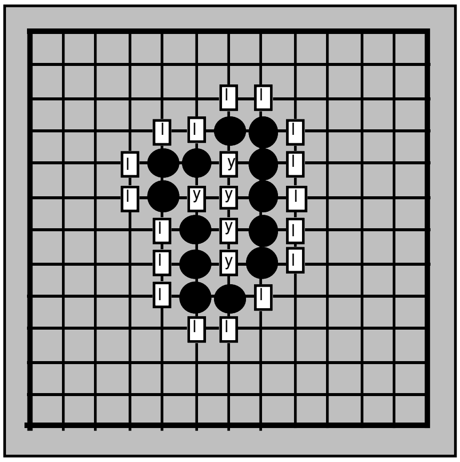


figure MM-territoire-1

Des questions viennent alors immédiatement à l'esprit du programmeur de Go :

A quoi servent d'autres combinaisons d'opérateurs de dilatations et d'érosions ?

Comment modéliser les territoires à plus grandes échelles ?

Il devient nécessaire de définir le vocabulaire de morphologie mathématique utilisé par la suite.

2ème intermède sur la morphologie mathématique

Nous appelons fermeture morphologique F, la composition de l'érosion morphologique E et de la dilatation morphologique D.

$$F = E \circ D$$

Nous appelons ouverture morphologique O, la composition de la dilatation morphologique D et de l'érosion morphologique E.

$$O = D \circ E$$

Nous appelons adhérence morphologique A, la fermeture morphologique F moins l'identité I :

$$A = F - I$$

Nous appelons opérateur morphologique X(m,n), la combinaison de n dilatations morphologiques suivies de m érosions morphologiques :

$$X(m,n) = E^m \circ D^n$$

Nous appelons opérateur morphologique Y(m,n), la combinaison de n érosions morphologiques suivies de m dilatations morphologiques :

$$Y(m,n) = D^m \circ E^n$$

Nous avons évidemment :

$$X(0,0) = Y(0,0) = I$$

$$X(1,1) = F$$
$$Y(1,1) = O$$

L'intérêt des opérateurs X et Y est d'agir sur les ensembles d'intersections à des échelles différentes.

Les opérateurs X servent à modéliser le territoire

Sur la figure *MM-intérieur-2* est représentée à nouveau une chaîne de pierres noires.

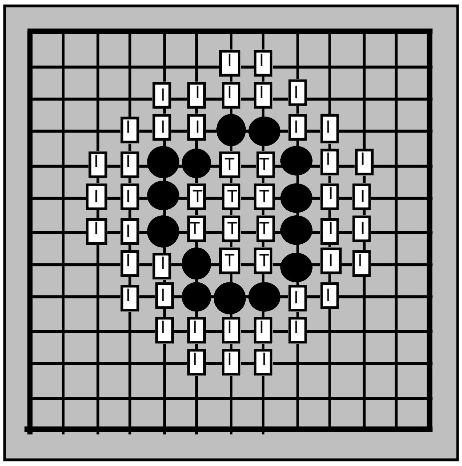


figure MM-intérieur-2

L'opérateur X(2,2) sert à trouver l'ensemble des intersections 'T' de la figure *MM-intérieur-2* qu'un joueur de Go appellera du "territoire". L'opérateur X(2,0) sert à trouver l'ensemble des intersections 'I' ou 'T' de la figure *MM-intérieur-2* ou "influence" des pierres noires.

Que se passe-t-il si nous enlevons des pierres noires ?

La figure *MM-intérieur-2-moins-pierres* est la figure *MM-intérieur-2* moins quelques pierres noires.

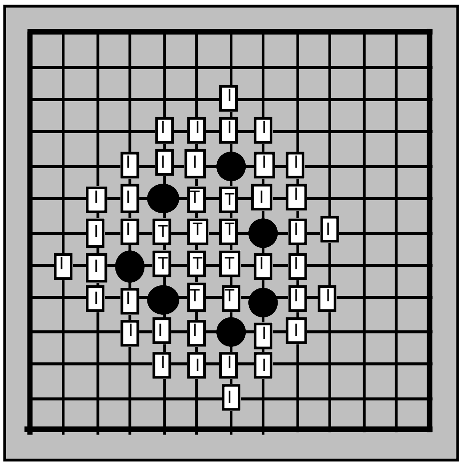


figure MM-intérieur-2-moins-pierres

On remarque que le "territoire" identifié par X(2,2) est strictement identique à celui de la figure *MM-intérieur-2* .

Par contre, il ne faut pas enlever trop de pierres sinon le "territoire" se réduit encore comme sur la figure *MM-intérieur-2-moins-beaucoup-de-pierres*

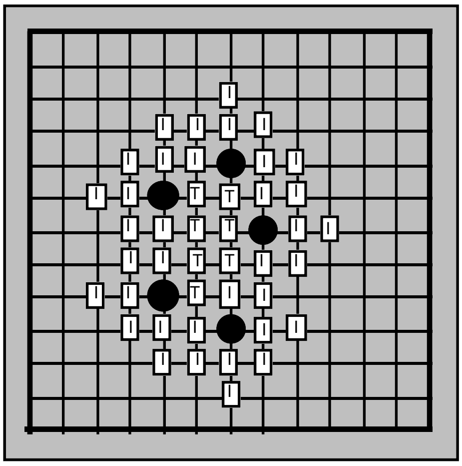


figure MM-intérieur-2-moins-beaucoup-de-pierres

Le "territoire" reconnu par l'opérateur X(2,2) correspond toujours au territoire reconnu par un expert du jeu de Go. Nous voyons que **les opérateurs X(m,n) semblent satisfaisants**.

Sur les figures *MM-extension-bord-3-3* et *MM-extension-bord-3-2* les opérateurs X(3,3) et X(3,2) reconnaissent "assez bien" le territoire contrôlé par les pierres noires :

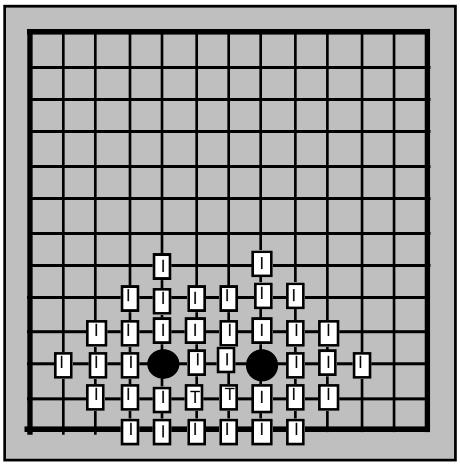


figure MM-extension-bord-3-3

On remarque que l'opérateur X(3,3) marche presque au sens où le "territoire" qu'il reconnaît est proche de celui d'un expert humain. De façon vague et floue un expert dira qu'il manque les deux intersections de la 1ère ligne sous le "territoire" reconnu sur la figure *MM-extension-bord-3-3*.

On se demande si en enlevant une érosion le territoire ne serait pas mieux reconnu :

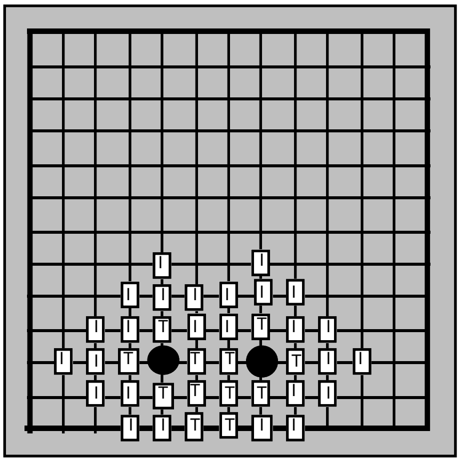


figure MM-extension-bord-3-2

Sur la figure *MM-extension-bord-3-2* le territoire reconnu est trop grand. Ceci attire l'attention sur la relation qui doit exister entre m et n pour que la reconnaissance des "territoires" soit correcte. En fait, X(3,2) donne un territoire trop grand et X(3,3) un territoire trop petit. L'avantage de X(3,3) sur X(3,2) est de donner l'identité sur la position triviale de la figure *MM-une-pierre-seule*.

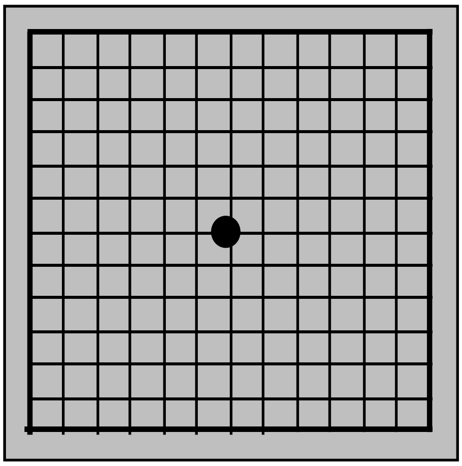


figure MM-une-pierre-seule

Nous avons suivi le principe d'identité pour que des territoires n'apparaissent pas là où ils n'existent pas.

<u>Le principe d'identité</u>

On accepte pour valeurs de m et n des valeurs telles que X(m,n) appliqué à une position du type de celle de la figure *MM-une-pierre-seule* (un goban avec une pierre au milieu) soit l'identité. Dans les conditions de ci-dessus, nous avons évidemment n = m. Nous verrons - au paragraphe sur la "logique floue" - lorsque nous ferons la transition symbolique-numérique que l'égalité n = m n'est plus vraie et qu'elle doit être remplacée par n = 1+m(m-1).

Conclusion

Nous avons montré par des exemples que les opérateurs de morphologie mathématique sont très utiles pour reconnaître des objets familiers pour un joueur de Go.

L'influence à une échelle donnée d'un groupe est le dilaté d'ordre N du groupe.

Le territoire à une échelle donnée d'un groupe est la fermeture d'ordres M et N du groupe.

Si l'on veut aller plus loin et reconnaître des territoires sur des positions issues de parties réelles, la morphologie mathématique "symbolique", telle qu'elle a été présentée ci-dessus, ne marche pas aussi bien que sur les cas d'école présentés ci-dessus. Les positions issues de parties réelles sont bruitées. Pour y reconnaître des territoires, un lissage numérique est nécéssaire. Nous montrerons au paragraphe "logique floue" (ou "morphologie mathématique floue" [Bloch & Maître 1992]) comment reconnaître des territoires sur des positions de parties réelles.

Confirmer l'utilité de concepts déjà identifiés par ailleurs : le concept de séparation par rapport à celui de connexion

Il est toujours difficile de cerner d'où nous provient une idée. Nous n'avons pas toujours conscience de son origine (jamais?). En fait, nous pensons qu'une idée vient progressivement, se confirmant ou s'infirmant au gré des expériences vécues. Ce paragraphe illustre ce fait sur une question que nous nous sommes posée pendant longtemps au cours de notre thèse :

Faut-il mettre ou non le concept de séparation dans notre modèle, sachant que nous avons déjà mis celui de connexion et que "Si on est connecté, on sépare l'adversaire".

En effet, de nombreuses discussions avec des joueurs de Go nous ont montré que beaucoup d'entre eux ne voyaient pas l'utilité des concepts de *zone* et *séparation* . Ces concepts sont en partie cachés par les concepts visibles de *groupe* et *connexion.*

Nous montrons comment la morphologie mathématique tranche sur l'utilité du concept de séparation.

Le théorème de Jordan [Serra 1982] dans un espace continu dit que :

Une courbe fermée sépare l'espace en deux composantes connexes.

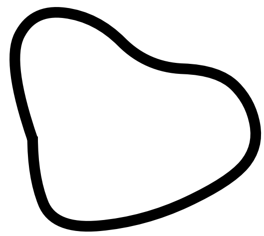


figure MM-Jordan-continu

Dans un espace discret, le théorème de Jordan doit être adapté en fonction de la connectivité de l'espace et de la courbe considérés. Cela pose de petits problèmes (les courbes ne doivent pas être trop "petites" ni trop "alambiquées"...) que nous ne détaillerons pas. Pour pouvoir énoncer ce théorème en espace discret, nous préfixons les énoncés avec "en général" :

En général, une courbe 8-connexe sépare l'espace en deux composantes 4-connexes.
En général, une courbe 6-connexe sépare l'espace en deux composantes 6-connexes.
En général, une courbe 4-connexe sépare l'espace en deux composantes 8-connexes.

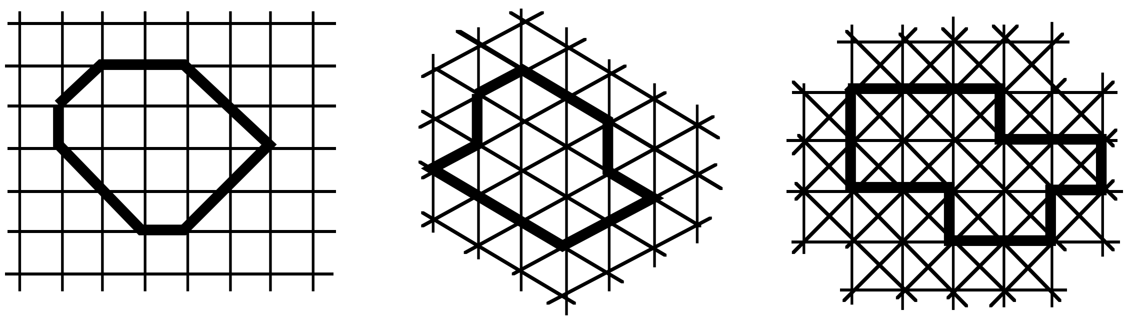


figure MM-Jordan-discret

La figure *MM-Jordan-discret* résume sous forme visuelle ces 3 adaptations du théorème de Jordan.

Un joueur de Go interprétera le dessin de gauche en disant que sur un goban (un espace 4-connexe), des pierres placées en nobi[1] ou en kosumi (8-connexité) séparent le goban en deux zones. Pour cette raison, nous affirmons que *le théorème de Jordan tranche en faveur de l'utilité du concept de séparation* (au Go 4-connexe). Le concept de connexion est plus fort que celui de séparation, ce que l'on peut traduire par :

connexion > séparation

Le dessin du milieu montre que sur un goban hexagonal (6-connexité), la connexion est strictement équivalente à la séparation en deux zones. Sur un goban hexagonal, le concept de séparation est inutile. On a :

connexion = séparation

Le dessin de droite montre que la 4-connexion entraîne la séparation en deux zones 8-connectées. La 8-connexion ne suffit pas à séparer.

connexion < séparation

Ces remarques sur des connectivités différentes de la connectivité usuelle pour le Go, peuvent nous faire réfléchir sur le Go. Les gobans sont-ils nés 4-connexes par hasard ? A-t-on réfléchi à tout cela avant ? Le jeu de Go sur un goban 6-connexe serait-il moins intéressant et le jeu de Go sur un goban 8-connexe complètement inintéressant ?

[1]Deux pierres voisines

Une implémentation efficace des opérateurs de dilatations et d'érosions en C

L'utilisation classique mais brutale[1] des opérateurs morphologiques est très coûteuse dès que l'ensemble de départ est grand. Le module dont nous parlons dans ce paragraphe est un module de morphologie mathématique indépendant du Go où les éléments des ensembles représentés peuvent prendre les valeurs 0 ou 1. Le cas du Go où les valeurs sont vide, blanc, noir s'y ramène facilement. Nous avons trouvé une représentation adaptée pour utiliser ces opérateurs de manière efficace. L'idée est premièrement de paralléliser les opérations en représentant un objet posé sur le goban par deux vecteurs d'entiers où chaque entier du premier (resp. deuxième) vecteur représente une ligne horizontale (resp. verticale) du goban. Deuxièmement, d'utiliser les opérateurs C >>, <<, | et &. En effet, si x représente une ligne du goban alors (x>>1) | (x<<1) [respectivement (x>>1) & (x<<1)], représente la ligne dilatée [respectivement érodée]. Pour obtenir le dilaté ou l'érodé d'un objet en dimension 2, on recombine comme il faut les lignes et les colonnes dilatées avec les opérateurs | et &. Le code C++ correspondant se trouve en annexe. Le coût en temps machine d'une dilatation ou d'une érosion devient alors indépendant de la taille de l'objet de départ, seulement de la taille du goban. Évidemment, pour opérer sur des ensembles petits, il est plus intéressant d'utiliser la représentation classique.

Bibliographie

[Serra 1982] - J. Serra - Image Analysis and Mathematical Morphology - Academic Press - London - 1982

[Aroutcheff 1992] - P. Aroutcheff - Une théorie du Go - non publié - 1992

[Zobrist 1969] - A. Zobrist, A model of visual organization for the game of Go, Proceedings AFIPS 34 103-112, 1969

[Bloch & Maître 1992] - I. Bloch, H. Maître - Ensembles flous et morphologie mathématique - Télécom Paris 92 D007 - Département Images, Groupe Image - Mars 1992

[Schmitt 1989] - M. Schmitt - Des algorithmes morphologiques à l'intelligence artificielle - Thèse, Ecole des Mines de Paris, fev. 1989

[1]Pour dilater un ensemble d'intersections, boucler sur toutes les intersections de l'ensemble, en mettant l'intersection et les voisines de l'intersection dans l'ensemble dilaté.

La logique floue

Introduction

Une des capacités des joueurs de Go humains est d'une part d'avoir une vision floue et vague des objets reconnus sur le goban et d'autre part de faire un raisonnement flou sur ces objets. INDIGO n'utilise pas de logique floue à proprement parler, mais il nous paraît intéressant de présenter ce qui se rapproche de la logique floue dans INDIGO. Dans INDIGO, la reconnaissance des territoires utilise de la morphologie mathématique que nous appelons floue car c'est une numérisation de la morphologie mathématique symbolique. D'autre part, le raisonnement ludique d'INDIGO utilise des symboles "imprécis". Ces symboles permettent de limiter les calculs coûteux pour ne pas perdre de temps. Ils permettent aussi de réduire le nombre de cas possibles lorsque des synthèses sur des résultats de jeux sont effectués.

Le but de ce paragraphe est de montrer comment la logique floue intervient dans la vision du goban dans INDIGO. Le raisonnement ludique flou dans INDIGO est présenté dans le chapitre sur l'évaluation de notre travail.

La vision floue du goban ou la modélisation des territoires, des influences et des espaces neutres par morphologie mathématique floue.

La vision d'un joueur de Go sur un goban est un pré traitement qui permet de savoir :
- précisément où identifier des groupes (là où sont les pierres !!!),
- de façon floue où identifier des territoires et des espaces vides.

Le but de ce paragraphe est de montrer comment la logique floue permet d'affiner l'apport de la morphologie mathématique pour la reconnaissance de l'influence et des territoires.

Le modèle de Zobrist se rapproche de la morphologie mathématique floue.

Chronologiquement, nous nous sommes inspiré du modèle de Zobrist [Zobrist 1969] pour modéliser l'influence qui est un modèle numérique. L'avantage de numériser l'information est de stabiliser la reconnaissance des objets.

Description de notre algorithme

Initialisation

Notre algorithme affecte +128 (resp. -128) aux intersections noires (resp. blanches) et 0 aux intersections vides et il effectue autant de dilatations et d'érosions qu'il est demandé. Pour reconnaître des influences à échelle N, il faut N dilatations et 0 érosion. Pour reconnaître des territoires à échelle N, il faut N dilatations et 1+N(N-1) érosions[1].

Dilatation

Pour chaque intersection du goban, si une intersection est positive (resp. négative) ou nulle et n'est pas voisine d'une intersection négative (resp. positive), il additionne (resp. soustrait) le nombre de voisines positives (resp. négative) à l'intersection.

[1] Pour respecter le principe d'identité de la position avec une pierre seule (cf. paragraphe précédent sur la morphologie mathématique tout court).

Exemple élémentaire

Position initiale, un ikken-tobi, 0 dilatation :

			128		128			

1 dilatation :

			1		1			
		1	128	2	128	1		
			1		1			

2 dilatations :

			1		1			
		2	2	3	2	2		
	1	2	132	4	132	2	1	
		2	2	3	2	2		
			1		1			

3 dilatations :

			1		1			
		2	2	3	2	2		
	2	4	6	6	6	4	2	
1	2	6	136	8	136	6	2	1
	2	4	6	6	6	4	2	
		2	2	3	2	2		
			1		1			

Erosion

Pour chaque intersection du goban, si une intersection est positive (resp. négative), il soustrait (resp. additionne) le nombre de voisines négatives (resp. positives) ou nulles à l'intersection en vérifiant qu'elle reste positive (resp. négative) ou nulle.

Suite de l'exemple élémentaire

3 dilatations et 1 érosion :

			2	2	2			
		4	6	6	6	4		
	2	6	136	8	136	6	2	
		4	6	6	6	4		
			2	2	2			

3 dilatations et 2 érosions :

				1				
		2	6	6	6	2		
		6	136	8	136	6		
		2	6	6	6	2		
				1				

3 dilatations et 3 érosions :

	5	6	5	
5	136	8	136	5
	5	6	5	

3 dilatations et 4 érosions :

	3	5	3	
2	136	8	136	2
	3	5	3	

3 dilatations et 5 érosions :

1	4	1
136	8	136
1	4	1

3 dilatations et 6 érosions :

	3	
135	8	135
	3	

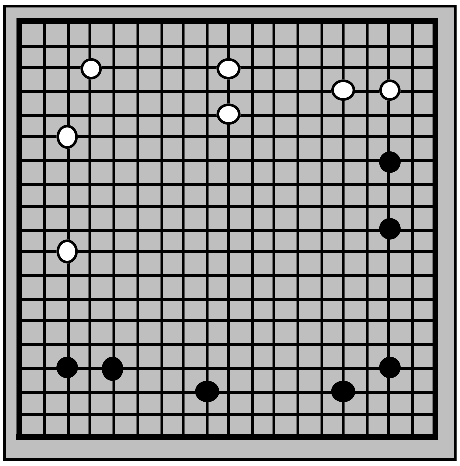
3 dilatations et 7 érosions :

132	8	132

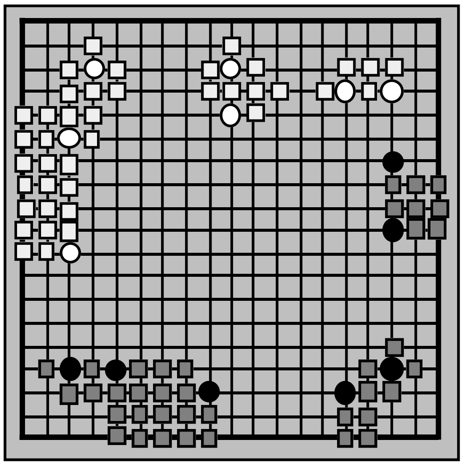
Exemples de positions réelles

Nous donnons quatre positions issues de parties de Shusaku, un joueur talentueux du XVIII ème siècle qui était invincible avec Noir [Power 1975]. Nous donnons les résultats des opérateurs X(nombre_dilatations, nombre_érosions) pour les valeurs de nombre_dilatations et nombre_érosions qui donnent les meilleurs résultats du point de vue de la reconnaissance des territoires par un joueur de Go humain. En plus, nous donnons les résultats d'un opérateur X(nombre_dilatations, 0) pour montrer visuellement la différence entre, influence et territoire d'une part, et dilatations morphologiques et fermetures morphologiques d'autre part.

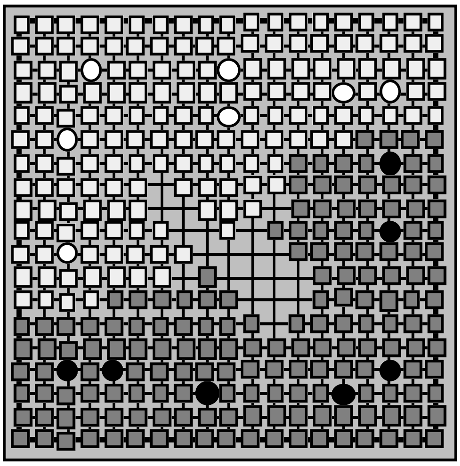
Position 1 :



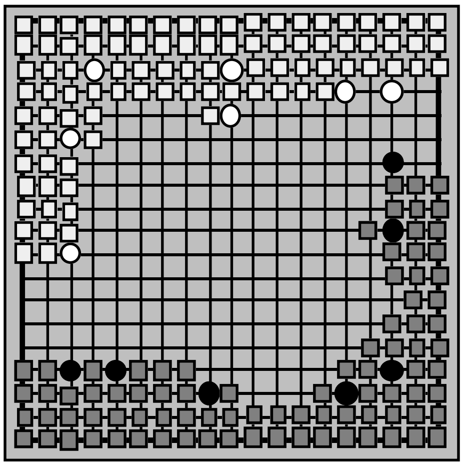
position 1



4 dilatations, 13 érosions



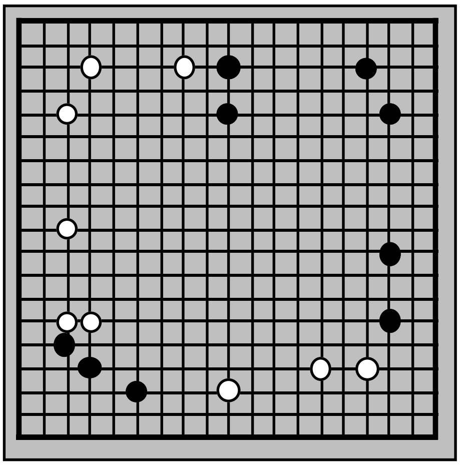
5 dilatations



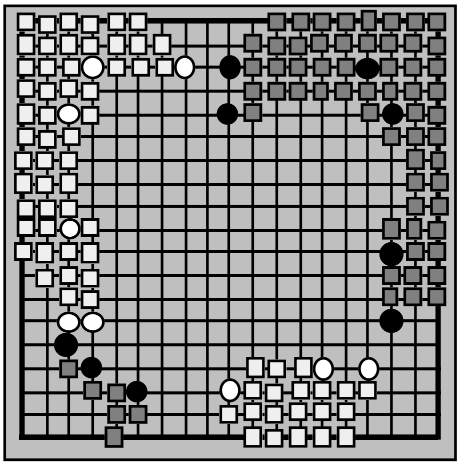
5 dilatations, 21 érosions

X(4, 13) ne suffit pas pour reconnaître les territoires. X(5, 21) convient. L'influence associée montrée par X(5, 0) montre un partage du goban possible mais nous pensons qu'il ne correspond pas à grand chose pour un joueur de Go.

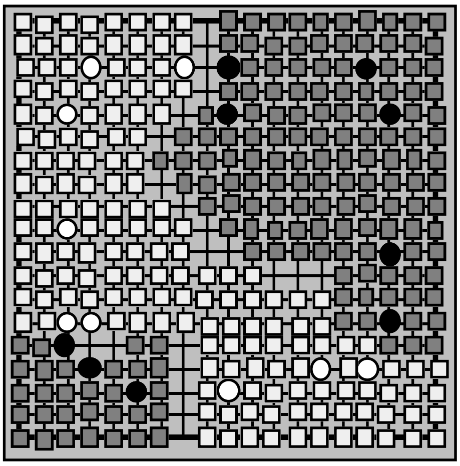
Position 2 :



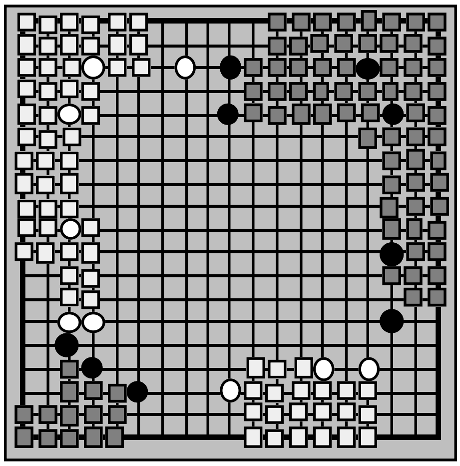
position 2



5 dilatations, 21 érosions



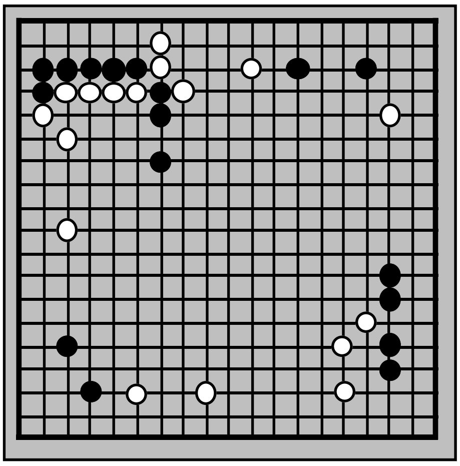
6 dilatations



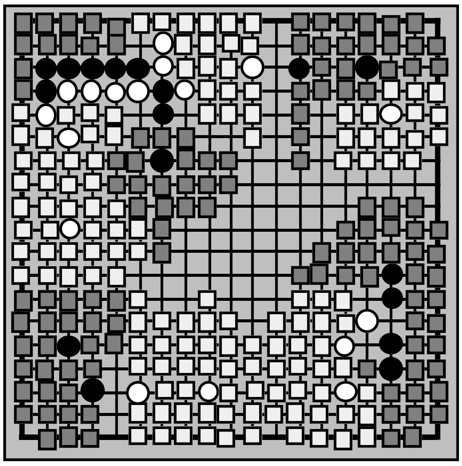
6 dilatations, 31 érosions

Ici X(5, 21) convient presque. X(6, 31) convient mieux. L'influence X(6, 0) ne correspond à rien.

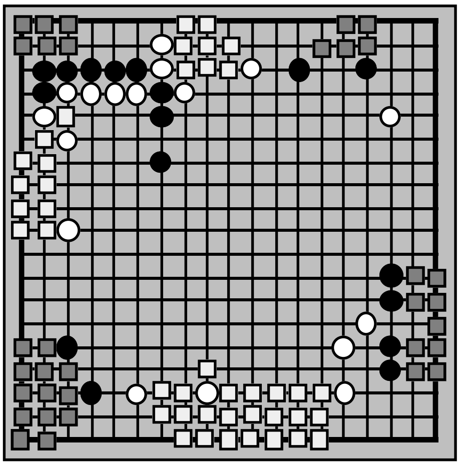
Position 3 :



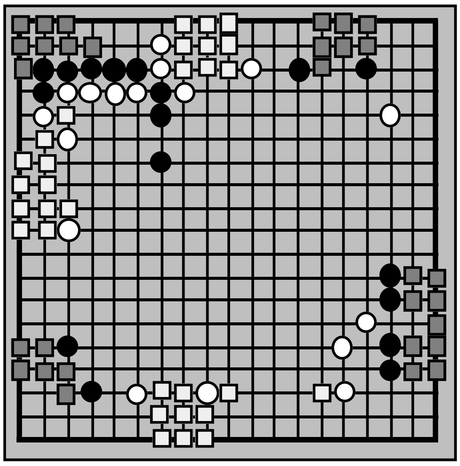
position 3



4 dilatations



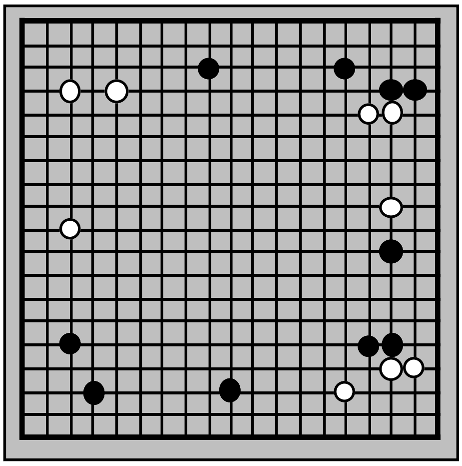
5 dilatations, 21 érosions



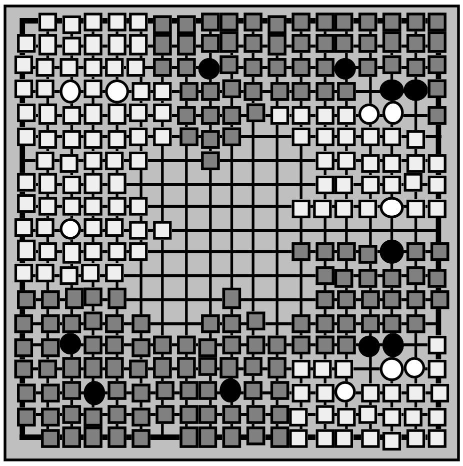
4 dilatations, 13 érosions

X(4, 13) et X(5, 21) conviennent assez bien car les pierres du fuseki sont basses (sur la 3ème ligne). A nouveau, X(4, 0) ne correspond pas à grand chose.

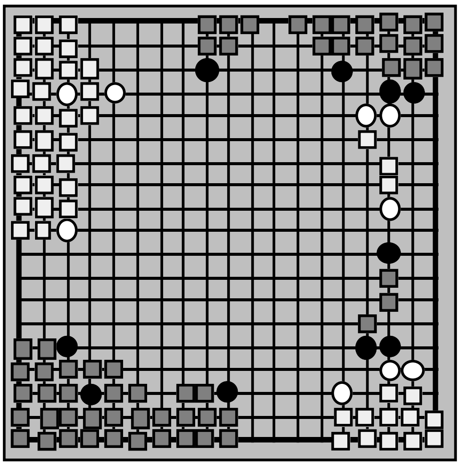
Position 4 :



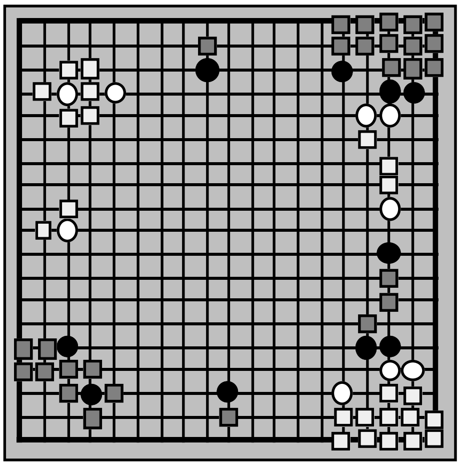
position 4



4 dilatations



5 dilatations, 21 érosions



4 dilatations, 13 érosions

Ici X(4, 13) convient presque pour les territoires. X(5, 21) convient mieux. X(4,0) correspond un peu à de l'influence car la position est assez simple.

Avantages et limites de l'approche

Avantages de la morphologie mathématique floue appliquée au Go

Savoir que la morphologie mathématique floue [Bloch & Maître 1992] existait m'a conforté dans l'utilité du modèle de Zobrist et sa généralisation grâce à la morphologie mathématique. Pour toutes ces positions , il existe un opérateur X(n, 1+n(n-1)) qui correspond au territoire.

Limites de la morphologie mathématique floue appliquée au Go

Mais les problèmes sont les suivants. Pour une position - et même pour un morceau de la position - il faut trouver les bonnes valeurs de n qui reconnaisse les territoires. Notre modèle ne trouve pas ces bonnes valeurs lui-même. Elles sont fixées une fois pour toute au début du programme. Même si l'on trouve ces bonnes valeurs, il ne faut pas qu'elles soient trop grandes car la reconnaissance peut être très coûteuse et nécessiter des temps de réponse trop longs du programme qui joue.

Bibliographie

[Bloch & Maître 1992] - I. Bloch, H. Maître - Ensembles flous et morphologie mathématique - Internal report Télécom Paris, 92D007, 1992

[Power 1982] - J. Power - Invincible, the games of Shusaku - Ishi Press, 1982

[Zobrist 1969] - A. Zobrist, A model of visual organization for the game of Go, Proceedings AFIPS 34 103-112, 1969

ETAT DE L'ART DE LA PROGRAMMATION DU JEU DE GO

Introduction

L'expression "Programmation du jeu de Go".

Au sens large, l'expression "Programmation du jeu de Go" englobent à la fois le problème de la **programmation d'une partie complète sur goban 19-19** et les sous-problèmes du jeu de Go tels que les **problèmes de fin de partie** et les **problèmes de tsumego**. Du point de vue mathématique, l'expression "programmation du jeu de Go" englobe aussi les problèmes théoriques de complexité posés par le jeu de Go et la théorie de Conway.

Plan du chapitre

Le plan de notre état de l'art sur la programmation du jeu de Go est le suivant

- Jouer une partie complète
  - Historique
  - L'état actuel
- Résoudre des sous-problèmes du jeu de Go
  - Les problèmes de tsumego
  - La toute fin de partie
- L'aspect mathématique
- La recherche en France
- Bibliographie

Jouer une partie complète

Historique

Un très bon point d'entrée sur l'histoire de la programmation du jeu de Go est dans [Hamann 1985]. Le problème du développement d'un programme de Go qui joue une partie complète est bien présenté dans [Kierulf & al 1990] par les auteurs de Smart Game Board, une plate-forme de développement pour le jeu de Go ou d'autres jeux de plateau.

L'histoire de la programmation du jeu de Go remonte au début des années soixante. En 1961, D. Lefkofitz a écrit un premier programme [Lefkovitz 1961]. Remus a écrit un programme qui apprend à jouer [Remus 1962] mais cela restait très rudimentaire. Thorp et Walden ont utilisé la machine pour explorer les stratégies sur des tout petits gobans [Thorp & Walden 1964] [Thorp & Walden 1972]. Zobrist a utilisé une technique de "potentiel visuel" pour écrire son programme [Zobrist 1969]. Sa technique qui entre aujourd'hui dans le moule de la morphologie mathématique est utilisée dans les programmes actuels sous des formes diverses. Ryder s'est intéressé à la structure des arbres engendrés par le Go [Ryder 1971]. Les années soixante-dix sont marquées par les travaux de Bruce Wilcox [Wilcox 1978] [Wilcox 1979], son programme d'alors atteignant le niveau de 25ème kyu. Dans les années quatre-vingt, les premières versions des programmes actuels apparaissent. Mano décrit une approche déclarative intéressante, mais lente, avec le langage Gopal [Mano 1984].

Etat actuel

Le niveau des meilleurs programmes est faible comparé à l'échelle humaine.

La programmation du jeu de Go possède la caractéristique suivante: les meilleurs programmes de Go ont le niveau de 10ème kyu, c'est à dire le niveau d'un joueur moyen en club[1]. Par comparaison, le meilleur programme d'Échecs a le niveau de Grand maître international. Le meilleur programme de Checkers a le niveau du champion du monde. Les meilleurs programmes d'Othello dépassent nettement le champion du monde. L'effort consacré à la programmation du jeu de Go dépasse largement celui de la programmation des Checkers ou de Othello mais est encore largement dépassé par celui de la programmation des Échecs.

Description "vague" des meilleurs programmes

Pour décrire les meilleurs programmes de Go actuels, nous nous basons sur la très faible et très vague littérature sur le sujet. En effet, les auteurs des meilleurs programmes de Go n'ont publié que très peu d'articles sur leur programme. Certains n'ont rien écrit du tout. Les quelques auteurs qui aient tenté de le faire sont restés soigneusement vagues et généraux pour ne pas dévoiler leurs secrets. C'est compréhensible, étant donné les gains substantiels que rapporte la possession d'un des meilleurs programmes du monde.

Goliath

Ce programme a été champion du monde des programmes en 1989, 1990, 1991. Son auteur est Mark Boon, hollandais 5ème dan amateur. Le niveau de Goliath est environ 10ème kyu. Nous pensons que **ce programme est le meilleur programme au monde** même s'il n'a pas participé aux derniers championnats du monde. Mark Boon qui n'a pas travaillé sur son programme en

[1]Précisons tout de même qu'il faut être passionné par le jeu de Go pour atteindre ce niveau.

1992 et 1993 pense qu'il n'est pas intéressant de faire participer un programme qui n'a pas progressé. Actuellement, Mark Boon a créé une entreprise dont le but est de développer un programme 5ème kyu d'ici à 3 ans. Il emploie 4 personnes depuis mi-94. Il pense que la raison pour laquelle le niveau des programmes est bloqué à 10ème kyu est la difficulté de ces programmes à apprendre. Cette affirmation laisse penser que la nouvelle version de Goliath contient des fonctions d'apprentissage. Un aperçu du développement de Goliath est dans [Boon 1991]. Le développement de Goliath a été fait en plusieurs étapes au cours d'une thèse. La première génération était un système expert où le coup joué était simplement celui conseillé par la première règle qui se déclenchait. La deuxième génération interprétait le goban pour jouer le coup. La troisième génération effectue des calculs, de la planification et joue des josekis. Son programme est connu pour avoir été le premier à utiliser intensivement du pattern-matching dont nous avons réutilisé les principes [Boon 1989].

<u>Handtalk</u>

Bénéficiant de l'absence de Goliath, ce programme a été champion du monde des programmes en 1993. En 1994, il est troisième. Son auteur, Mr Chen, est chinois. Le niveau du programme est environ 12ème kyu. Aucune description du programme n'existe. Il a été écrit entièrement en assembleur. Il est très rapide et possède un bon niveau stratégique.

<u>Go Intellect</u>

Ce programme a été 2ème à différents championnats du monde derrière Goliath. Depuis que Goliath ne participe plus, il a été champion du monde en 1992 et 1994. Son auteur est Ken Chen, américain d'origine asiatique, 6ème dan amateur. Le niveau du programme est environ 12ème kyu. Deux articles décrivent le programme. L'identification des groupes faite par le programme est décrite dans [Chen 1989]. Le processus de décision du coup est décrit dans [Chen 1990]. Il est un descendant de Go Explorer.

<u>Star of Poland</u>

Ce programme a été à plusieurs reprises aux places d'honneur du championnat du monde. En 1993, il était deuxième et en 1994, il est quatrième. Son auteur est Janus Kraczek, polonais, 5ème dan amateur. Son niveau est environ 13ème kyu. Une description des heuristiques d'évaluation de la base de vie d'un groupe de pierres est décrit dans [Kraszek 1988]. Une discussion à propos de l'homéostasis est dans [Kraszek 1990]. Kraszek pense que la première difficulté de la programmation du Go est la gestion de la multitude et la diversité des connaissances.

<u>Many Faces of Go</u>

Ce programme a été à plusieurs reprises bien placé au championnat du monde. En 1994, il a terminé deuxième. Son niveau est environ 14ème kyu. Son auteur est David Fotland, américain, 2ème dan amateur. Son auteur décrit assez longuement son programme dans [Fotland 1992] et anime régulièrement la mailing liste computer-go@oxford.uk. Nous pensons que ce programme a progressé courant 1994 car les parties Many Faces of Go - INDIGO de début 1994 nous laissaient penser que Many Faces of Go était environ 16ème kyu sur IGS.

<u>Dragon</u>

Ce programme a été à plusieurs reprises assez bien placé au championnat du monde. La seule référence que nous possédions est [Chin & Yeh 1989].

<u>Swiss Explorer</u>

Ce programme a été à plusieurs reprises assez bien placé au championnat du monde. Son auteur est Martin Müeller, autrichien, 5ème dan amateur. Son niveau est environ 14ème kyu. Il a été développé avec Smart Game Board pendant une thèse. Il est un descendant de Go Explorer. Son auteur tente comme nous d'utiliser la théorie des jeux de Berlekamp pour l'appliquer au Go [Müeller 1993].

Go Generation

Ce programme a été bien placé au championnat du monde 1993. Il a été développé sur machine parallèle dans le cadre du projet japonais de langage de 5ème génération. Son niveau est environ 15ème kyu. Un des participants japonais est Shirayanagi qui a écrit un article sur la représentation des connaissances par raffinements successifs [Shirayanagi 1989].

Poka

Ce programme a participé à plusieurs reprises au championnat du monde. Son auteur est Howard Landman, américain, 5ème dan amateur. Son niveau est environ 17ème kyu. Son auteur s'intéresse de près à la théorie des jeux appliquée au jeu de Go.

Nemesis

Ce programme a été développé dans les années soixante-dix par Bruce Wilcox, américain, 5ème dan amateur. Son niveau est 17ème kyu environ. Son auteur a écrit de nombreux articles [Wilcox 1978] [Wilcox 1979] et développé une théorie du jeu de Go appelée Instant Go.

Gogol

Ce programme est développé par Tristan Cazenave, français, 2ème kyu. Son niveau est approximativement 17ème kyu. Il utilise entre autres des techniques d'acquisition de connaissances automatique [Cazenave 1994]. C'est le partenaire artificiel favori de Indigo. Ils ont joué plus de 50 parties sur 19-19 entre Septembre 93 et Juillet 94.

INDIGO

Indigo est développé par nous-même, 3ème dan amateur. Le niveau de INDIGO est approximativement 18ème kyu. Il est décrit dans ce document. La technique d'explicitation des connaissances est décrite dans [Bouzy 1994a]. Une description du niveau "groupe" figure aussi dans [Bouzy 1994b].

Résoudre des sous-problèmes

Toute fin de partie

Sur des fins de parties où le jeu de Go est décomposable en sous-jeux indépendants, la machine peut jouer mieux que les meilleurs joueurs de Go professionnels [Berlekamp 1991] [Berlekamp & Wolfe 1994]. En pratique, les obstacles qui rendent inutilisables la théorie de Berlekamp dans une partie complète sont : la difficulté de modéliser l'information en entrée de la théorie (la reconnaissance des groupes et leur état, le découpage du jeu global en sous-jeux), l'hypothèse forte d'indépendance entre les jeux en entrée (comment le savoir a priori ?), la restriction du champ d'application de la théorie au tout petit yose (5 à 10 coups à la fin sur 150 coups au total, soit environ 5% des coups) et, paradoxalement, la perfection des résultats de la théorie (qui serait sans rapport avec la médiocrité des résultats obtenus pendant le reste de la partie).

Tsumego

Sur des problèmes de base de vie de groupes, Risiko résout [Wolf 1992] et engendre [Wolf 1993] des problèmes du niveau 1er à 5ème dan amateur. En pratique, les obstacles qui rendent inutilisable un programme de tsumego comme Risiko, dans un programme qui joue une partie complète, sont analogues à ceux énoncés pour la théorie de Berlekamp : la difficulté de modéliser l'information en entrée du programme de tsumego (la reconnaissance des groupes, leur encerclement), la restriction du champ d'application aux groupes instables *complètement* encerclés (en partie, 95% des problèmes à résoudre s'appliquent à des groupes instables *partiellement* encerclés) et, paradoxalement, l'excellent niveau des résultats de Risiko (qui serait sans rapport avec le faible niveau obtenu par ailleurs).

La théorie de Benson qui énonce des théorèmes sur la vie et la mort [Benson 1976] [Benson 1979] est en partie utilisée par les programmes de Go. Encore une fois, ce type de théorie est inutilisable telle quelle dans un programme de Go qui joue une partie complète.

L'aspect mathématique

La théorie des jeux de Conway

La théorie mathématique qui est la plus utile au jeu de Go est la théorie des jeux de Conway [Conway 1982]. Nous présentons cette théorie utilisée dans la programmation du jeu de Go au paragraphe Domaines Voisins - Théorie des jeux.

Les premières théories

Au début de la programmation du jeu de Go, plusieurs articles théoriques sont parus mais ils ne nous ont pas aidé du tout dans notre démarche pratique. Nous les listons ci-dessous pour information uniquement. Le Go est de complexité polynomiale en espace [Lichtenstein & Spiser 1978]. Etant donné un goban N*N avec une position, il est "complet en temps exponentiel de décider si Noir a une victoire forcée ou de décider le meilleur coup de Noir" [Robson 1983].

En France

Historique

En France, différents travaux et séminaires ont été organisés. Hervé Dicky et Denis Feldmann, se sont intéressés au problème dans les années 1970. Dans le début des années 80, Jean-François Alleton a commencé à écrire un programme de Go [Alleton 1983]. Pierre Aroutcheff s'est intéressé aux propriétés des intersections et des pierres et aux fonctions de potentiel [Aroutcheff 1992]. François Myzessin a écrit un programme en Pascal. En 1986, au cours d'un stage de fin d'année à l'ENSIMAG, Jacques Danysz, encadré par Philippe Bizard, s'est intéressé à des concepts topologiques et géométriques sur les groupes de pierres [Danysz & Bizard 1986]. Une journée sur la programmation du jeu de Go a été organisée en 1989 par Jean Michel à l'E.N.S. En 1990, au cours d'un stage de DEA, Laurent Demailly, encadré par Bernard Victorri, a travaillé sur les réseaux de neurones appliqués au début de partie [Demailly 1990] . Une journée internationale a été organisée à deux reprises, en 1992 et en 1993, par Franck Lebastard à l'INRIA [Méhat 1993] [Victorri 1993] [Pompidor 1993]. En 1990 et 1991, Pierre Pompidor a fait une thèse sur l'apprentissage symbolique à partir d'exemples géométriques appliqués au jeu de Go [Pompidor 1993].

Depuis Septembre 91, nous avons donc fait une thèse sur la modélisation du jeu de Go en simulant le comportement humain sous la direction de Jacques Pitrat. Nous avons utilisé une technique d'explicitation de connaissances basée sur une méthode informatique [Bouzy 1994a]. Nous avons présenté la modélisation des groupes dans [Bouzy 1994b]. Depuis Septembre 91, Patrick Ricaud, également dirigé par Jacques Pitrat, fait une thèse sur l'abstraction et la stratégie appliquées au début de partie du jeu de Go. Depuis Septembre 1993, Tristan Cazenave, aussi dirigé par Jacques Pitrat, fait une thèse sur l'apprentissage appliqué au jeu de Go [Cazenave 1994]. Il écrit le programme de Go Gogol.

Bibliographie française :

[Alleton 1983] - Alleton J.F.: Go et informatique, Regards sur le Go, Revue Française de Go - Mars 1983.

[Aroutcheff 1992] - P. Aroutcheff - Une théorie du go - non publié - 1992

[Bouzy 1994a] - B. Bouzy - Explicitation de connaissances du joueur de Go - Rapport interne du LAFORIA - Septembre 1994

[Bouzy 1994b] - B. Bouzy - Modélisation des groupes au jeu de Go - Rapport interne du LAFORIA - Septembre 1994

[Cazenave 1994] - T. Cazenave - Un système apprenant à jouer au Go - 2ème Rencontres Nationales des Jeunes Chercheurs en IA - Marseille 1994

[Danysz & Bizard 1986] - J. Danysz, P. Bizard - Géométrie et jeu de go, analyse d'une situation - Projet de 3ème année ENSIMAG, 1986

[Demailly 1990] - Laurent Demailly - Rapport de stage de DEA, Projet d'informatisation du jeu de Go - 1990

[Méhat 1993] - J. Méhat - Checking static life of Groups on a SIMD Computer in the Game of Go - Cannes Workshop Computer Go - 1993

[Pompidor 1991] - P. Pompidor - Apprentissage symbolique par exemples et contre-exemples géométrisables en prise de décision: le système Fongus - Journées françaises de l'apprentissage, Sète 1991

[Pompidor 1993] - P. Pompidor - Protocole et implémentation multi-agent pour le temps réel : application à la prise décision planaire - Cannes Workshop Computer Go - 1993

[Victorri 1993] - B. Victorri - Eléments d'une théorie géométrique du jeu de Go - Cannes Workshop Computer Go - 1993

Bibliographie

[Benson 1976] - D.B. Benson, Life in the game of Go, Information Sciences 10, 17-29, 1976

[Benson 1979] - D.B. Benson B.R. Hilditch J.D. Starkey - Tree analysis techniques in Tsumego, IJCAI 79,1, 1979

[Berlekamp 1991] - E.R. Berlekamp - Introductory overview of mathematical Go endgames - Proceedings of symposia in applied mathematics - 43 - 1991

[Berlekamp & Wolfe 1994] - E.R. Berlekamp, D. Wolfe - Mathematical Go Endgames - Nightmares for the Professional Go Player - Ishi Press International - San Jose, London, Tokyo - 1994

[Boon 1989] - M. Boon - Pattern matcher of Goliath - Computer Go 13, winter 89-90

[Boon 1991] - M. Boon - Overzicht van de ontwikkeling van een Go spelend programma - Afstudeer scriptie informatica onder begeleiding van prof. J. Bergstra - 1991

[Bouzy 1994a] - B. Bouzy - Explicitation de connaissances du joueur de Go - Rapport interne du LAFORIA - Septembre 1994

[Bouzy 1994b] - B. Bouzy - Modélisation des groupes au jeu de Go - Rapport interne du LAFORIA - Septembre 1994

[Cazenave 1994] - T. Cazenave - Un système apprenant à jouer au Go - 2ème Rencontres Nationales des Jeunes Chercheurs en IA - Marseille 1994

[Chin & Yeh 1989] - Chin & Yeh - Design and construction of Dragon - Computer Go 10, spring 89

[Chen 1989] - K. Chen - Group identification in computer Go - Heuristic programming in Artificial Intelligence 1 - D.N.L. Levy &Beal D.F. (éds) Ellis-Horwood Prenctice-Hall 1989

[Chen 1990] - K. Chen - Move decision process of Go intellect - Computer Go 14, spring 90

[Conway 1982] - J. Conway, E.R. Berlekamp, R. Guy - Winnings ways - tome 1 & 2 - Academic press - 1982

[Fotland 1992] - D. Fotland - Many Faces of Go, documentation and playing algorithm - 1992

[Hamann 1985] - Hamann C.M. - Chronologie der programierung des japonischen Brettspiels Go, eine herausforderung an die Kunstliche Intelligenz - Angewandte informatik, 27,12, 1985

[Kierulf & al 1990] - A. Kierulf, K.Chen, J. Nievergelt - Smart game board and Go explorer: a study in software and knowledge engineering - Communications of the ACM, 33, 2, 1990

[Kraszek 1988] - J. Kraszek - Heuristics in the life and death algorithm - Computer Go n°9, winter 88-89 [8]

[Kraszek 1990] - J. Kraszek - Looking for resources in AI - Computer Go 14, spring 90

[Lefkovitz 1960] - D. Lefkovitz, A strategic pattern recognition program for the game of Go. University of Pennsylvania, the Moore school of Electrical Engineering, Wright Air Development Division, Technical note 60-243, 1-92, 1960

[Lichtenstein & Spiser 1978] - D. Liechtenstein M. Spiser - Go is polynomial-space hard, 19th Annual symposium on foundation of Computer Science, 48-54, 1978

[Mano 1984] - Y. Mano - An approach to conquer difficulties in developping a Go playing program, Journal Information processing, 7, 2, 1984

[Müeller 1993] - M. Müeller - Game Theories and Computer Go - Cannes Workshop Computer Go - 1993

[Remus 1962] - H. Remus, Lernversuche an der IBM704. Lernende Automaten 144-154, 1961

[Robson 1983] - J.M. Robson, The complexity of Go, Proceedings IFIP, 413-417, 1983

[Ryder 1971] - J.L. Ryder - Heuristic analysis of large trees as generated in the game of Go - Ph.D. dissertation - Stanford University - Microfilm 72-11, 654, 1-298, 1971

[Shirayanagi 1989] - Shirayanagi - Knowledge representation and its refinement - Computer Go 11, summer 89

[Thorp & Walden 1972] - E. O. Thorp, W.E. Walden, A computer assisted study of Go on M*N boards, Information Sciences 4, 1-33, 1972

[Thorp & Walden 1964] - E. O. Thorp, W.E. Walden, A partial analysis of Go, Computer Journal 7, 203-207, 1964

[Wilcox 1978] - B. Wilcox, W. Reitman, Pattern Recognition and pattern directed inference in a program for playing Go. Waterman, D.A., Hayes-Roth, F. (eds): Pattern directed interference systems. Academic press, New York, 503-523, 1978

[Wilcox 1979] - B. Wilcox W. Reitman - The structure of the Go program Interim.2, IJCAI 79,1, 1979

[Wolf 1992] - T. Wolf - Tsumego with Risiko - School of Math. Sciences, Queen mary & Westfield College, Mile end road, London E1 4NS, England - 1992

[Wolf 1993] - T. Wolf - Quality improvements in Tsumego mass production - Cannes Workshop Computer Go - 1993

[Zobrist 1969] - A. Zobrist, A model of visual organization for the game of Go, Proceedings AFIPS 34 103-112, 1969

PARTIE 3 : LE MODÈLE INDIGO

Dans cette partie, nous présentons essentiellement le modèle **computationnel**. A la fin des chapitres ou paragraphes importants, nous effectuons des **correspondances** avec le modèle **cognitif**.

Le modèle computationnel possède des composantes essentielles : la notion d'**objet**, l'utilisation du concept de **jeu** et la **structuration en niveaux**.

Les objets

Notre modèle **reconnaît des objets** sur le goban. Son nom complet est inspiré de la conception orientée objet qui nous a guidé au cours de son implémentation : **MOOD INDIGO** signifie My Object Oriented Design Is Now Designed In Good Objects. Les objets appartiennent à plus de 50 classes. Une taxonomie simplifiée de la hiérarchie des classes figure en conclusion du chapitre.

Le jeu

Nous avons présenté le concept de jeu de façon générale dans le chapitre sur la théorie des jeux dans la deuxième partie du document. Dans INDIGO, **un jeu est associé à une action sur un objet** reconnu sur le goban. Par exemple, le jeu de la santé d'un groupe est associé à l'action de tuer ou sauver un groupe (l'objet fondamental au Go). Le joueur de la couleur du groupe gagne le jeu de la santé du groupe si le groupe vit et il perd si le groupe meurt. Le concept de jeu permet de faire des calculs associés à une action sur un objet.

Nous rappelons ci-dessous un résumé des symboles utilisés pour décrire l'état dynamique d'un jeu.

> : le jeu est gagné même si l'adversaire commence,
***** : le premier qui joue gagne,
< : le jeu est perdu même si on commence,
0 : le premier qui joue perd (impasse).

Dans ce chapitre, le mot *statique* réfère aux "jugements" d'une position au Go et le mot *dynamique* réfère aux "calculs".

Les niveaux

Le modèle est constitué par des niveaux d'abstraction croissante.

- le niveau **zéro**,
- le niveau **élémentaire**,
- le niveau **itératif**,
- le niveau **global**.

Nous suivons cet ordre pour présenter ces niveaux. Chaque niveau traite des classes particulières d'objets. La figure *Modèle-niveaux* présente les classes d'objets traitées par chaque niveau.

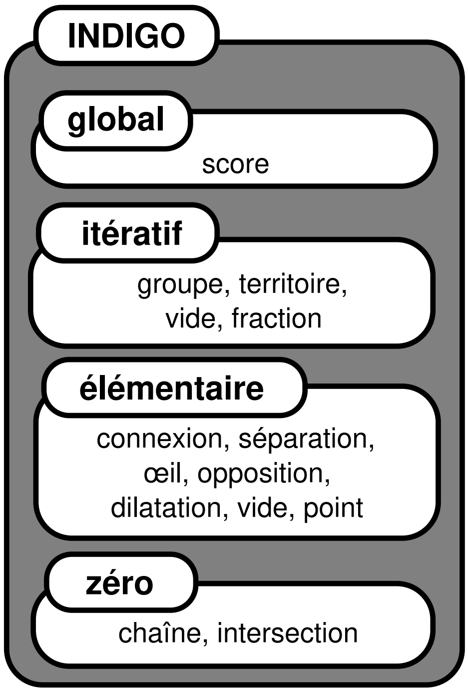


figure Modèle-niveaux

Une confusion qui n'en est pas une :

Au cours de la présentation, un niveau peut être désigné, soit par son nom de niveau, soit par le nom du concept principal contenu dans ce niveau. Par exemple, nous écrirons parfois le *niveau groupe* au lieu d'écrire le *niveau itératif* et parfois nous écrirons le *niveau chaîne* au lieu d'écrire le *niveau zéro*.

L'incrémentalité

Puis, nous présentons la notion fondamentale d'incrémentalité telle qu'elle est traitée dans INDIGO.

Une correspondance entre des concepts présents dans INDIGO et les connaissances du joueur humain

A chaque fois que nous présenterons un concept, implémenté dans le modèle computationnel d'INDIGO, (représenté à ce titre par un rectangle) nous ferons une correspondance avec une connaissance du modèle cognitif du joueur humain, plus ou moins consciente (représentée à ce titre, par un ovale blanc, gris clair ou gris foncé, cf. notations introduites en partie 1). Par exemple, pour exprimer que le concept de regroupement, présent dans le modèle computationnel d'INDIGO, est un concept plutôt non conscient, en partie conscientisable, nous dessinerons la figure *Correspond-exemple*.

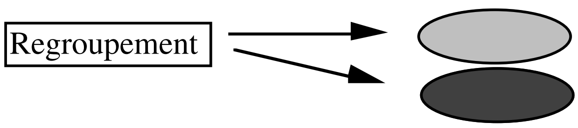


figure Correspond-exemple

Nous donnerons au passage les termes qui décrivent ce concept chez le joueur humain. Par exemple, *connexion* pour le concept humain qui correspond au concept de regroupement dans INDIGO.

Plan de la partie

LE NIVEAU ZÉRO

Présentation

Le proverbe: *"Si vous ne savez pas lire les shichos, ne jouez pas au Go."* a été repris par Mark Boon dans sa thèse [Boon 1991] et adapté à la programmation du jeu de Go : *"Si vous ne savez pas programmer les shichos, ne programmez pas le Go."*

Le niveau zéro a pour objectif de connaître l'état des chaînes et des intersections vides du goban en n'utilisant que des concepts présents dans la règle du jeu : les libertés essentiellement. Ce niveau est composé de deux jeux:

le **jeu de la chaîne**,
le **jeu de l'intersection**.

Un principe de ce niveau est de ne pas utiliser d'information conceptuellement élevée car il doit être utilisé par tous les autres. D'où son nom : le niveau zéro. Le jeu de la chaîne et le jeu de l'intersection ont un point commun : ils utilisent le même générateur de coups et la même analyse statique du jeu dès que l'intersection est occupée par une pierre. Nous avons donc défini une généralisation de ces deux jeux qui s'appelle le **jeu simple**. Nous présentons donc le jeu simple puis le jeu de la chaîne et enfin le jeu de l'intersection. Nous appelons *this* la chaîne instance, *that* les chaînes voisines de *this*, *thathat* les chaînes voisines d'une chaîne *that*.

Le jeu simple

Analyse dynamique

L'analyse dynamique est héritée de celle du jeu en général: alpha-bêta en profondeur d'abord.

Analyse statique

L'analyse statique est spécifique pour le jeu de la chaîne et pour le jeu de l'intersection.

Génération de coups d'attaque :

Si *this* a 1 liberté
    prendre *this*
    Fin

Si *this* a 2 ou 3 libertés
    Si forme
        jouer la forme
    Si une chaîne *that* est atari
        si une chaîne *thathat* est atari
            prendre *thathat*
        sinon,
            si sortie
                sortir *that*
    Si ataris sur *this*,
        faire ataris à *this*

Si *this* a 3 libertés,
    Si une chaîne *that* a 2 libertés
        si une chaîne *thathat* est atari
            prendre *thathat*
        sinon,
            si sortie
                sortir *that*
    Sinon,
        boucher une liberté de *this*

Génération de coups de défense :

Si *this* a une sortie à 4 libertés ou plus
    sortir *this*
    Fin

S'il existe un *that* qui comprend plus de 3 pierres et est atari,
    prendre *that*
    Fin

Pour chaque *that* qui comprend 1 ou 2 pierres et est atari,
    prendre *that*

Si *this* a 1 liberté,
    sortir *this*
    Fin

Si *this* a 2 ou 3 libertés,
    si sortie
        sortir *this*
    si *that* a 2 libertés,
        faire ataris à *that*
    si *that* a 3 libertés,
        boucher une liberté de *that*

Le jeu de la chaîne

Couleur :

Il est de la couleur de la chaîne *this*

Gain :

Si *this* a 4 libertés

Perte :

Si *this* a 0 liberté

Le jeu de l'intersection

Couleur :

La couleur est spécifiée par l'utilisateur du jeu. Si le jeu est spécifié d'une couleur et que l'intersection est finalement contrôlée par cette couleur (resp. par l'autre couleur), le jeu est > (resp. <). Dans le cas ou le jeu est finalement *, la couleur n'est pas utile.

Gain et Perte :

Pour une couleur donnée, il est gagné (respectivement perdu) si l'une des conditions suivantes est remplie :

une chaîne de cette (respectivement de l'autre) couleur est sur l'intersection avec les mêmes conditions de gain que pour le jeu de la chaîne,

l'intersection est vide mais l'adversaire (respectivement on) ne peut y jouer régulièrement.

Jusqu'à combien de libertés aller ?

Nous avons beaucoup hésité pour choisir la complexité de ce niveau. Nous hésitons toujours. Nous voulons un niveau suffisamment simple, pour être utilisé par tous les autres jeux sans biais et pour être rapide. Nous voulons aussi un niveau suffisamment fort pour voir des captures de chaînes qui paraissent évidentes au joueur de Go moyen.

Le concept de liberté de la chaîne *this* est essentiel pour ce niveau. Cependant, plus il y a de libertés pour *this*, moins le critère de liberté est prépondérant devant d'autres critères. La question posée est de savoir si une chaîne est stable si elle a 2, 3, 4 ou 5 libertés ?

Les meilleurs programmes actuels ont ce seuil à 4 (Goliath [Boon 1991]) ou même 5 (Many faces of Go [Fotland 1992] , Star of Poland) libertés selon leur auteur.

Quand le nombre de libertés augmente, le concept de liberté n'est plus un bon critère pour le jeu de la chaîne car il devient de plus en plus coûteux.

2, 3, 4 ou 5 libertés ?

Un niveau, avec un seuil de stabilité égal à 2, est évidemment inacceptable car ce niveau ne verrait pas les shichos.

Un niveau, avec un seuil de stabilité égal à 3, voit les shichos comme sur la figure *Zéro-shicho* :

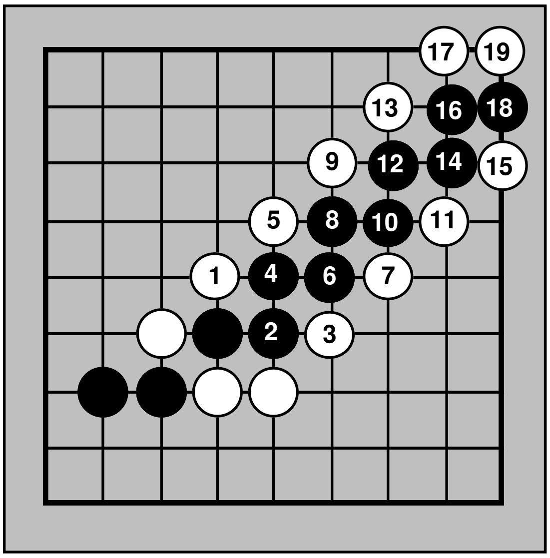


figure Zéro-shicho

Un niveau avec un seuil de stabilité égal à 3 voit aussi la fameuse capture de la pierre sur la deuxième ligne, une des premières choses que les débutants apprennent, comme sur la figure *Zéro-capture-2ème-ligne* :

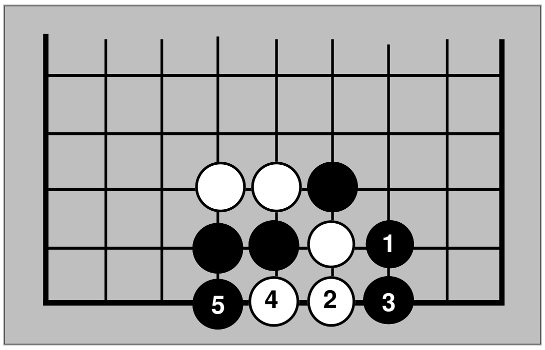


figure Zéro-capture-2ème-ligne

Un niveau avec un seuil de stabilité égal à 3 voit aussi les formes spécifiées par le programmeur au sein de la génération de coups. Par exemple, il peut voir le géta comme sur la figure *Zéro-géta* :

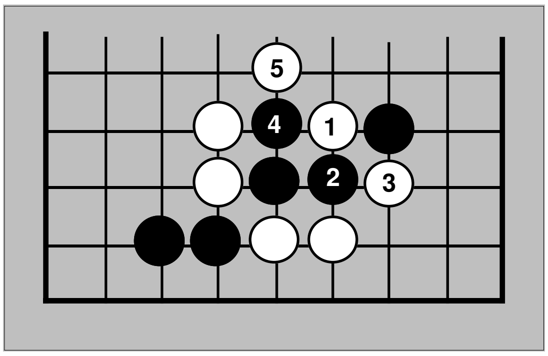


figure Zéro-géta

Un niveau avec un seuil de stabilité égal à 4 voit la capture sur la 3ème ligne comme sur la figure *Zéro-capture-3ème-ligne* :

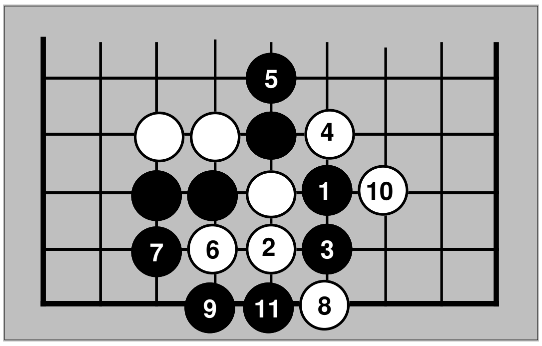


figure Zéro-capture-3ème-ligne

Un jeu de la chaîne avec un seuil de stabilité à 4 libertés voit la capture alors qu'un jeu de la chaîne avec un seuil de stabilité à 3 libertés ne la voit pas. Pour un joueur 10 ème kyu, ce type de capture est presque évident. En plus, la séquence de capture paraît ne pas utiliser d'autre concept que celui de liberté. Ces arguments poussent à avoir un seuil à quatre libertés.

A notre avis, 5 est trop élevé. Pour capturer des chaînes à plus de 4 libertés, il faut utiliser d'autres concepts que celui de liberté. Ce n'est pas une façon humaine de capturer, mais pour des programmes dont l'objectif n'est pas de valider un modèle cognitif mais d'avoir le meilleur rapport performance/coût, c'est peut-être une bonne méthode. Le paradoxe du seuil à 5 libertés est qu'une pierre posée sur une espace vide du goban est instable !

Historique :

L'ancien modèle d'INDIGO avait un jeu de la chaîne à 3 libertés. INDIGO était pratiquement beaucoup plus rapide pour faire une partie de Go complète. Le modèle théorique qui sous-tend INDIGO voyait la capture de chaîne sur la troisième ligne avec les concepts des niveaux supérieurs (groupe, fraction, séparation, encerclement, inimitié, santé, etc.). Pratiquement, INDIGO pouvait aussi voir la capture, mais par chance seulement puisque pratiquement le niveau groupe est statique.

Donc tout dépend de l'objectif que l'on se fixe. En théorie, le seuil à 3 libertés paraît meilleur : un niveau zéro rudimentaire, le niveau sur les groupes faisant le reste. En pratique, la puissance des machines actuelles étant ce qu'elle est, le seuil à 4 libertés est meilleur. Finalement, dans INDIGO nous avons fixé le seuil de stabilité du jeu de la chaîne à quatre libertés.

Conclusion

Le niveau zéro est la base tactique de notre modèle sur le Go. Trouver la complexité adaptée de ce niveau pour qu'il s'intègre efficacement dans le reste du modèle est fondamental. En théorie, un seuil de stabilité des chaînes à 3 libertés paraît correct mais en pratique, un seuil à 4 libertés donne de meilleurs résultats.

<u>Correspondance avec le degré de conscience humain</u>

Nous pensons que le jeu-simple, généralisation du jeu de la chaîne et du jeu de l'intersection, est un jeu conscient, ou au moins conscientisable, chez les joueurs de Go humains. Ce que nous résumons par la figure *Correspondance-jeu-simple*. Pour en parler, ils utilisent des termes comme : shisho, geta, atari, liberté, damezumari.

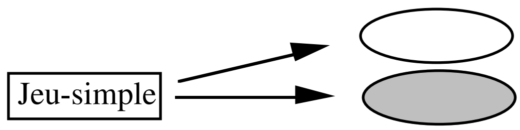


figure Correspondance-jeu-simple

Bibliographie

[Boon 1991] - Mark Boon - Overzicht van de ontwikkeling van een Go spelend programma - Afstudeer scriptie informatica onder begeleiding van prof. J. Bergstra - 1991

[Fotland 1992] - D. Fotland - Many faces of Go, documentation and playing algorithm - 1992

Introduction

Le goban contient des informations confuses a priori. Ces informations sont nombreuses, hétérogènes et ludiques. Pour jouer un coup, le joueur humain classe ces informations suivant différents points de vue.

Le niveau élémentaire doit classer cette information **hétérogène** pour les niveaux supérieurs en respectant plusieurs buts.

Un premier but est d'offrir au niveau supérieur une information **précise**. En effet, cette information est intermédiaire. Elle est utilisée par deux niveaux supérieurs : "groupe" et "global". Une petite imprécision dans ce niveau peut se traduire par de grosses imprécisions dans les niveaux supérieurs et surtout par un très mauvais coup en fin de traitement.

Un deuxième but est de fournir une information **complète**, pour ne pas avoir de trous de comportement dans le programme.

Un troisième but est de produire cette information **rapidement.** En effet, INDIGO doit jouer un coup en moins de 1 minute.

Un quatrième but est d'être constitué de concepts **élémentaires**. Ce but est directement lié à la modélisation dans un domaine complexe. Si le programmeur ne veut pas éternellement refaire les choses, il doit identifier les concepts élémentaires du domaine en les disséquant le plus possible. Il évite ainsi des cercles vicieux de conception et il les fait utiliser indépendamment et facilement par les concepts supérieurs.

Nous avons modélisé chaque point de vue par un jeu au sens de Conway avec un résultat (>, *, <, 0). Le niveau intermédiaire est donc constitué d'une collection de **jeux**.

Nous présentons ce niveau en trois parties. D'abord nous décrivons le langage de **règles**. qui conseillent des coups dans les jeux. Ensuite, nous donnons la description des jeux en **l'état actuel** du niveau intermédiaire dans le système INDIGO. Au delà de cette liste de jeux, il nous a paru intéressant de présenter les difficultés rencontrées pour aboutir à l'état actuel. Nous présentons donc la **méthode, spécifique à ce niveau, utilisée** pour arriver à identifier certains jeux cruciaux et les formaliser.

Le plan de ce niveau est le suivant:

Le langage d'expression des règles

La caractéristique principale des règles d'un programme de Go est de **contenir des dessins dans les prémisses**. En effet, il est inimaginable de transmettre de façon procédurale et linéaire des connaissances visuelles à une machine. Des règles avec des patterns à deux dimensions en partie gauche permettent de transmettre plus aisément des connaissances visuelles à un programme de Go.

<u>Exemple de règle communiquée à la machine</u>

```
PIERRE
# # #  ##
# # ○ # #
# ● + ● #
# # ○ # #
# # #  # #

BORD
# # # ##
# # # ##
# # # ##
# # # ##
# # # ##

COUPS NOIRS
00000
00000
00100
00000
00000

COUPS BLANCS
00000
00000
00100
00000
00000

CONNEXION = *
00000
00000
00100
00000
00000

SEPARATION = *
00000
00000
00100
00000
00000
```

Cette règle dit que si le dessin du haut est reconnu et que l'on n'est pas près du bord, il est conseillé de jouer au centre, pour Noir et pour Blanc, pour connecter et pour séparer l'adversaire. Elle dit aussi que les jeux de la connexion et de la séparation de l'adversaire sont dans l'état dynamique *. Cela sera décrit plus loin.

La grammaire

Les règles sont exprimées dans le langage dont la grammaire est la suivante[1].

règle : partie-gauche partie-droite
partie-gauche : pattern-pierre pattern-bord (condition)*
condition : LIBERTE = nombre
partie-droite : (attribut)*
attribut : conseil / point-de-vue
conseil : coups-blancs / coups-noirs
point-de-vue : nom = état pattern-bit
état : état-dynamique / état-statique
coups-blancs : COUPS NOIRS pattern-bit
coups-noirs : COUPS BLANCS pattern-bit
pattern-pierre : PIERRE (lettre-pierre)25*
lettre-pierre : ● / ○ / + / A / M / W / #
pattern-bord : BORD (lettre-bord)25*
lettre-bord : B / A / C / K / #
pattern-bit : (bit)25*
bit : 0 / 1
nom : OEIl / CONNEXION / SEPARATION / OPPOSITION / DILATE / CHAINE / ZONE / POINT / VIDE / FUSEKI
état-statique : GAGNE / PERDU / AUTRE
état-dynamique : > / < / * / 0

Des commentaires sur la grammaire

'**partie-gauche**' est spécifique du jeu de Go. Deux des conditions qui la constituent sont picturales. L'une contient une disposition de pierres ('pattern-pierre'), l'autre contient une disposition par rapport au bord du goban ('pattern-bord').

Les autres conditions '**condition**' sont rudimentaires et ne servent que pour le jeu de la chaîne et de l'intersection; ces conditions portent sur le nombre de libertés de la chaîne de la case centrale du pattern.

'**partie-droite**' est une liste d'attributs.

Un '**attribut**' peut être statique ou dynamique. Un attribut statique permet d'interpréter une petite partie du goban sous un point de vue particulier et de conseiller des coups sous ce point de vue. Par exemple, du point de vue de la connexion, une règle peut spécifier qu'une connexion est possible et conseiller un coup. Un attribut dynamique permet également d'interpréter une petite partie du goban sous un point de vue particulier mais l'information qu'il contient est le résultat d'un jeu. Par exemple, une règle peut spécifier qu'une connexion est * et conseiller un coup de connexion sûre.

[1] / ( ) * sont des métasymboles au sens usuel.

Les **patterns** sont entendus au sens visuel et pictural du terme. Les patterns sont des fenêtres ou dessins 5 par 5 qui représentent la disposition des pierres, du bord ou des coups à jouer sur le goban. Au départ *nous avons défini le langage décrit ici pour représenter l'information visuelle du Go*. Il est impensable de représenter l'équivalent des connaissances visuelles sous forme de procédures qui seraient associées à chaque pattern et de maintenir la connaissance visuelle sous forme procédurale. Pour le programmeur, il est indispensable de communiquer facilement avec le programme et des dessins sont très utiles. La taille maximale des dessins est 5 par 5 car beaucoup de formes tactiques du Go y tiennent. L'avantage de limiter à 5-5 est lié à l'implémentation en C++ du programme : 25 est inférieur à 32. Ainsi, un entier 32 bits peut représenter un pattern en machine. Le pattern-matching utilise alors largement l'opérateur & pour comparer la position réelle du goban avec le pattern [Boon 1989]. Pour couvrir une grande partie des formes utilisées fréquemment au Go et faire un premier programme de Go, une fenêtre 5-5 suffit. Pour accélérer le pattern-matching, on peut hacher les formes. L'intérêt dépend du nombre de règles dans la base. En ce qui nous concerne, nous avons séparé les bases de règles en sous bases qui ne contiennent pas suffisamment de règles pour que le hachage soit nécessaire.

'**condition**' est une condition assez rudimentaire qui est généralement liée au nombre de libertés des pierres. Cette partie des prémisses d'une règle n'a pas été spécialement approfondie. Pour améliorer le langage il serait bon de généraliser les conditions avec l'effort de programmation associé.

'**lettre-pierre**' peut valoir Noir (représenté dans notre thèse par ●) , Blanc (○), Vide (+), ¬Noir (M), ¬Blanc (A), ¬Vide (W) ou #[1].

'**lettre-bord**' peut valoir Bord (B) , ¬Bord (A), Coin (C), ¬Coin (K) ou #.

'**digit**' peut valoir 0 ou 1[2].

'**état**' peut avoir une valeur ludique dynamique (>, <, *, 0) ou statique (gagné, perdu, autre) selon que la règle est dynamique ou statique.

[1]La taille des patterns est fixée, donc certaines intersections n'ont pas d'importance. Elles sont indiquées par un # dans le pattern. On aurait pu imaginer d'omettre les # puisqu'ils sont indifférents.
[2]Même remarque pour le 0 que pour le #.

Convention

Nous avons fait une **convention** implicite dans l'écriture de règles :

Noir est *ami*.

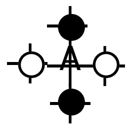
Blanc est *ennemi*.

Les attributs de la partie droite des règles donnent des points de vue amis (de Noir).

Présentation d'une règle pour le lecteur

Pour alléger la présentation au lecteur, nous lui présentons les règles sous la forme de l'exemple de la figure *Elem-règle-lecteur*.

SI



ALORS

Conseil pour Noir: A
Conseil pour Blanc: A
Connexion * en A
Séparation * en A

figure Elem-règle-lecteur

Lorsque le point de vue utilisateur de la règle ne sera pas ambigu, et pour alléger la présentation, nous ne mettrons que le pattern de la partie gauche de la règle comme sur la figure *Elem-règle-lecteur-allégée*. Le A indique le coup conseillé par la règle. Pour simplifier la présentation, nous supposons que les coups amis ou ennemis sont conseillés au même endroit (en A). Ce qui est le cas dans 90% des cas. Dans INDIGO, la distinction est faite.

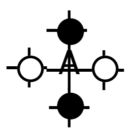


figure Elem-règle-lecteur-allégée

Les jeux du niveau élémentaire

Nous avons identifié et formalisé 8 jeux au niveau élémentaire. Pour chaque jeu, nous présentons intuitivement le jeu au lecteur, ensuite nous donnons la définition des règles du jeu (gain, perte), puis nous donnons les règles engendrées par le joueur, puis nous décrivons l'aspect dynamique du jeu, enfin nous donnons les perspectives envisageables.

Le jeu de la connexion

Reconnaissance

La règle du jeu de Go utilise la connexion pour définir les chaînes. Le joueur de Go utilise une notion de connexion pour reconnaître les groupes de pierres, base de son raisonnement. Ainsi, bien reconnaître les connexions permet de bien reconnaître les groupes et de partir sur des bases saines pour raisonner.

Pour qu'un jeu de la connexion soit reconnu, il faut deux pierres amies dans une même fenêtre. Il peut y avoir aussi des pierres ennemies. La couleur des pierres amies définit la couleur du jeu.

Les règles du jeu

Conditions terminales

gain :

Spécifié par une règle.

perte :

Spécifié par une règle.

Règles pour jouer des coups

Le jeu de la connexion engendre les coups conseillés par des règles. La figure *Elem-connexion-statique* montre des exemples de règles. Les A symbolisent les coups engendrés par le jeu de la connexion.

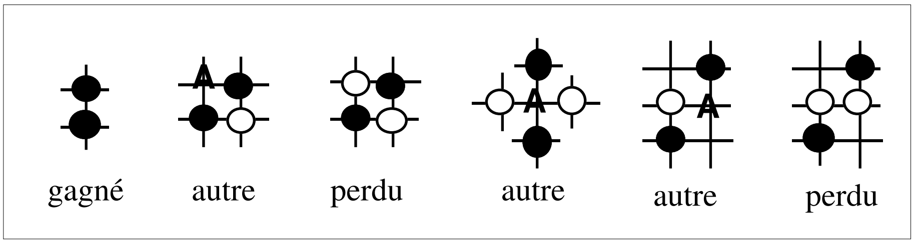


figure Elem-connexion-statique

L'aspect dynamique

En "calculant", INDIGO détermine l'état dynamique des jeux de la connexion. La figure *Elem-connexion-dynamique* montre des résultats de jeu de la connexion .

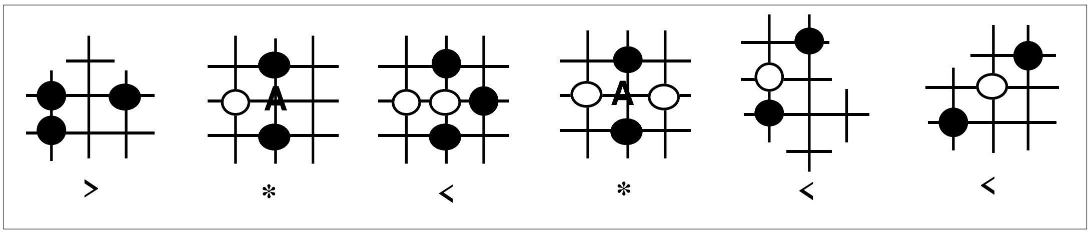


figure Elem-connexion-dynamique

Rappel :

Le résultat d'un jeu est > si l'action associée est atteinte, que Noir ou Blanc commence.
Il est < si l'action associée n'est pas atteinte, que Noir ou Blanc commence.
Il est * si l'action associée est atteinte si l'ami (ici Noir) commence, et non atteinte si l'ennemi (ici Blanc) commence. Si le jeu est *, A dèsigne le coup conseillé.

Explication par calcul :

La figure *Elem-connexion-calcul* explique comment les formes de la figure *Elem-connexion-dynamique* sont été classées. La première ligne suppose que Noir commence et la deuxième ligne suppose que Blanc commence. Les "oui" indiquent que le but (la connexion) est atteint. Les "non", que le but n'est pas atteint.

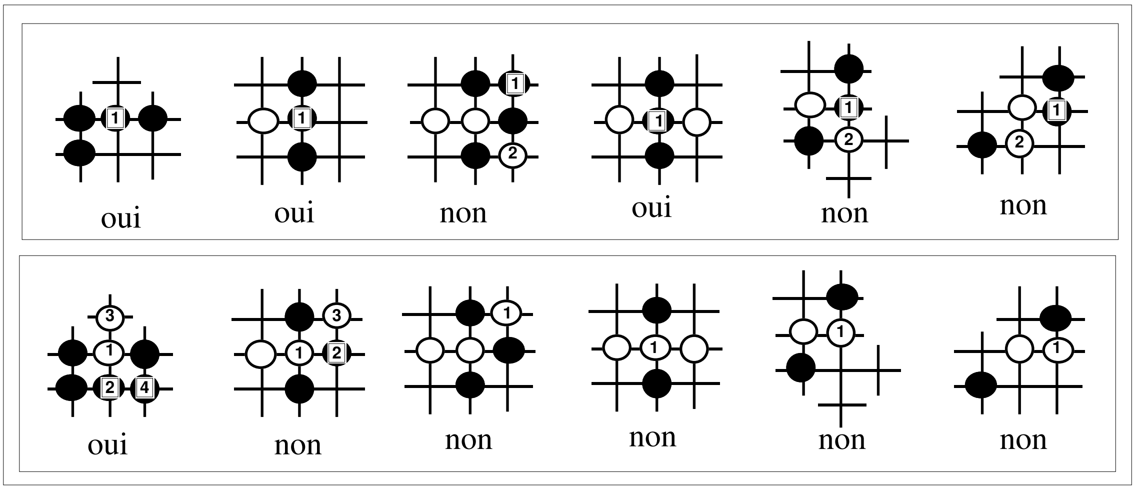


figure Elem-connexion-calcul

Autres informations importantes

Actuellement, la base contient environ 200 règles de connexion. Ce nombre est assez grand car l'état d'une connexion dépend non seulement de la disposition des pierres mais aussi de sa position par rapport au bord. Si 60 formes de connexion couvrent la plupart des cas fréquents dans une partie complète, il en faut environ deux fois plus pour décrire ce qui se passe près du bord.

50 règles sont statiques et 150 règles sont dynamiques. En effet, la taille maximale, 5-5, des formes permet dans la plupart des cas d'y inclure le résultat d'un jeu de la connexion. En cours de partie, INDIGO fait très peu de calculs de connexion, il utilise seulement les résultats des règles dynamiques de connexion.

Le jeu de la connexion utilise le jeu de l'intersection. Si une intersection vide est conseillée par une règle, le coup n'est réellement engendré que si le jeu de l'intersection est *. Si le jeu de l'intersection est de la couleur de la connexion, le jeu de la connexion est >, sinon le jeu de la connexion est <.

Notre base a été constituée à la main régulièrement en fonction des besoins du programme. Son point faible actuel est la difficulté de mise à jour. Pour l'augmenter ou la valider, il serait nécessaire d'automatiser sa constitution en fonction de la distance des deux pierres amies, la disposition des pierres ennemies, la position du bord. Pour connaître la valeur ludique des attributs de la partie droite de la règle, il faut écrire un programme qui calcule les jeux de la connexion pour toutes ces formes.

Perspectives envisageables

Il est possible d'améliorer le jeu de la connexion en intégrant la possibilité de capturer une chaîne ennemie, avec le jeu de la chaîne, pour se connecter.

Le jeu de la séparation de l'adversaire

Reconnaissance

La règle du jeu de Go n'utilise pas explicitement la séparation des chaînes par l'adversaire. Pourtant, le joueur de Go utilise une notion de séparation pour séparer l'adversaire.

Comme pour le jeu de la connexion, le jeu de la séparation est reconnu lorsque deux pierres amies sont dans une même fenêtre. Il peut y avoir aussi des pierres ennemies. La couleur des pierres amies définit la couleur du jeu.

Les règles du jeu

Conditions terminales

gain :

Spécifié par une règle.

perte :

Spécifié par une règle.

Règles pour jouer des coups

Les remarques faites sur le jeu de la connexion restent valables pour le jeu de la séparation de l'adversaire en deux. La figure *Elem-séparation-statique* donne des exemples de formes avec leur état statique.

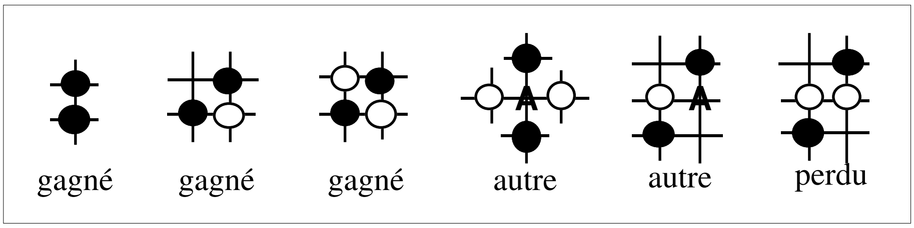


figure Elem-séparation-statique

L'aspect dynamique

En calculant, le système INDIGO peut dire quel est l'état dynamique des jeux de la séparation de la figure *Elem-séparation-dynamique*.

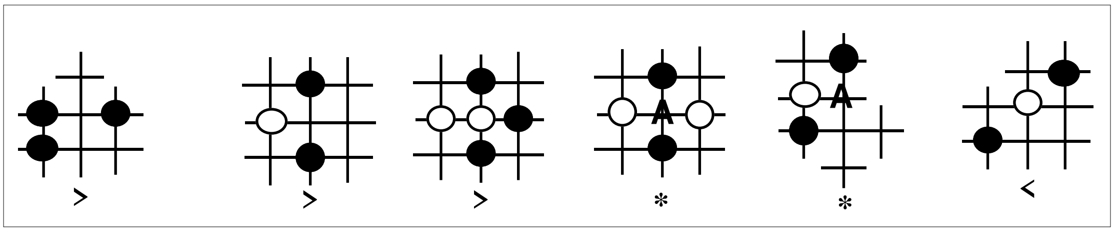


figure Elem-séparation-dynamique

Explication par calcul :

La figure *Elem-séparation-calcul* explique par le calcul comment sont classées les formes de la figure *Elem-séparation-dynamique* .

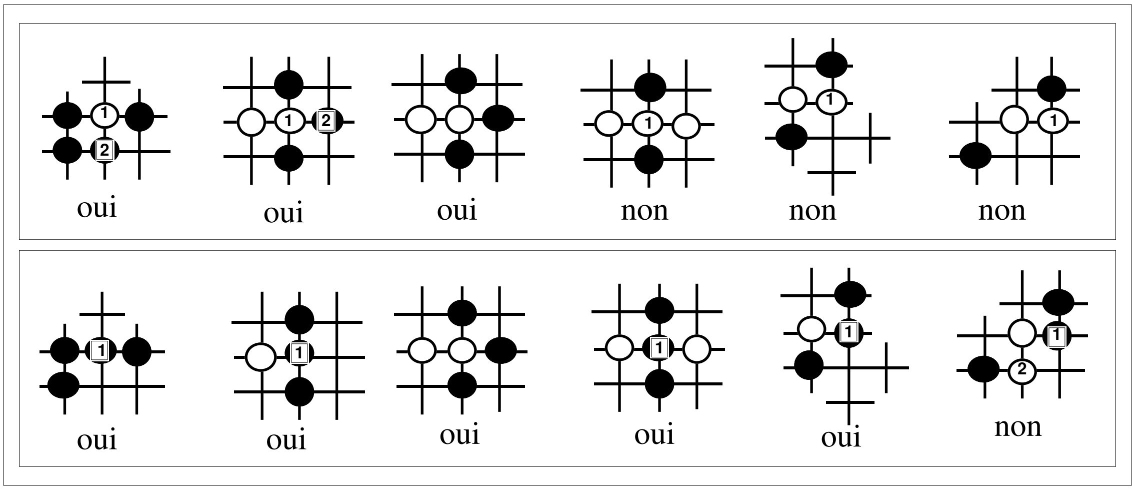


figure Elem-séparation-calcul

Autres informations importantes

Le jeu de la séparation et le jeu de la connexion utilisent les mêmes règles. En effet l'ensemble des parties gauches des règles de séparation et l'ensemble des parties gauches des règles de connexion décrites ci-dessus sont égaux. Ainsi, il a été plus pratique de fusionner les règles de séparation avec les règles de connexion. Il est donc finalement important de préciser que INDIGO contient 200 règles "topologiques".

Les informations importantes du jeu de la connexion citées précédemment restent donc valables pour le jeu de la séparation.

Dans le paragraphe sur la méthode utilisée pour détecter le concept de séparation, nous préciserons comment la séparation et la connexion sont liées mais différentes.

Le jeu du point

Aperçu

Le jeu du point est un concept complexe. Les joueurs de Go ne l'utilisent pas explicitement. Nous avons mis du temps à l'expliciter. Il est donc nécessaire de commencer par un aperçu de ce concept.

Premier aperçu

Sur la figure *Elem-point-1* , les deux intersections vides ont l'apparence de deux points de territoire. En effet le jeu de l'intersection est noir sur ces deux intersections (si Blanc joue dedans il est capturé au coup suivant).

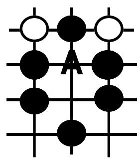


figure Elem-point-1

Par contre, on peut se demander si ces intersections contrôlées par Noir, seront des points de territoires à la fin de la partie.

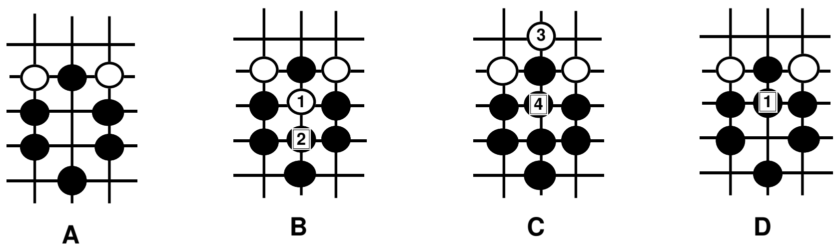


figure Elem-point-2

Sur la position A de la figure *Elem-point-2* , les deux intersections centrales ont donc l'apparence d'un territoire. (Le jeu de l'intersection sur ces deux intersections est Noir). Par contre sur la position B, 1 menace de s'installer et 2 contrôle en capturant 1. Sur la position C, 3 oblige 4. Les deux intersections centrales sont remplies et ne sont plus des points de territoire. Donc, si Blanc joue le premier, les deux intersections ne sont pas des points de territoire.

Si Noir joue le premier, comme sur la position D de la figure *Elem-point-2*, un point de territoire subsiste.

Le jeu du point sert à savoir quelles intersections, contrôlées par une couleur au sens du jeu de l'intersection, sont vraiment des points de territoire.

Plus généralement, sur la position A de la figure *Elem-point-3* , on peut se demander quelles intersections sont des vrais points de territoire sachant que les intersections marquées par des carrés ■ sur la position B ont un jeu de l'intersection Noir.

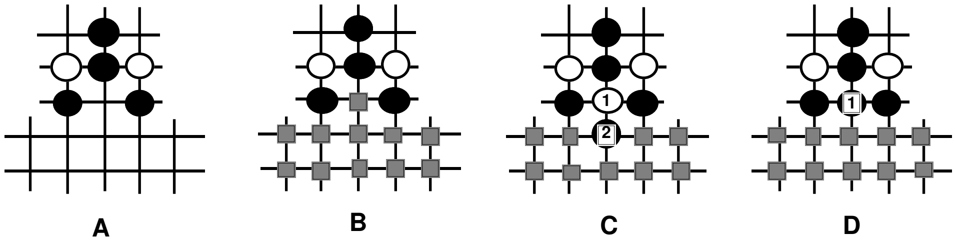


figure Elem-point-3

Si Blanc joue le premier, les points de territoire sont indiqués par les ■ de la figure C. Si Noir joue le premier, les points de territoire sont indiqués par les ■ de la figure D.

De même, sur la position A de la figure *Elem-point-4* , on peut se demander quelles intersections sont des points de territoire sachant que les intersections marquées par des carrés ■ sur la position B ont un jeu de l'intersection Noir.

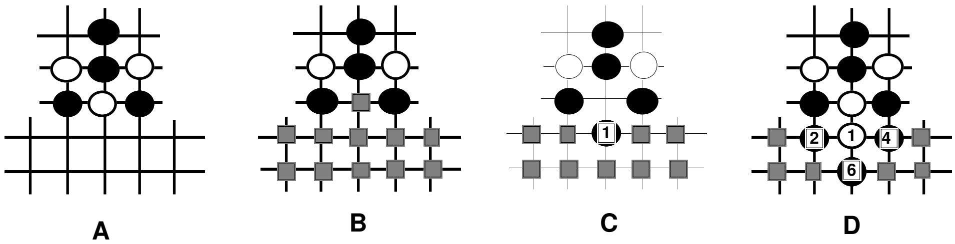


figure Elem-point-4

Si Noir joue le premier en capturant la pierre blanche, les points de territoire sont indiqués par les ■ de la figure C.

Si Blanc joue le premier en sortant sa pierre blanche, Noir devra jouer 3 coups sur les 3 libertés de la chaîne blanche pour la capturer. On retrouvera alors la position de la figure *Elem-point-1* où l'on sait qu'il n'y a aucun point de territoire. Donc, si Blanc joue le premier, les points de territoire de la position A sont indiqués par les ■ de la position D.

Reconnaissance

Pour reconnaître la présence d'un jeu du point, il faut avoir identifié un territoire au sens du jeu de l'intersection et un pattern conceptuel de la forme de la figure *Elem-règle-point-LFD* .

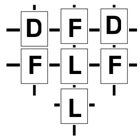


figure Elem-point-LFD

Les intersections marquées F doivent être occupées par un joueur donné. La couleur de ce joueur détermine la couleur du jeu.

Les intersections marquées L doivent être contrôlées par ce joueur.

Les intersections marquées D doivent être contrôlées par l'adversaire ou * pour le jeu de l'intersection.

Les règles du jeu

Conditions terminales

Le jeu se termine lorsqu'aucun coup ne peut plus être engendré. Dans ces positions, le résultat n'est pas ludique (gagné, perdu) mais le nombre de points du territoire reconnu.

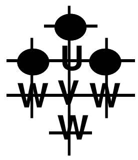


figure Elem-point-UVW

Les intersections cruciales pour savoir si un point existe sont dans l'ordre U, V, W de la figure *Elem-point-UVW* .

N.B. Dans le cas de l'œil de taille 1, V et W n'ont pas d'importance, seule U est importante. Dans le cas de l'œil de taille 2, W n'a pas d'importance, seules U et V ont de l'importance. Dans le cas d'un territoire de taille assez grande, U, V et W ont de l'importance.

Règles pour jouer des coups

Les coups sont engendrés par des règles du type de celle de la figure *Elem-point-règle*. Les A indiquent les coups à engendrer.

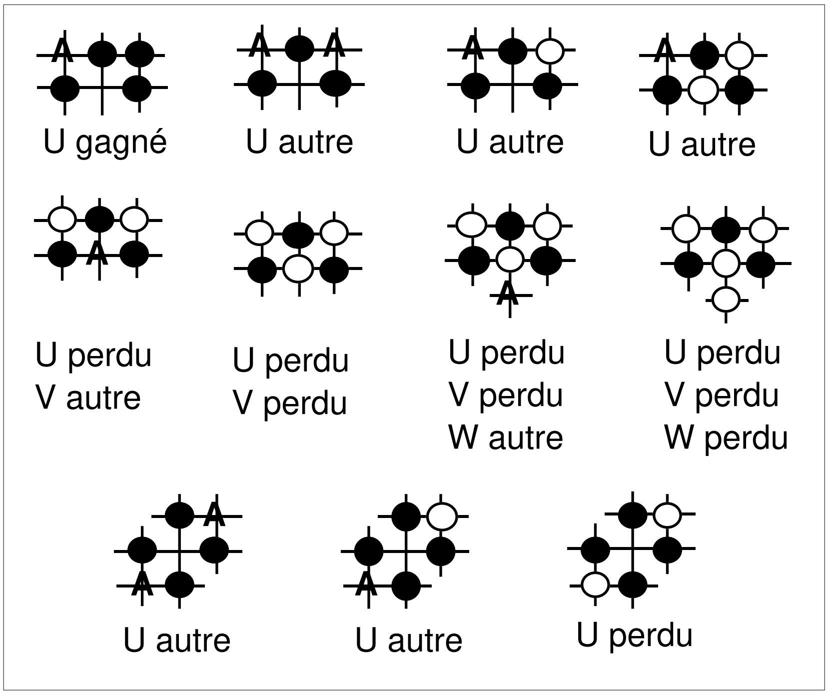


figure Elem-point règle

Les règles spécifient si les intersections de type U, V ou W sont des points 'gagné', 'perdu' ou 'autre'.

L'aspect dynamique

L'aspect dynamique a été décrit dans le premier aperçu. La taille de l'arbre est minime, environ 4 coups de profondeur avec 1 ou 2 coups possibles à chaque nœud. En fait, le calcul sur le jeu du point n'est jamais déclenché tel quel. C'est toujours le jeu de l'œil ou le jeu de la séparation du territoire en 2 qui "appellent" le jeu du point. Plus exactement, le jeu de l'œil ou le jeu de la séparation du territoire en 2, utilisent les règles point pour une première recherche sélective.

Autres informations importantes

Le jeu du point est important car il est **utilisé par le jeu de l'oeil et le jeu de la séparation du territoire en deux**. Le jeu du point contient 40 règles statiques.

Perspectives envisageables

Nous ne pensons pas que l'on puisse encore découper le jeu du point en sous-jeux. Historiquement, le jeu du point a été explicité par les travaux sur les deux jeux utilisateurs actuels : le jeu de l'œil et le jeu de la séparation du territoire en deux. Si une perspective existe, ce sera la fusion du jeu de l'œil et du jeu de la séparation du territoire en deux en un seul jeu qui retournera le "nombre" d'yeux d'un territoire (0, 1, 2, 3, ..., { 0 | 1 }, { 1 | 2 }, { 2 | 3 }).

Le jeu de l'œil

Reconnaissance

L'importance du concept d'œil résulte du proverbe suivant :

Si un groupe a deux yeux il est vivant.

Sur un goban, un joueur de Go reconnaît un œil à sa forme. Les figures *Elem-œil* et *Elem-œil-2* donnent des exemples d'yeux à calculer.

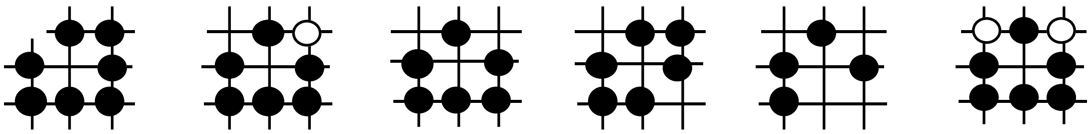


figure Elem-règle-œil

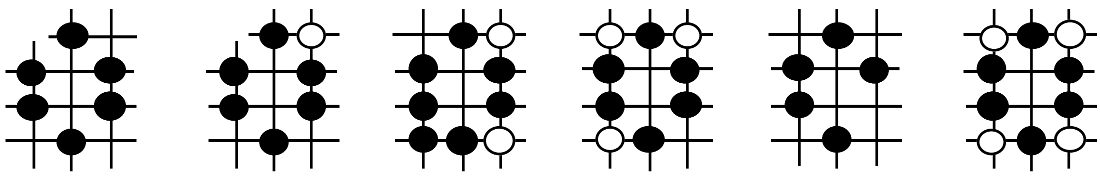


figure Elem-règle-œil-2

Pour déclencher un calcul sur le jeu de l'œil, un territoire de taille 1 ou 2 doit être reconnu, au sens du jeu de l'intersection.

Les règles du jeu

Conditions terminales

gain :

Pas de coup possible. 1 ou 2 points de territoire.

perte :

Pas de coup possible. 0 point de territoire.

Règles pour jouer des coups

Les coups sont engendrés par les règles du jeu du **point** et par les **ataris** sur les chaînes amies constituantes de l'œil.

L'aspect dynamique

En calculant, le système INDIGO peut dire quel est l'état dynamique des jeux de l'œil de la figure *Elem-œil-résultat*.

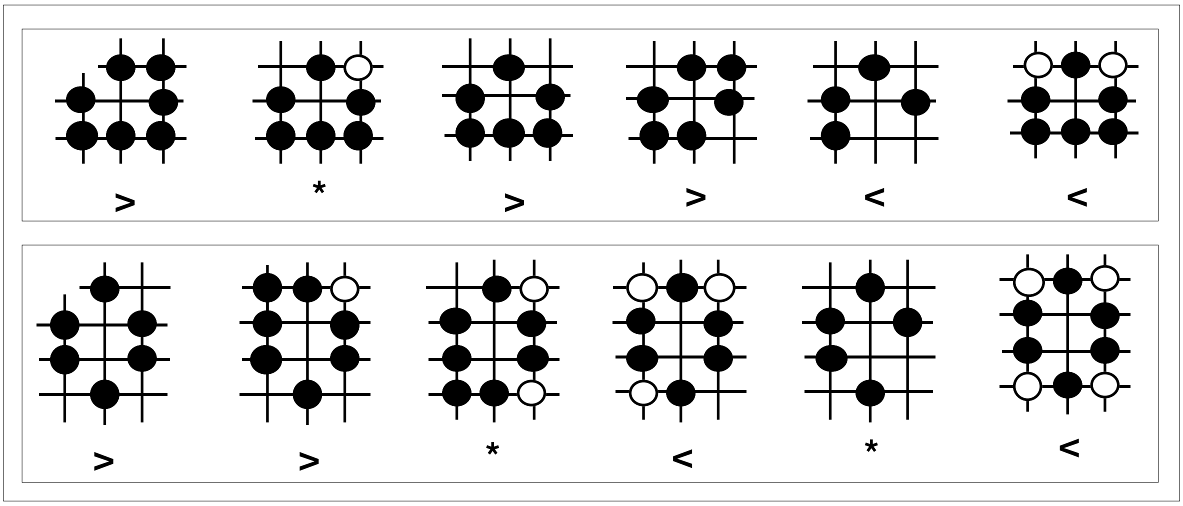


figure Elem-œil-résultat

Nous avons supposé que les coups d'atari sur chaîne frontière sont engendrés.

Explication par calcul :

La figure *Elem-œil-calcul* donne les séquences de coups qui expliquent les résultats du jeu de l'œil donnés par la figure *Elem-œil-résultat*.

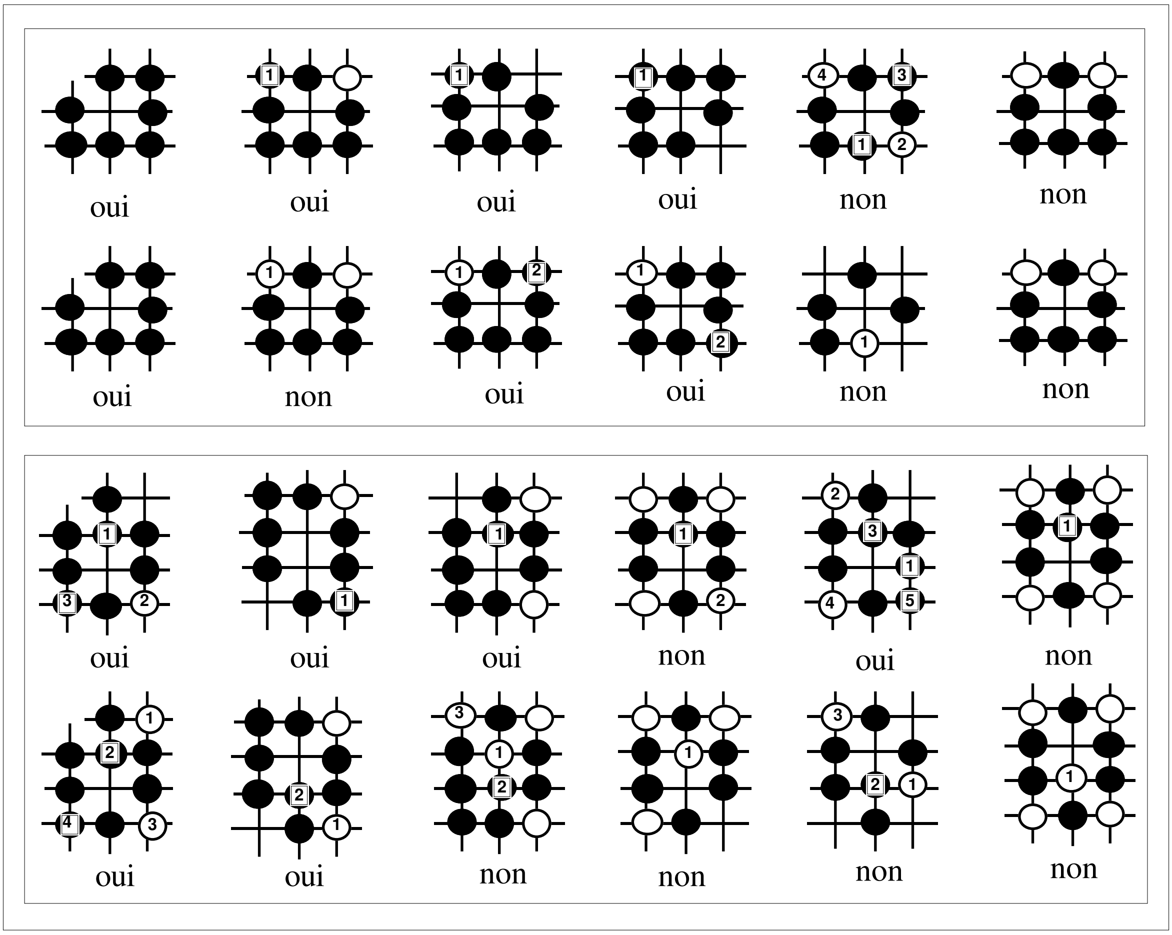


figure Elem-œil-calcul

Autres informations importantes

Le jeu de l'œil présenté ci-dessus est le fruit d'une étude poussée sur le jeu de l'œil et le jeu de la séparation du territoire en 2 réunis. Au départ nous avions modélisé le jeu de l'œil autrement. Nous présentons cette autre modélisation dans le paragraphe sur la méthode utilisée pour identifier et formaliser les jeux du niveau intermédiaire.

Le jeu de la séparation du territoire en 2

Reconnaissance

Pour déclencher un calcul sur le jeu de la séparation du territoire en 2, un territoire de taille 3 ou plus doit être reconnu, au sens du jeu de l'intersection.

Les règles du jeu

Conditions terminales

gain :

2 territoires distincts avec au moins un point de territoire et les chaînes frontières positives.

perte :

1 chaîne frontière du territoire capturée physiquement ou 1 seul territoire avec des règles "poison" négatives.

Règles pour jouer des coups

Les règles pour jouer des coups sont les **ataris** sur les chaînes frontières du territoire, les coups engendrés par les règles du **point**, les règles "**chaîne-poison**", les règles "**fraction-poison**".

Les règles chaîne-poison

Les règles de type chaîne-poison sont des règles statiques utilisées par le jeu de la séparation du territoire en 2. Les formes de chaîne-poisons sont des formes constituées par une chaîne maximale. Elles permettent de discriminer si une chaîne est un poison pour l'adversaire : si l'adversaire capture cette chaîne, on peut l'empêcher d'y faire deux yeux. La figure *Elem-pattern-chaîne-poison* montre des exemples de patterns de chaîne-poison. Rappelons que M signifie ¬Noir.

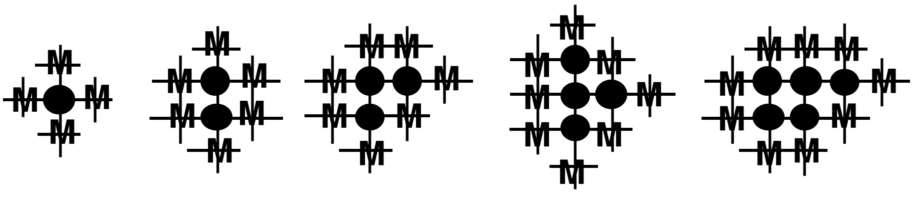


figure Elem-pattern-chaîne-poison

Selon le type du pattern, la partie droite de la règle spécifie si la forme est "morte"[1] et s'il est possible de jouer un coup qui fait grossir cette chaîne en la laissant "morte". La figure *Elem-règle-chaîne-poison* montre les coups possibles A sur les exemples précédents. Les coups autres que A sont mauvais car la chaîne ne sera plus un poison.

[1]Si l'adversaire capture physiquement une chaîne morte il n'obtiendra que 0 ou 1 œil.

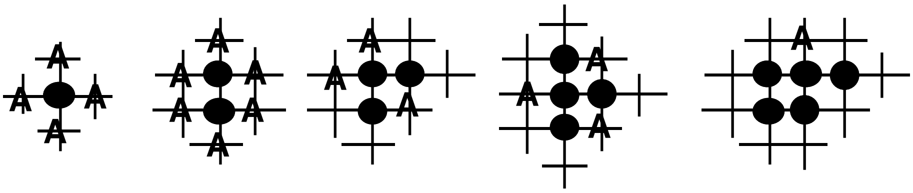


figure Elem-règle-chaîne-poison

Actuellement, la base contient 10 règles de chaîne-poison. Elle a été constituée à la main. Elle est facile à maintenir.

les règles fraction-poison

Les règles de type fraction-poison sont des règles statiques utilisées par le jeu de la séparation du territoire en 2. Les formes de fraction-poison sont reconnues sur un territoire obtenu par morphologie mathématique. Elles permettent de discriminer si une fraction est un poison pour l'adversaire, au sens où si l'adversaire donne cette forme à son territoire, il ne pourra pas le partager en deux. La figure *Elem-règle-fraction-poison* montre des exemples de règles "fraction-poison".

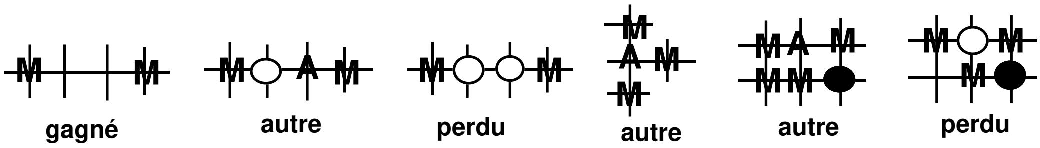


figure Elem-règle-fraction-poison

M signifie ¬Noir et A le coup conseillé.

Actuellement, la base contient 50 règles de fraction-poison. Elle a été constituée à la main. Elle n'est plus très facile à maintenir. Il serait peut-être bon de pouvoir la créer automatiquement par programme.

L'aspect dynamique jeu de la séparation du territoire en 2

Il fonctionne sur le même principe que le jeu de l'œil, c'est-à-dire en utilisant le jeu du point. Une première recherche sélective consiste à ne jouer que les coups conseillés par les ataris, par les règles "point". Ainsi le territoire réel se réduit. Quand les coups engendrés par le jeu du point sont épuisés, une deuxième recherche sélective commence avec les coups engendrés par les règles "chaîne-poison" et "fraction-poison".

Sur la figure *Elem-JST2-dynamique* , nous donnons trois positions avec le résultat du jeu de la séparation du territoire en 2.

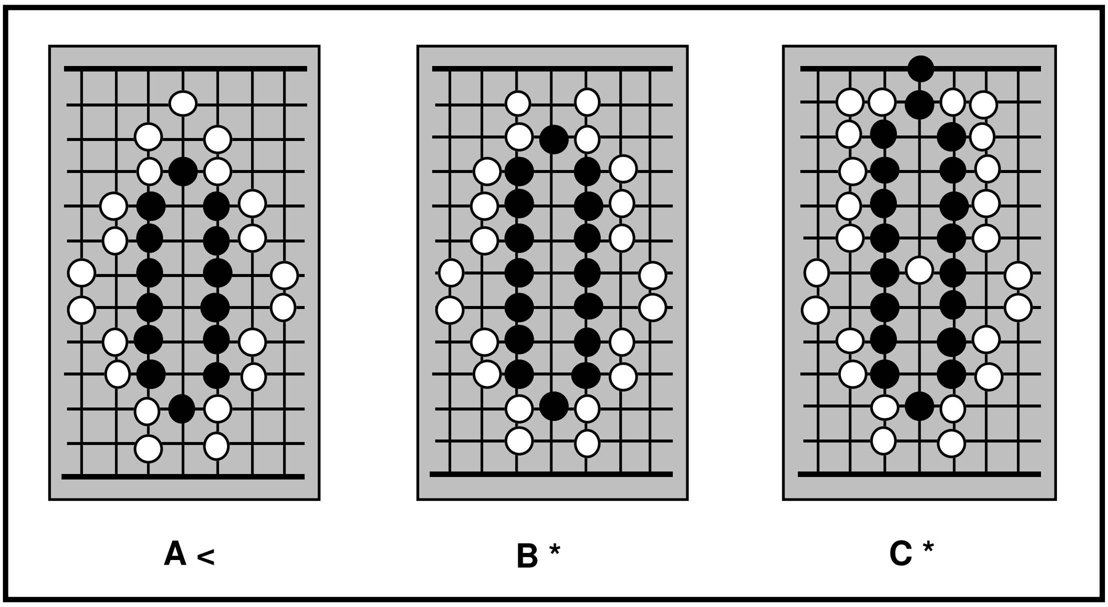


figure Elem-JST2-dynamique

Explication par calcul :

La figure *Elem-JST2-calcul* donne une variante des calculs qui expliquent ces résultats pour chaque exemple et en supposant que Noir commence puis que Blanc commence.

Les séquences de coups contiennent des coups engendrés par les bases de règles "point" ou "poison" comme cela est précisé ci-dessous.

Position A:
- Si Noir commence,
  - 1, 2, 3 sont engendrés par "point".
  - 4 est engendré par "poison".
- Si Blanc commence,
  - 1, 2, 3, 4 sont engendrés par "point".

Position B:
- Si Noir commence,
  - 1, 2, 3 sont engendrés par "point".
  - 4, 5 sont engendrés par "poison".
- Si Blanc commence,
  - 1, 2, 3, 4 sont engendrés par "point".
  - 5 est engendré par "poison".

Position C:
- Si Noir commence,
  - 1, 2, 3 sont engendrés par "point".
  - 4, 5 sont engendrés par "poison".
- Si Blanc commence,
  - 1, 2, 3 sont engendrés par "point".
  - 4, 5 sont engendrés par "poison".

En ce sens, le jeu de la séparation du territoire est conçu sur le modèle de l'approche 1 du paragraphe Méta du paragraphe Domaines Voisins.

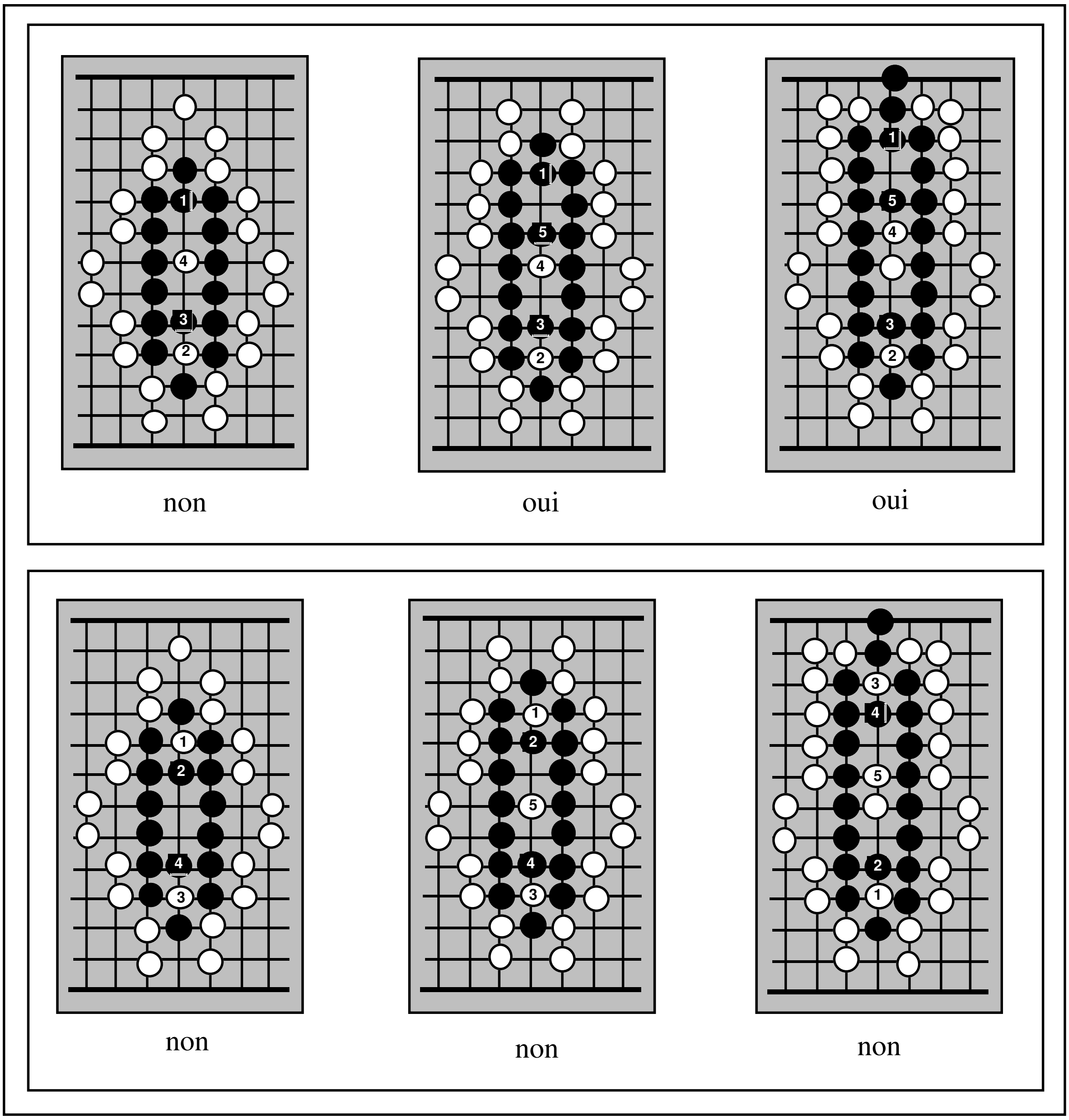


figure Elem-JST2-calcul

Autres informations importantes

La base de règles a été découplée.

Une réussite des règles "poison" est d'avoir été **découplée en deux bases de règles distinctes** : "chaîne-poison" d'une part et "fraction-poison" d'autre part. Avant le découplage de "chaîne-poison" et "fraction-poison", nous avions beaucoup de règles (1000 = 10 * 50 * 2 = nombre-actuel-règles-chaîne-poison * nombre-actuel-règles-fraction-poison * prise-en-compte-du-bord). Maintenant, nous avons 50 + 10 = 60 règles. La différence est sensible.

Un autre avantage est d'avoir rendu cette **base indépendante du bord** du goban. En effet les règles "fraction-poison" ont implicitement des frontières floues ou solides, c'est-à-dire avec présence ou non de pierres. Il aurait été maladroit de vouloir engendrer explicitement ces formes.

<u>La recherche sélective s'effectue en deux temps.</u>

Le fait de commencer la recherche sur les coups engendrés par les règles "point" puis par les règles "poison" limite la recherche et approche le résultat trouvé. Nous avons envisagé de jouer les coups "poison" (engendrés seulement lorsque le territoire est localisé) plus tôt. Le problème est d'engendrer ces coups avant d'avoir localisé le territoire. En fait, après une recherche de ce type les intersections où des coups "poison" ont été engendrés sont connues. On peut donc recommencer la recherche en engendrant ces coups dès le début.

<u>D'autres systèmes :</u>

Le lecteur remarquera que les exemples donnés sont étiquetés "corridor" par Berlekamp [Berlekamp & Wolfe 1994]. Le problème résolu ici est différent car Berlekamp "compte" le nombre de points du territoire alors que le jeu présenté ici "compte" le nombre d'yeux.

Le programme RISIKO de Thomas Wolf [Wolf 1992] [Wolf 1993] est particulièrement performant sur ce type de problème. Il trouve la solution rapidement. Mais ce système est spécialisé sur ce type de problèmes (appelé problèmes de "Vie et mort").

Perspectives envisageables

Peut-être sera-t-il possible de généraliser encore le jeu de l'oeil et le jeu de la séparation en 2, en un seul jeu, qui serait le jeu du territoire. Les valeurs du résultat de ce jeu seraient alors des nombres qui indiqueraient le nombre d'yeux possibles pour les deux joueurs au lieu de spécifier le résultat avec >, <, *.

Le jeu de l'opposition

Une image qui fait comprendre ce qu'est le jeu de l'opposition est celle de deux plantes ennemies (une ronce et du chiendent par exemple) qui se développeraient l'une à côté de l'autre de façon à étouffer l'autre. Ainsi, dés qu'une extrémité d'une plante est proche d'une extrémité de l'autre plante, elle cherche à se développer plus vite que l'autre pour l'étouffer. Au jeu de Go, les plantes sont les groupes de pierres. La façon dont les groupes grossissent en présence de groupes ennemis est spécifiée par les règles ou formes du jeu de l'opposition.

Reconnaissance

On reconnaît le jeu de l'opposition dès que des formes du type de celle de la figure *Elem-opposition* matchent.

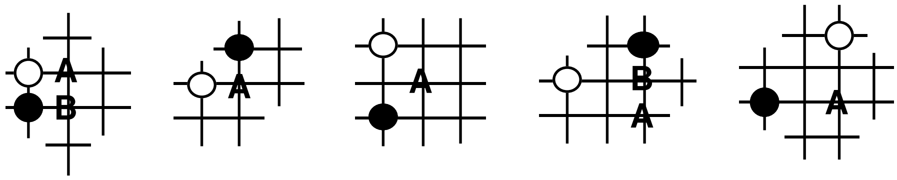


figure Elem-opposition

Les règles du jeu

Conditions terminales

Il n'y en a pas car le jeu de l'opposition dure un seul coup.

Règles pour jouer des coups

Voir figure *Elem-opposition*

L'aspect dynamique

Trivial car le jeu se réduit à { gagné | perdu }. Le premier qui joue gagne. L'arbre associé à la partie gauche et à la partie droite du jeu est de profondeur 1.

Autres informations importantes

Le joueur de Go a beaucoup d'idées concernant ces formes d'opposition. Les raisons de jouer ce coup dépendent du contexte et peuvent être complexes. L'idée la plus simple à notre avis est de jouer A ou B comme sur la figure *Elem-opposition* et d'expliquer ce type de coup par un concept appelé "opposition", par définition. Nous avons créé la base de règles "opposition". Nous avons établi que les coups engendrés par ces règles sont des coups d'"opposition".

Actuellement, la base contient 50 règles d'opposition. Elle a été constituée à la main. Elle est facile à maintenir.

Perspectives envisageables

Il serait bon de créer automatiquement cette base par programme[1]. Il faudrait aussi modéliser les conditions terminales de ce jeu plus précisément pour qu'il ne soit pas si rudimentaire.

[1]Le lecteur pourra se reporter au paragraphe Domaines voisins - Méta - pattern

Le jeu de la dilatation

Une image qui fait comprendre ce qu'est le jeu de la dilatation est celle d'une plante (une fougère par exemple) qui se développerait pour recevoir un maximum de lumière. Ainsi, dés qu'une extrémité de la plante est proche de lumière, elle cherche à se développer vers elle. La façon dont les groupes grossissent en présence de vide, est spécifiée par les règles ou formes du jeu de la dilatation.

Reconnaissance

On reconnaît le jeu de la dilatation dès que des formes du type de celle de la figure *Elem-dilatation* matchent.

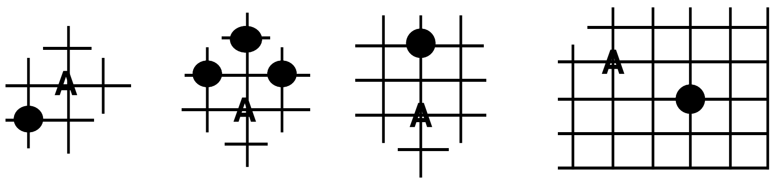


figure Elem-dilatation

Les règles du jeu

Conditions terminales

Il n'y en a pas car le jeu de la dilatation dure un seul coup.

Règles pour jouer des coups

Voir figure *Elem-dilatation.*

L'aspect dynamique

Trivial car le jeu se réduit à { gagné | perdu }. Le premier qui joue gagne. L'arbre associé à la partie gauche et à la partie droite du jeu est de profondeur 1.

Autres informations importantes

Actuellement, la base contient 15 règles de dilatation. Elle a été constituée à la main. Elle est facile à maintenir.

Le jeu du remplissage du vide

Une image qui fait comprendre ce qu'est le jeu du remplissage du vide est celle d'un jardinier qui voudrait semer une graine sur un espace inoccupé et ensoleillé. La graine semée pourra devenir une plante et réagir face à d'autres plantes se développant près d'elle. Ainsi, dés qu'un espace adapté à la culture est reconnu, le jardinier y sème une graine. La façon dont un joueur occupe les espaces vides du goban est spécifiée par les règles ou formes du jeu du remplissage du vide.

Reconnaissance

On reconnaît le jeu du remplissage du vide dès que des formes du type de celle de la figure *Elem-vide* matchent.

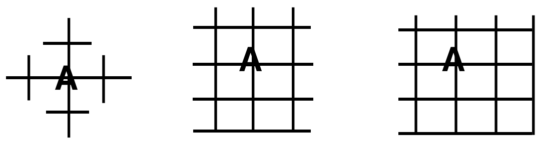


figure Elem-vide

Les règles du jeu

Conditions terminales

Il n'y en a pas car le jeu du remplissage du vide dure un seul coup.

Règles pour jouer des coups

Voir figure *Elem-vide.*

L'aspect dynamique

Trivial car le jeu se réduit à { gagné | perdu }. Le premier qui joue gagne. L'arbre associé à la partie gauche et à la partie droite du jeu est de profondeur 1.

Autres informations importantes

Actuellement, la base contient 10 règles de remplissage du vide. Elle a été constituée à la main. Elle est facile à maintenir.

La méthode pour identifier et formaliser les jeux du niveau élémentaire

Le but de ce paragraphe est de **présenter comment nous avons identifié et formalisé certains jeux**. Le plan de ce paragraphe est le suivant :

Le jeu de la séparation de l'adversaire, un complément par rapport au jeu de la connexion

Le jeu du point, outil général pour le jeu de l'œil et le jeu de la séparation du territoire en 2

Les jeux de l'opposition, de la dilatation, du remplissage du vide, généralisations des jeux topologiques

Le jeu de la séparation de l'adversaire, un complément par rapport au jeu de la connexion

Nous avons montré dans le chapitre "méthode" comment il était difficile d'identifier le concept de séparation de l'adversaire caché par le concept de connexion. Nous avons également montré au chapitre "domaines voisins" comment le théorème de Jordan en morphologie mathématique nous avait conforté dans l'idée de l'utilité du concept de séparation de l'adversaire sur un goban 4 connecté.

Ici, nous présentons la différence entre le concept de séparation de l'adversaire et le concept de connexion en classant des formes topologiques dans des tableaux à deux entrées, connexion et séparation. Ces exemples confirment que **connexion implique séparation de l'adversaire** mais que **la réciproque est fausse**. La figure *Elem-topo-statique* classe les patterns de la figure *Elem-connexion* suivant les l'interprétations statiques de la connexion et de la séparation.

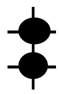
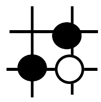
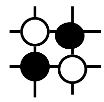
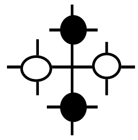
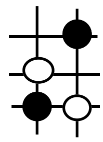
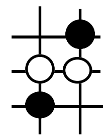
connexion / séparation	gagné	autre	perdu
gagné			
autre			
perdu			

figure Elem-topo-statique

Il faut noter que ce classement est purement statique et que les patterns montrés sont très petits. La figure *Elem-topo-2* montre des patterns un peu plus grands.

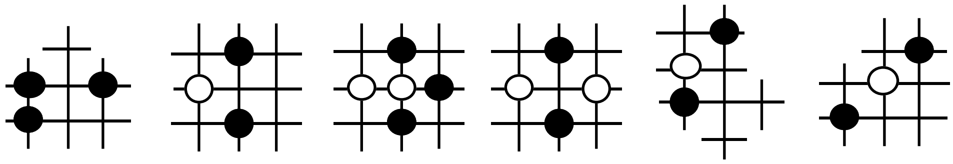


figure Elem-topo-2

La figure *Elem-topo-dynamique* classe les formes de la figure *Elem-topo-2* suivant un point de vue dynamique pour la connexion et la séparation.

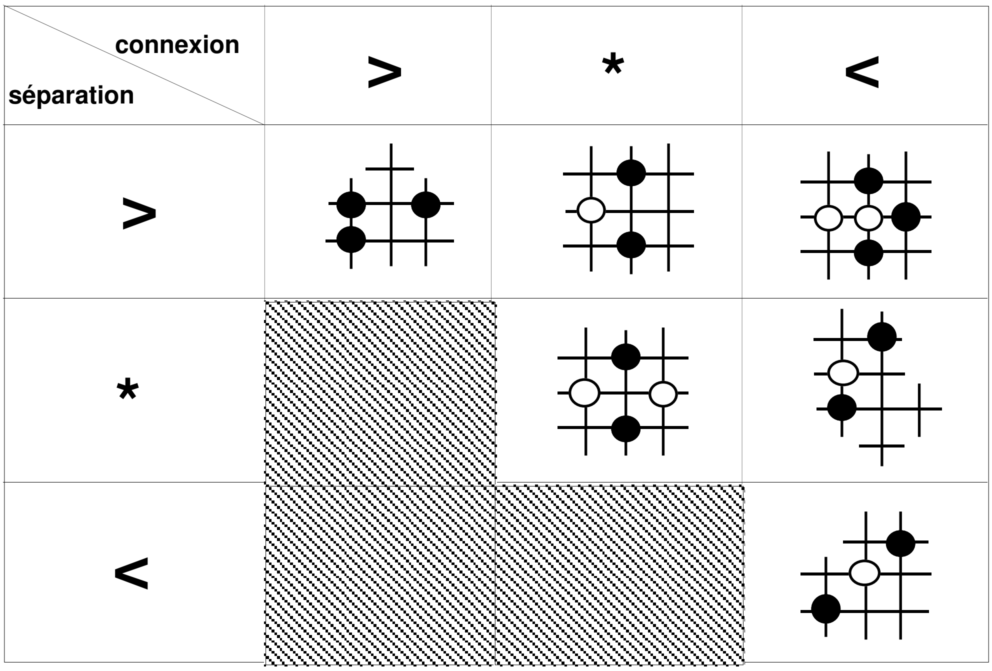


figure Elem-topo-dynamique

La haute densité d'information topologique contenue dans des patterns aussi petits que 5-5 nous a amené à obtenir beaucoup de règles dynamiques plutôt que statiques.

L'intérêt du concept de séparation ne peut apparaître en observant uniquement le niveau élémentaire. Il faut voir le niveau itératif pour voir son intérêt : **la connexion servira pour construire les groupes** et **la séparation pour construire les fractions**.

Le jeu du point, outil général pour le jeu de l'œil et le jeu de la séparation du territoire en 2

Chronologiquement, nous avons modélisé le jeu de l'œil de taille 1 avec une **méthode spécifique**. Puis, nous avons modélisé le jeu de l'œil de taille 2 avec une **méthode assez lourde**. Puis nous avons modélisé le jeu de la séparation du territoire en deux avec une **méthode plus générale**. Ce faisant, nous avons remarqué les points communs entre jeu de l'œil et jeu de la séparation du territoire en 2. Cela a donné naissance au jeu du point. Nous avons modélisé à nouveau le jeu de l'œil (taille 1 et 2) et de la séparation du territoire en deux en utilisant le jeu du point.

Il est intéressant de raconter cette expérience car elle montre plusieurs choses :

La méthode lourde nous a permis de mettre en œuvre des outils automatiques de création de patterns.
La méthode lourde aurait dû être reconnue comme telle à l'époque pour nous faire sentir que nous étions sur une mauvaise piste.

Le jeu du point était bien caché derrière les concepts d'œil et de territoire.
La méthode spécifique du jeu de l'œil de taille 1 est une compilation spatiale de la méthode du jeu du point.

Nous ne pouvions pas faire autrement pour en arriver là.
Modéliser le jeu de l'œil est plus compliqué qu'il n'y paraît.
On ne trouve pas les choses du premier coup.

Dans ce paragraphe nous présentons :
la méthode spécifique de l'œil de taille 1,
la méthode lourde de l'œil de taille 2,
la méthode spécifique de l'œil de taille 1, une compilation de la méthode générale.

La méthode spécifique de l'œil de taille 1

Au départ nous ne savions pas comment modéliser le jeu de l'œil de façon générale par rapport à la taille de l'œil. Nous avons donc modélisé le jeu de l'œil de taille 1.

Pour plus de clarté, nous donnons à nouveau des exemples d'yeux de taille 1.

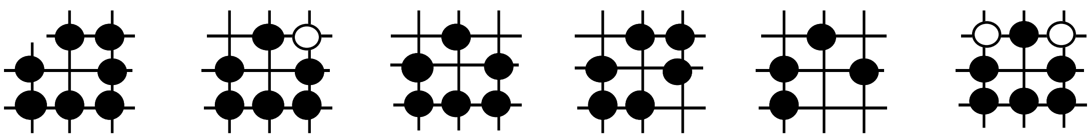


figure Elem-règle-œil

Le lecteur remarquera que ces yeux ont des caractéristiques communes : une intersection centrale vide, des intersections voisines de la même couleur (noire) et des intersections en diagonales de couleur indifférente mais généralement de la même couleur (noire).

Les règles de la classe œil possèdent une partie gauche avec un pattern qui contient donc implicitement l'information suivante[1] :

un ensemble d'intersections L (souvent de taille 1 ou 2) sur lequel l'œil est situé,

un ensemble d'intersections frontière F sur lequel se trouvent les pierres amies frontière de l'œil,

un ensemble d'intersections D en diagonale de l'ensemble L qui doit être le plus amical[2] possible,

L'œil de la figure *Elem-règle-œil* est implicitement de la forme donnée par la figure *Elem-règle-œil-LFD* :

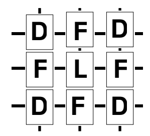


figure Elem-règle-œil-LFD

Le paysage nécessaire au déclenchement du jeu de l'œil étant fixé, se pose la question de savoir si l'œil est vrai ou faux - si le jeu de l'œil est > ou <.

Un proverbe existe à ce sujet dans la littérature sur le Go :

Si 3 diagonales sur 4 sont amicales, l'œil est vrai.

Il est très utile. Nous l'avons utilisé pour modéliser l'état statique d'un œil. Nous n'avons pas utilisé les règles du jeu du point. C'est **ce proverbe qui est la caractéristique de ce que nous appelons la méthode spécifique**.

Le lecteur remarquera que seul l'oeil de gauche de la figure *Elem-règle-œil* est statiquement vrai.

La méthode lourde de l'œil de taille 2

Si la taille de L vaut 2, cela se complique et aucun proverbe comme le précédent n'émerge. La méthode spécifique ne s'applique plus. La figure *Elem-règle-œil-2* montre des exemples d'œil de taille[3] 2

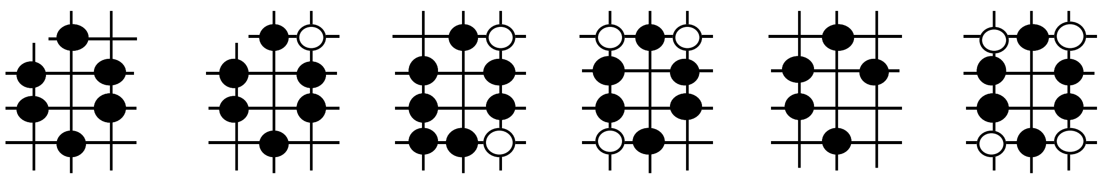


figure Elem-règle-œil-2

[1]De la même manière que dans le jeu du point (cf. jeu du point - reconnaissance).
[2]Dans le contexte des yeux, amical peut signifier une pierre amie dessus ou un jeu de l'intersection ami.
[3]C'est un abus de langage qui signifie que l'ensemble L a une taille 2.

On retrouve encore conceptuellement les ensembles L, F et D décrits pour les yeux de taille 1 et le jeu du point (cf. jeu du point - reconnaissance). La figure *Elem-règle-œil-2-LFD* illustre ce fait.

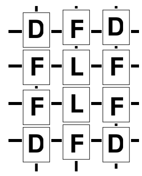


figure Elem-règle-œil-2-LFD

Pour des petites fractions de taille 2, cela se compliquait et nous n'avions pas trouvé d'autre moyen que d'**engendrer explicitement toutes les formes** avec une fraction de taille 2, une disposition du bord et des intersections diagonales, vides, amies ou ennemies. Cela donnait au moins 200 patterns. Ce qui était beaucoup pour un composant élémentaire. C'est pourquoi nous appelons cette méthode, la **méthode lourde**. L'intérêt de ce travail passé est d'avoir écrit **un programme qui engendre automatiquement une base de règles** même si celle-ci est spécifique des yeux du Go. Le nombre de règles a beaucoup diminué (80) lorsque nous avons classé les formes suivant 3 classes ludiques (>, <, *) et que nous avons généralisé automatiquement ces règles.

Nous avons abandonné cette façon de reconnaître les yeux lorsque les parties réelles ont montré que cette base de 200 règles était incomplète[1]. En effet de nombreux yeux en formation ne sont pas complètement entourés, comme le 5ème œil de la figure *Elem-règle-œil-2*. Il aurait fallu engendrer encore toutes les règles correspondant à ce cas. Peut-être aurait-il fallu 500 ou 1000 règles pour traiter ce seul problème ! Notre méthode commençait à montrer ses limites. Nous pensions à l'époque avoir sans doute une mauvaise représentation du problème. mais nous ne savions pas faire autrement. En modélisant le jeu de la séparation du territoire en 2, nous avons trouvé une nouvelle technique et nous avons donc abandonné cette base de règles d'œil de taille 2.

[1]Au Go, une base de formes bornées à une partie du goban est toujours incomplète. Mais notre base était vraiment incomplète au sens où des formes qui paraissaient utilisées par des joueurs moyens manquaient encore dans notre base.

La méthode spécifique du jeu de l'œil de taille 1, compilation de la méthode générale.

L'avantage d'avoir des règles de type "point" est de **découpler spatialement les patterns** qui déclenchent les règles engendrant les coups. Dans l'ancienne méthode, un pattern servant aux yeux de taille 1 (resp. 2) contenait l'état de 9 (resp. 12) intersections et une information implicite sur les ensembles L, F et D. Avec la nouvelle méthode, un pattern est **plus petit** (7 intersections maximum) et **contient explicitement l'information** sur les ensembles L, F et D. Le nombre de règles de la nouvelle méthode est nettement inférieur (40) à celui de l'ancienne méthode : 50 règles pour les yeux de taille 1 plus 200 règles pour les yeux de taille 2. A l'exécution, les patterns de type "point" peuvent se comparer à la position réelle suivant 4 (resp. 2) directions pour les yeux de taille 1 (resp. 2), comme le montre la figure *Elem-règles-point-LFD-œil-de-taille-1* (resp. figure *Elem-règles-point-LFD-œil-de-taille-2*).

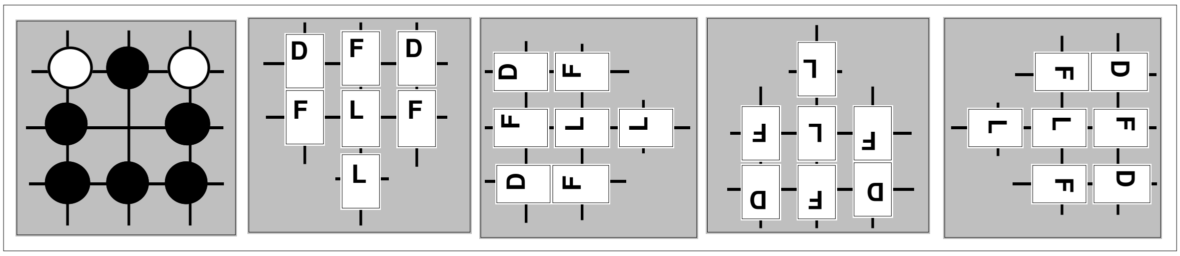


figure Elem-règles-point-LFD-œil-de-taille-1

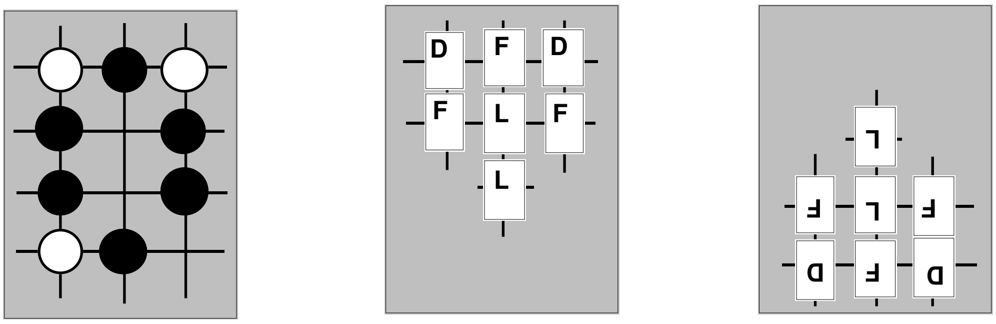


figure Elem-règles-point-LFD-œil-de-taille-2

Ces figures montrent que la nouvelle méthode a une efficacité variable suivant la taille de l'œil. Elle essaie de comparer des patterns suivant 4 directions pour une taille 1 (4 pour 1). Elle essaie de comparer les patterns suivant 2 directions pour une taille 2 (2 pour 2). Pour des territoires étudiés de plus grande taille, l'efficacité dépend de la forme du territoire. Mais, de façon générale, **plus la taille du territoire étudié est grande, meilleure est l'efficacité de la nouvelle méthode**.

D'autre part, **la base de règles de type point est utilisée quelle que soit la taille du territoire étudié**.

Après coup, nous avons remarqué que notre ancienne méthode, si elle était spécifique et dépendante de la taille de l'œil étudié, offrait l'avantage d'être plus efficace pour les yeux de taille 1 que la nouvelle méthode. Toujours après coup, nous avons remarqué que **l'ancienne méthode pour l'œil de taille 1 était une compilation spatiale de la nouvelle méthode**. C'est pourquoi elle était plus efficace dans le cas de l'œil de taille 1.

Jusque là, nous pensions que les automatismes ou connaissances spécialisées d'un être humain devenaient non conscientes. Ce qui précède est un contre-exemple. Au départ de la construction de notre modèle, nous avions mis dans le modèle ce dont nous étions conscients : la méthode spécifique dans le cas de l'étude des yeux. Après un long travail de modélisation, nous avons découvert la méthode générale dont nous n'étions pas conscients au départ. Ainsi, **dans le cas de l'étude de l'œil, les connaissances spécialisées étaient conscientes et les connaissances générales étaient non conscientes** - mais conscientisables.

Les jeux de l'opposition, de la dilatation, du remplissage du vide, simplifications successives des jeux topologiques sur le critère du nombre d'extrémités.

Les règles de notre système contiennent une information implicite qui est importante : les liaisons ou embryons de liaison qui se caractérisent par le nombre d'extrémités de ces liaisons ou embryons de liaison (0, 1, 2 ,3 ou 4 extrémités). Chronologiquement, nous avons modélisé les jeux topologiques (connexion et séparation de l'adversaire) où 2 extrémités amies et un voisinage ennemi sont présents, cela donne 200 règles. Puis, nous avons modélisé le jeu de l'opposition où 1 extrémité amie et 1 extrémité ennemie sont présentes, cela donne 80 règles. Puis, nous avons modélisé le jeu de la dilatation où 1 extrémité amie est présente, cela donne 20 règles. Puis, nous avons modélisé le jeu du remplissage du vide où aucune extrémité n'est présente, cela donne 10 règles. Sur le critère du nombre d'extrémités, nous sommes allés du **particulier vers la simplicité. Nous avons conscience du particulier**. Celui-ci cache la simplicité comme l'arbre cache la forêt.

Conclusion

Résumé de l'état actuel

La précision

Nous avons obtenu des concepts suffisamment **précis** pour l'utilisation que les niveaux supérieurs en font. Une partie de la précision d'une information est contenue dans sa définition même. Nous pensons avoir défini nos jeux correctement pour avoir ce premier niveau de précision statique. Une autre partie de la précision est obtenue par calcul. Le niveau élémentaire est essentiellement **dynamique**. La contrepartie de la précision apportée par le calcul est le temps de réponse. Nous avons privilégié la précision face au temps de réponse.

Des concepts élémentaires mais dépendants

Les concepts obtenus sont **élémentaires**. Ils dépendent les uns des autres par différents types de relations : **particularité**, **implication**, **utilisation**. Nous résumons ci-dessous les quelques relations de dépendance entre les concepts actuels de notre système.

"topologique" est plus particulier que "opposition" qui est plus particulier que "dilate" qui est plus particulier que "vide",
"connexion" implique "séparation de l'adversaire",
"oeil" utilise "point",
"séparation du territoire en deux" utilise "point", "chaîne-poison" et "fraction-poison".
Tous les concepts du niveau élémentaire utilisent "intersection" (qui appartient au niveau du dessous, le niveau zéro).

La dépendance apparente actuelle des concepts ne doit pas cacher un fait important : notre travail n'a pas consisté à trouver des concepts dépendants, au contraire, il a consisté à rendre ces concepts de plus en plus autonomes, simples et indépendants les uns des autres. Il faut bien comprendre que nos concepts actuels sont dépendants mais beaucoup moins qu'au départ.

La complétude

Actuellement, nous avons :

200 règles topologiques,
80 règles d'opposition,
20 règles de dilatation,
10 règles de remplissage du vide,
40 règles du point,
5 règles d'œil,
15 règles de chaîne-poison,
50 règles de fraction-poison.

Ce qui fait un total de **plus de 400 règles**. Pendant une période, les erreurs d'INDIGO étaient causées essentiellement par une quantité insuffisante de règles dans chaque classe de règles. Actuellement, les erreurs d'INDIGO sont causées plus par les limitations de notre modèle que par l'absence de règles. Seuls les besoins du niveau supérieur (le niveau itératif) pourront nous pousser à construire de nouvelles classes de règles et savoir si le niveau est complet.

Le temps de réponse

Nous pensons que les traitements du niveau élémentaire ne sont **pas encore assez rapides.** D'autres programmes de Go utilisent des heuristiques qui peuvent être approximatives mais ont l'avantage d'être rapides [Kraszek 1988] [Boon 1991] au détriment de la précision. La rapidité de ces programmes est une interrogation pour nous.

Les informations sous-entendues dans notre présentation et utilisées par INDIGO.

Le bord

Pour ne pas compliquer la présentation, nous n'avons pas parlé de l'**effet de bord** du goban. Il est évident que **le bord a un impact énorme** sur les règles et les jeux présentés ici. En gros, les joueurs de Go considèrent que les intersections des quatre premières lignes du goban[1] subissent l'effet du bord du goban. Pour les jeux en général, la présence et la disposition du bord modifient le résultat des jeux. Il faut donc plus de règles pour décrire ces jeux. Pour une règle matchant au centre du goban, et suivant son degré de symétrie, il faut faire correspondre 2 à 4 règles, matchant avec le bord. Donc, si une base de règles ne tient pas compte du bord, la base de règles qui tient compte du bord peut contenir environ 4 fois plus de règles!

Le lecteur comprendra qu'il est intéressant de concevoir les bases de règles indépendamment du bord si cela est possible. (la base *poison* est dans ce cas). Souvent, cela n'a pas été possible.

D'autre part, le bord peut modifier la structure implicite d'une règle. Par exemple, le jeu de la séparation de l'adversaire, près du bord, peut ne requérir qu'une seule extrémité noire, le bord jouant le rôle de la 2ème extrémité.

Utilisation du jeu de l'intersection

Nous n'avons pas alourdi la présentation des jeux avec l'utilisation du jeu de l'intersection au niveau inférieur. Il faut savoir que **tous les jeux utilisent le jeu de l'intersection**. Les jeux topologiques, de l'opposition, de la dilatation et du remplissage du vide vérifient que les coups conseillés par leurs calculs sont * pour le jeu de l'intersection. Sinon, ces jeux déconseillent le coup et changent convenablement leur résultat dynamique. Le jeu du point utilise le jeu de l'intersection pour les intersections diagonales. Pour un seul calcul du jeu du point, le jeu de l'intersection peut être appelé plusieurs fois. Le jeu de la séparation du territoire en deux n'utilise surtout pas le jeu de l'intersection pour les coups conseillés par les "poisons" car ceux-ci ont un jeu de l'intersection négatif.

[1] 99% des intersections sur 9-9, 80% sur 13-13 et 50% sur 19-19!

Résumé de la méthode utilisée

De la dépendance vers l'indépendance :

Au tout départ, les concepts à peine identifiés étaient indépendants : ils étaient trop jeunes pour avoir eu le temps de tisser entre eux un réseau de relations. Très vite, ils sont devenus **très dépendants** les uns des autres, car mal identifiés et mal agencés les uns par rapport aux autres. Notre travail a essentiellement consisté à les disséquer pour les couper en concepts plus élémentaires et indépendants les uns des autres et les replacer les uns par rapports aux autres. Pour ce faire, la validation informatique nous a beaucoup aidé. A l'arrivée, nos concepts sont toujours dépendants mais plus **indépendants** que dans la phase initiale.

Du particulier au simple :

Au départ, nous avions conscience de connaissances particulières (les connexions et séparations qui font interagir **2** extrémités amies avec **1** voisinage ennemi) et nous avons commencé par les modéliser. Ensuite, nous avons modélisé des connaissances de plus en plus simples : les oppositions qui n'ont qu' **1** voisinage ami et **1** voisinage ennemi, les dilatations qui n'ont qu' **1** voisinage ami, les formes vides qui ont **0** voisinage ami ou ennemi) Nous sommes allés **du particulier vers le simple.**

Du caché au découvert :

Le concept de séparation est resté longtemps caché par celui de connexion. Les proverbes tout faits qui s'appliquent aux yeux simples ont caché le concept de "point".

Le concept de séparation a été dévoilé grâce aux disciplines voisines (topologie et morphologie mathématique). Le concept de "point" a été dévoilé grâce à la synergie qui existe entre le jeu de l'œil et le jeu de la séparation du territoire en 2. Au départ le concept de point était imbriqué et caché par l'œil (ou territoire de taille 1). Il a commencé à être visible avec l'œil double (ou territoire de taille 2). Il est devenu assez visible et nécessaire avec la séparation du territoire en 2 (territoire de taille 3 ou plus). Étant général, le concept du "point" a permis de résoudre les 3 cas d'alors : oeil de taille 1, œil de taille 2 et territoire de taille 3 ou plus. Nous avons alors réuni les deux cas d'œil. Finalement les yeux et la séparation du territoire en deux sont tous les deux résolus par le concept de "point".

Les concepts cachés derrière des concepts particuliers, ont été difficiles à découvrir.

Correspondance avec le degré de conscience humain

Certains jeux du niveau élémentaire en cachent d'autres. Nous résumons ce fait par les 3 figures qui suivent.

Les jeux topologiques

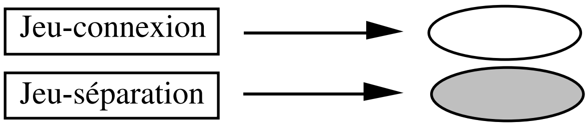


figure Correspondance-jeu-simple

Les joueurs humains utilisent le terme de "*connexion*" pour parler du concept de connexion (!), le terme "*coupe*" pour parler du concept opposé, la déconnexion. Ils utilisent aussi les termes "*coupe*" ou "*séparation*" pour parler du concept de séparation, qui peut être vu comme une séparation de l'adversaire, même si celui-ci est absent (cf. Partie 2, verbalisations, histoire comme ça).

Les jeux vitaux

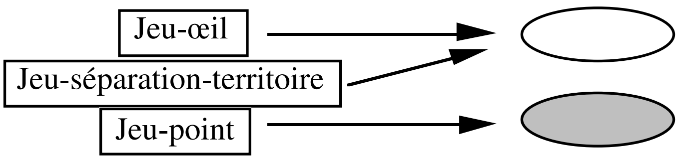


figure Correspondance-œil-et-territoire-cache-point

Les joueurs humains utilisent le terme d' "*œil*" pour parler du concept d'œil (!), l'expression "*faire deux yeux*" pour parler du concept de séparation de territoire en deux. Ils utilisent les termes "*vrai*" ou "*faux*" ou "*demi-*" pour parler du résultat du jeu de l'œil (respectivement >, <, * dans INDIGO). Ils utilisent les expressions "*boucher le territoire*" ou "*faire une bouse*" ou "*prise de couilles*" (sic) pour parler des coups joués pendant le jeu du point. Ces expressions imagées sont le produit apparent d'un concept caché que nous avons difficilement conscientisé : le jeu du point.

Les jeux morphologiques

Du simple au complexe, les jeux du remplissage du vide, de la dilatation, de l'opposition et topologiques (connexion et séparation) ont des correspondances différentes avec les degrés de conscience d'un joueur humain, fonction du niveau du joueur.

Première aperçu de l'apprentissage du débutant.

Chez le joueur novice complet (30 ème kyu), nous avons la figure suivante :

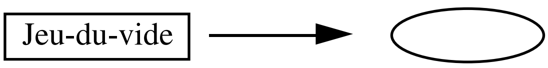


figure Correspondance-topo-oppos-dilat-vide-30

Le jeu du vide est conscient chez le novice complet.

Chez le joueur débutant (25 ème kyu), nous avons la figure suivante :

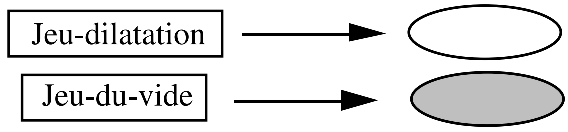


figure Correspondance-topo-oppos-dilat-vide-25

Le jeu du vide devient non conscient et le jeu de la dilatation est appris et conscient.

Chez le joueur faible (20 ème kyu), nous avons la figure suivante :

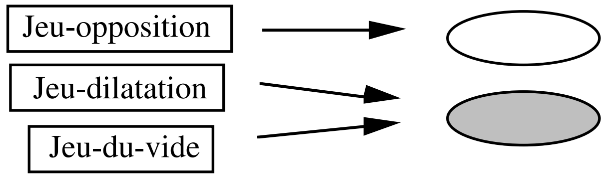


figure Correspondance-topo-oppos-dilat-vide-20

Le jeu de la dilatation devient non conscient et le jeu de l'opposition est appris et conscient.

Chez le joueur moyen (15 ème kyu), nous avons la figure suivante :

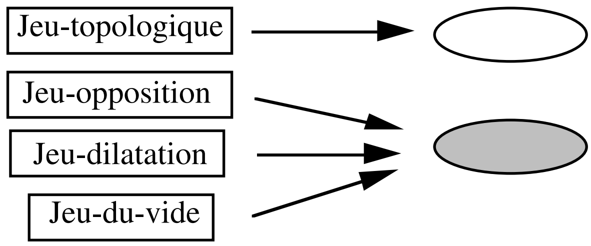


figure Correspondance-topo-oppos-dilat-vide-15

Le jeu de l'opposition devient non conscient et les jeux toplogiques sont appris et conscients.

Pour nous faire comprendre nous avons cité des niveaux approximatifs : 15 ème kyu, 20 ème kyu, 25 ème kyu ou 30 ème kyu mais évidemment les choses ne sont pas si simples en réalité.

<u>Deuxième aperçu de l'apprentissage du débutant.</u>

Selon nous, une meilleure approximation de l'apprentissage du joueur humain dans ces domaines topologiques et morphologiques correspondrait plutôt aux figures suivantes où nous avons rajouté des ovales gris foncé qui représentent les connaissances intuitives.

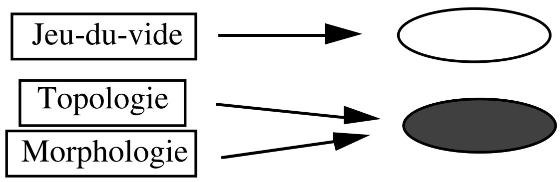


figure Apprend-topo-morpho-30

Le novice complet (30 ème kyu) dispose de connaissances topologiques et morphologiques générales et intuitives. Il est conscient du jeu du remplissage du vide.

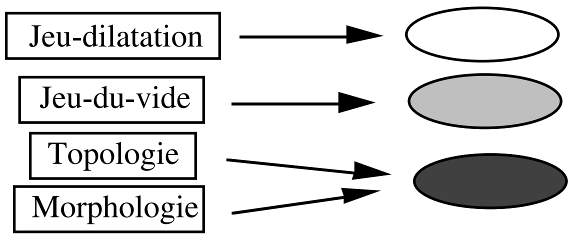


figure Apprend-topo-morpho-25

Le joueur débutant (25 ème kyu) adapte les connaissances - intuitives - topologiques et morphologiques au cas particulier du jeu de la dilatation au Go pendant que le jeu du remplissage du vide devient un automatisme conscientisable.

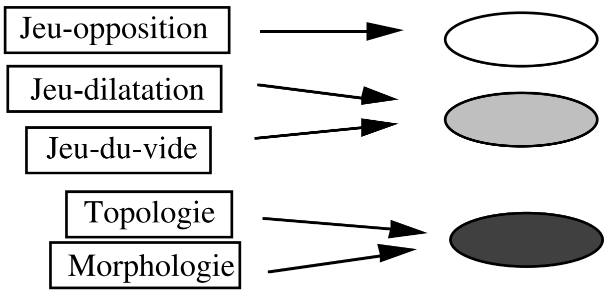


figure Apprend-topo-morpho-20

Le joueur faible (20 ème kyu) adapte encore les connaissances intuitives au cas particulier du jeu de l'opposition au Go pendant que le jeu de la dilatation devient un automatisme conscientisable.

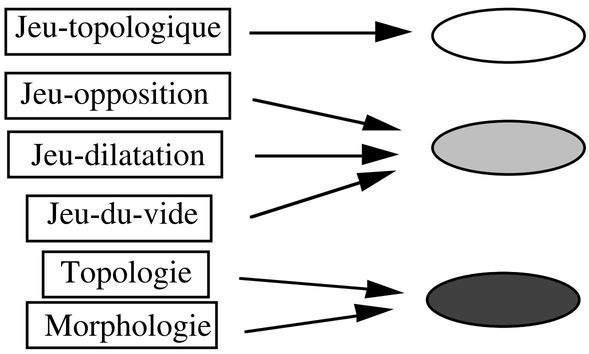


figure Apprend-topo-morpho-15

Le joueur moyen (15 ème kyu) adapte encore les connaissances intuitives au cas particulier du jeu de l'opposition au Go pendant que le jeu de la dilatation devient un automatisme conscientisable.

La facilité de l'apprentissage chez certains joueurs nous fait faire à la remarque suivante :

Le processus d'apprentissage d'un joueur de Go correspondrait non pas à la seule acquisition de connaissances nouvelles, inexistantes avant l'apprentissage, mais plutôt à une *découverte* de *connaissances générales* et *cachées*, *pré-existantes* à l'apprentissage. Le processus de découverte serait, en fait, une *adaptation au jeu de Go*, ou *spécialisation pour le jeu de Go*, des connaissances générales sur le monde réel.

Les termes ou expressions associés au concept de remplissage du vide sont "*occuper les gros points"* (en début ou milieu de partie) ou "*jouer les dame"* (en fin de partie).

Les termes ou expressions associés au concept de dilatation sont "*renforcer"* ou "*bétonner"* (pour le défenseur) et pour le concept dual, l'érosion : "*réduire"* ou "*attaquer"* (pour l'attaquant).

Les termes ou expressions associés au concept d'opposition sont "*sortir"* ou "*encercler"* ou "*jouer le coup naturel"* ou "*prendre le point vital"*.

On observe que ces termes peuvent n'avoir aucun rapport apparent avec les concepts qu'ils désignent. Par exemple, le terme "*attaque*" associé au concept d'érosion est un terme général que nous associons plutôt à une notion ludique : l'attaquant est le premier qui joue en général. Autre exemple, les mots "*naturel*" ou "*vital*" sont fréquemment employés. Ils correspondent, selon nous, à des concepts généraux de morphologie ou topologie.

Bibliographie

[Berlekamp & Wolfe 1994] - E.R. Berlekamp, D. Wolfe - Mathematical Go Endgames - Nightmares for the Professional Go Player - Ishi Press International - San Jose, London, Tokyo - 1994

[Boon 1989] - M. Boon - Pattern matcher of Goliath - Computer Go 13, winter 89-90

[Boon 1991] - Mark Boon - Overzicht van de ontwikkeling van een Go spelend programma - Afstudeer scriptie informatica onder begeleiding van prof. J. Bergstra - 1991

[Kraszek 1988] - J. Kraszek - Heuristics in the life and death algorithm - Computer Go n°9, winter 88-89

[Wolf 1992] - T. Wolf - tsumego with Risiko - School of Math. Sciences, Queen mary & Westfield College, Mile end road, London E1 4NS, England - 1992

[Wolf 1993] - T. Wolf - Quality improvements in tsumego mass production - Cannes Workshop Computer Go - 1993

LE NIVEAU "ITÉRATIF" : GROUPE, TERRITOIRE, ESPACE VIDE ET FRACTION

Introduction

Nous présentons dans ce paragraphe les objets du niveau "itératif". La caractéristique des objets de ce niveau est d'être constitués par itérations. Il y a 4 types d'objets "itératifs" :

les groupes
les territoires
les espaces vides
les fractions

Selon nous, les **groupes** sont les objets les plus importants au Go. Ils sont constitués itérativement à partir des chaînes de la règle du jeu au moyen des connexions >. Leur caractéristique principale est d'avoir des propriétés permettant de savoir s'ils sont "morts", "vivants" ou entre les deux, et ainsi d'interpréter une position. Une description des groupes de notre système INDIGO figure également dans [Bouzy 1994b].

Les **territoires** permettent deux choses : qualifier la base de vie d'un groupe et donner un score à une position en terme de points. Ils sont constitués, parallèlement aux groupes, itérativement[1] avec l'opérateur de fermeture morphologique appliqué aux groupes.

Les **espaces vides** reflètent la course pour remplir le vide d'une partie de Go. Ils sont constitués itérativement avec l'opérateur d'ouverture morphologique appliqué aux intersections non contrôlées du goban.

Les **fractions** partagent le goban en rendant relativement indépendants les objets situés dans des fractions différentes. Ce qui est utile quand on veut faire des calculs le plus localisés possible. Elles permettent aussi de détecter les encerclements des groupes. Elles sont constituées itérativement à partir du goban entier fractionné par les séparations >.

[1] Le nombre d'itérations dépend de l'échelle à laquelle on veut les construire.

Le plan de ce niveau est le suivant :

Les groupes

Nous présentons dans ce paragraphe le concept de groupe. C'est le concept le plus intéressant car il permet d'interpréter une position de Go puis de décider du coup global à jouer.

Introduction

Nous montrons d'abord ce qu'est un "groupe" au sens de la **connexion**. Ensuite nous présentons les propriétés d'un groupe en deux parties : les **propriétés intrinsèques ou "intérieures"** et les **propriétés interactives ou "extérieures"**. Une propriété intrinsèque d'un groupe représente la capacité du groupe à être autonome ou indépendant de son voisinage. Une propriété interactive d'un groupe synthétise à un premier niveau l'état des interactions de ce groupe avec ses voisins. Enfin, nous présentons la propriété de "santé" qui synthétise à un deuxième niveau l'état des différentes propriétés intrinsèques et interactives. Nous verrons que si l'état de la santé d'un groupe est inférieur à un seuil, le groupe est "mort" et une "catastrophe" se produit. Les idées qui sous-tendent les propriétés du concept de groupe sont le paradigme **intérieur-extérieur**, la **synthèse** et la **similarité d'échelle**. Nous illustrons la présentation des groupes avec un exemple.

Construction

La figure *Groupe-exemple* montre une position de Go.

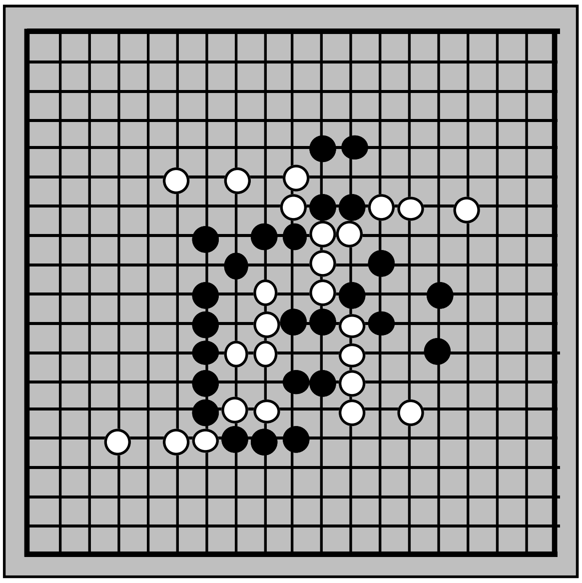
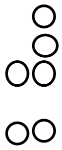
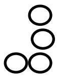


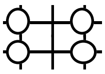
figure Groupe-exemple

Par connexion.

Le modèle considère les connexions > comme des éléments qui permettent de fusionner les groupes entre eux.

Par exemple, le groupe: 

est construit par la fusion des chaînes  et .

par l'intermédiaire de la connexion > : 

La connexion est fondamentale car elle permet de construire les groupes sur lesquels le raisonnement ultérieur est basé.

La figure *Groupe-reconnu* montre les groupes reconnus par connexion.

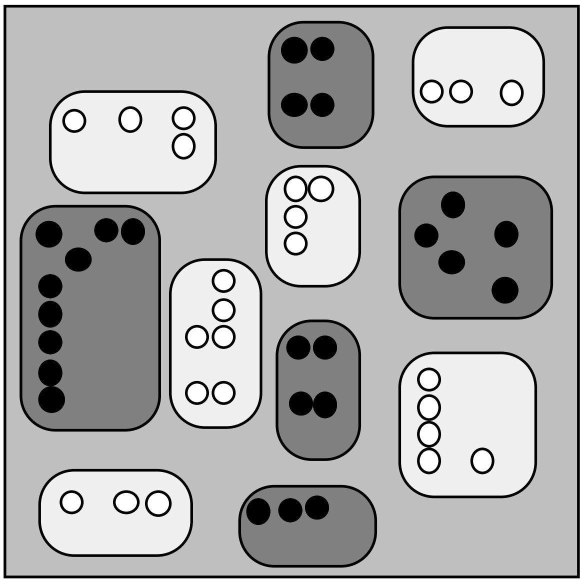


figure Groupe-reconnu

La construction des groupes basée sur les connexions > est classique dans les programmes de Go [Boon 1991] [Fotland 1992] [Cazenave 1994].

Par la morphologie mathématique

Pendant une période, nous avions construit les groupes avec les connexions > et l'opérateur de fermeture morphologique appliqué aux chaînes du goban. C'était une approximation trop forte qui donnait des groupes trop grands qui étaient déconnectés facilement par des adversaires humains ou artificiels possédant le jeu de la connexion. Nous avons abandonné cette voie. Pourtant Go Intellect [Chen 1989] utilise des techniques proches de la morphologie mathématique pour reconnaître les groupes.

Par fusion de groupes morts ou par "catastrophe"

Nous verrons au paragraphe "catastrophe" comment un groupe "mort" permet aux groupes qui le "mangent" de fusionner autour de lui en un seul groupe. Nous ne pouvons pas présenter ce type de construction sans avoir discuté des propriétés d'un groupe.

Les propriétés des attributs d'un groupe en général

Les attributs du groupe suivent les idées de synthèse, de distinction entre "intérieur" et "extérieur" et de similarité avec l'échelle de pierres.

Les idées de séparation intérieur-extérieur et de synthèse se retrouvent dans la taxonomie des classes.

Les attributs du groupe sont répartis dans 3 sous-classes d'attributs. Les flèches de la figure *type-attribut* indiquent une relation d'héritage 'est-un'.

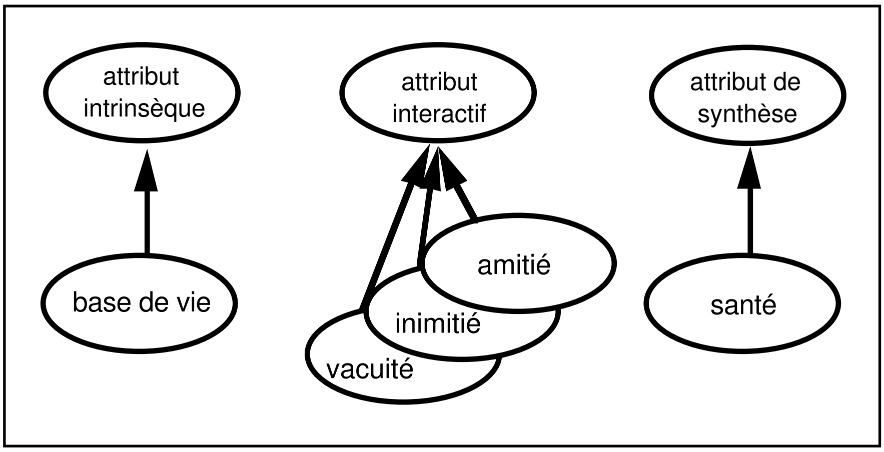


figure type-atttribut

Chaque attribut du groupe reflète une synthèse.

Un premier niveau de synthèse

La base de vie d'un groupe synthétise l'information sur le nombre d'yeux d'un territoire contrôlé par ce groupe.

L'amitié d'un groupe synthétise l'information sur le nombre et l'état des interactions amies du groupe.

L'inimitié d'un groupe synthétise l'information sur le nombre et l'état des interactions ennemies du groupe.

La vacuité d'un groupe synthétise l'information sur le nombre et l'état des interactions vides du groupe.

La figure *Groupe-premier-niveau-synthèse* résume visuellement ce premier niveau de synthèse.

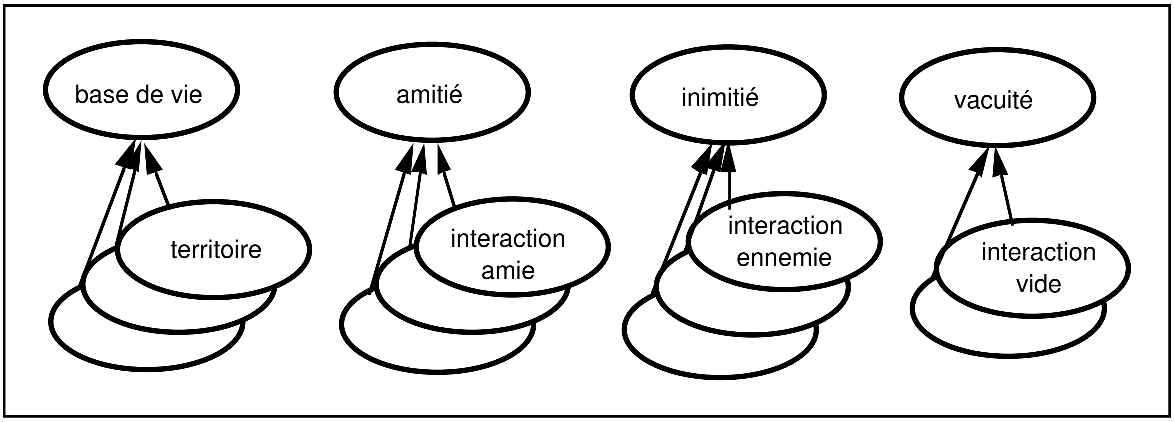


figure Groupe-premier-niveau-synthèse

Un deuxième niveau de synthèse

Enfin, la santé d'un groupe synthétise à un deuxième niveau l'information sur la base de vie, l'amitié, l'inimitié et la vacuité.

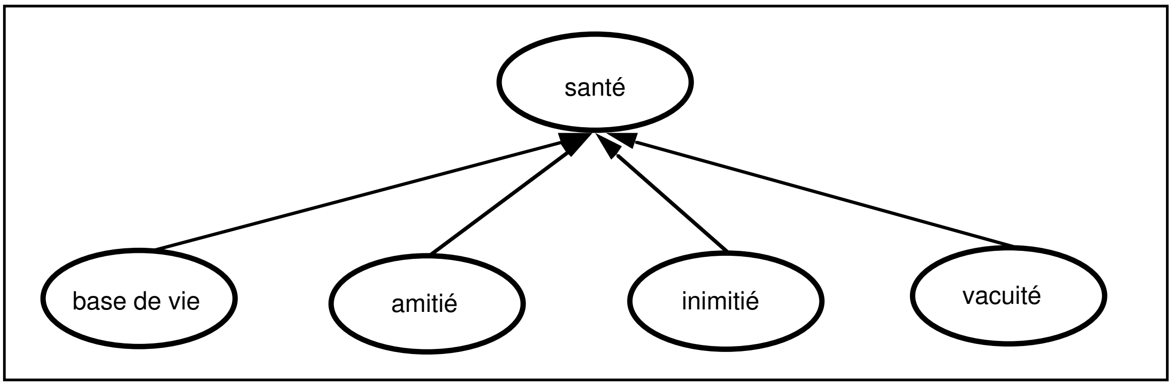


figure deuxième-niveau-synthèse

Un attribut intrinsèque

Un attribut intrinsèque est un attribut du groupe indépendant du voisinage extérieur au groupe. Il décrit une propriété **intérieure** du groupe.

La base de vie

Les groupes définis au paragraphe précédent ont une propriété intrinsèque, la _base de vie_. La base de vie est un des attributs de la classe groupe. La base de vie fait la somme de tous les yeux contenus dans les territoires contrôlés par le groupe. On peut additionner des résultats du jeu de l'œil ou du jeu de la séparation du territoire en 2 selon la taille du territoire. Cette somme ou synthèse de premier niveau est faite suivant ce que nous appelons les "règles de recomposition dynamiques" au paragraphe Domaines Voisins - Méta - Métajeu de la Partie 4 de ce document.

Il est très important de savoir si les jeux calculés au niveau "intermédiaire" sont indépendants ou non.

Des jeux (des yeux?) indépendants

Si les jeux sont indépendants, on peut "**additionner**" les résultats des jeux. On compte :
2 pour un jeu de la séparation du territoire en 2 >,
1,5 pour un jeu de la séparation du territoire en 2 *,
1 pour un jeu de la séparation du territoire en 2 < ou un jeu de l'œil >,
0,5 pour un œil *.
Si cette somme est supérieure ou égale à 2, la base de vie est >. Si elle vaut 1,5 alors, la base de vie est *. Si elle vaut 0, 0,5 ou 1 alors la base de vie est <[1]. On peut aussi écrire explicitement les règles de recomposition de résultats :

Si un groupe a plus de 2 yeux> ou (1 œil > et 2 yeux *) ou plus de 4 yeux * alors base de vie>.
Si un groupe a (1 œil> et 1 œil*) ou 3 yeux* alors base de vie*.
Sinon base de vie<.

Cela fait apparaître l'importance des nombres 0, 1, 2, 3 et 4. Nous y reviendrons dans la conclusion.

Des jeux (des yeux?) dépendants

Si les jeux sont dépendants on ne peut pas additionner les résultats comme ci-dessus. On a le choix entre deux solutions. La première est plus précise et plus coûteuse que la deuxième. Elle consiste à recalculer les jeux dépendants en parallèle. La deuxième consiste à utiliser des règles de recomposition floues du type :

Si un groupe a au moins 3 yeux> alors base de vie>.

Ces règles ont l'avantage de ne pas conclure à des résultats faux et de ne pas cacher l'incertitude *. Par contre, ces règles n'effectuent plus vraiment leur rôle de synthèse ou d'abstraction. L'attribut base de vie du groupe n'est plus qu'un collecteur d'yeux qui perd la précision attendue au départ.

En pratique, le temps étant un paramètre crucial, nous avons utilisé cette approche dans INDIGO.

[1]Cependant avoir une somme égale à 0, 0,5 ou 1 n'est pas indifférent en cas de "combat" mais c'est un autre problème.

Exemple

La figure *Groupe-base-de-vie* montre à gauche la position considérée avec des territoires contrôlés par un groupe. Les  indiquent les intersections contrôlées par un groupe. Elles sont chacune le lieu de calcul sur le jeu de l'œil, tous <[1]. A droite de cette figure, nous montrons la base de vie des groupes.

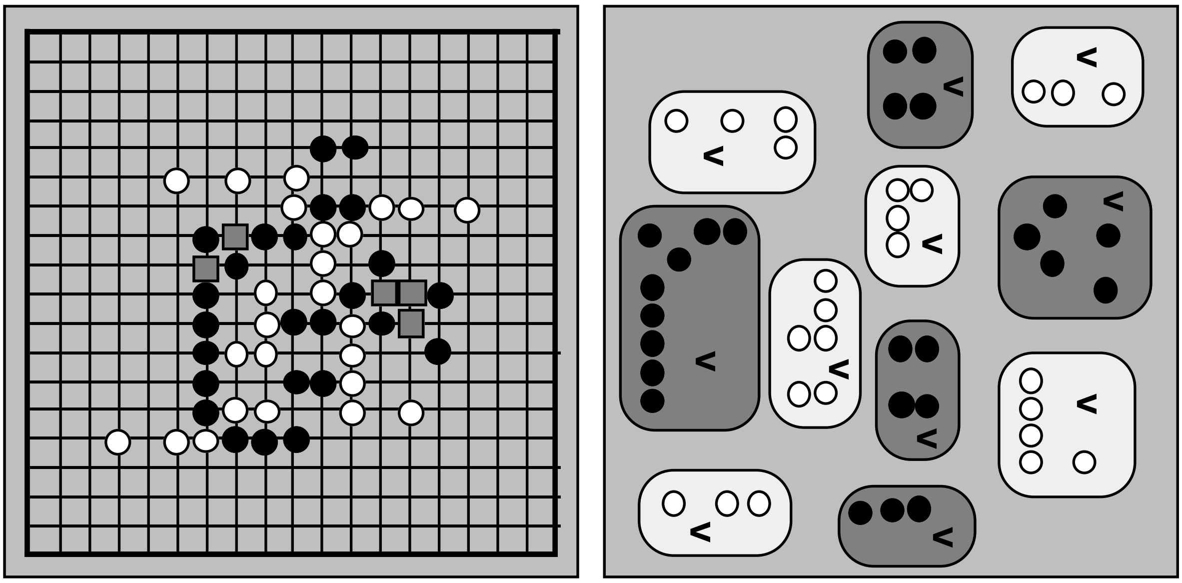


figure Groupe-base-de-vie

[1]L'exemple choisi ici ne reflète pas au mieux le concept de base de vie. Les groupes sont instables et ont des bases de vie <. Nous le gardons tout de même par souci d'homogénéité avec l'ensemble de la présentation du concept de groupe. Un meilleur exemple se trouve dans le paragraphe Domaines Voisins - Méta. Nous conseillons vivement au lecteur de s'y reporter.

Les attributs interactifs

Les attributs interactifs d'un groupe décrivent les propriétés du groupe liées à son **extérieur**. L'extérieur du groupe est constitué de groupes amis et ennemis et d'espaces vides. Les attributs interactifs synthétisent l'état des interactions du groupe avec ces voisinages amis, ennemis et vides.

Un groupe a des interactions amies avec des groupes amis synthétisées dans le concept d'*amitié*, des interactions ennemies avec des groupes ennemis synthétisées dans le concept d'*inimitié* et enfin des interactions vides synthétisées dans le concept de *vacuité*. L'amitié, l'inimitié et la vacuité sont des attributs de la classe groupe.

La vacuité

En théorie

La vacuité d'un groupe synthétise l'état de l'ensemble des interactions du groupe avec des espaces vides. La position de la figure *Groupe-Vacuité* montre un espace vide à grande échelle qui fait le tour du goban.

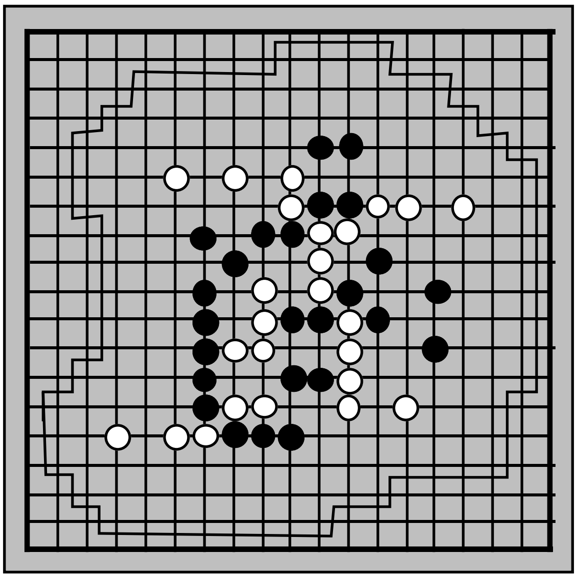


figure Groupe-Vacuité-théorique

La vacuité d'un groupe synthétise l'état de l'ensemble des interactions vides du groupe de la manière suivante:

Si un groupe a plusieurs[1] *interactions vides alors vacuité>.*
Si un groupe a une interaction vide alors vacuité.*
Si le groupe n'a pas d'interaction vide alors vacuité<.

Nous avons abandonné cette idée car la reconnaissance des espaces vides est coûteuse en temps machine, elle dépend de l'échelle à laquelle les espaces vides sont reconnus, elle dépend de l'état des groupes (et on tombe dans un cercle vicieux...). De plus, les règles ci-dessus devraient prendre en compte la taille ou l'échelle des espaces vides.

[1]Voir en conclusion de la présentation des groupes, la discussion à propos de la synthèse de premier niveau effectuée par un attribut de groupe

<u>En pratique :</u>

En pratique, la vacuité d'un groupe synthétise l'état des jeux de la dilatation du groupe. La figure *Groupe-vacuité-pratique* montre à gauche l'ensemble des coups engendrés par le jeu de la dilatation pour tous les groupes du goban. Ces coups sont indiqués par des D.

Pour calculer la valeur des vacuités des groupes en pratique, nous avons utilisé les règles suivantes.

Si un groupe a plus de deux coups de dilatation alors vacuité>.
Si un groupe a un ou deux coups de dilatation alors vacuité.*
Si le groupe n'a pas de coup de dilatation alors vacuité<.

A droite est indiquée la valeur de vacuité de chaque groupe.

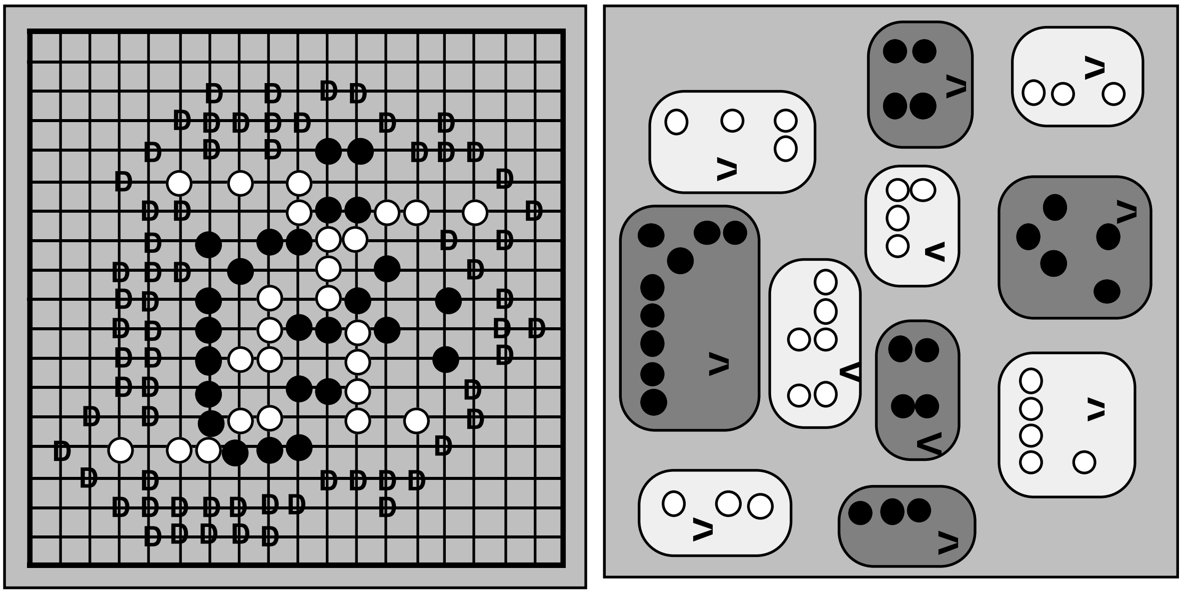


figure Groupe-vacuité-pratique

La notion de vacuité présentée dans cet exemple est simple. En fait la vacuité pour un groupe généralise la notion de liberté pour une chaîne. On peut considérer que la vacuité présentée ici collecte les "*libertés d'ordre 2 ou 3*" d'un groupe. La règle qui dit que "*la vacuité est < si aucune liberté d'ordre 2 ou 3 n'existe*" cache un point important qui existe dans notre système INDIGO. En cas de combat entre groupes à vacuité <, la course aux libertés permet de savoir quel groupe est le plus fort. Le nombre de libertés du groupe est stocké dans l'objet vacuité. Ce nombre sera utilisé en cas de combat[1].

---

[1]Le concept de combat est rudimentaire dans INDIGO. Il sera présenté au paragraphe sur les "fractions".

L'amitié

L’amitié d’un groupe synthétise l’état de l’ensemble des interactions amies du groupe. Une interaction amie entre deux groupes synthétise l'état de l'ensemble des connexions * entre les deux groupes.

Connaître la valeur des interactions avec les groupes ennemis voisins :

Connaître la valeur d'une interaction amie est simple. INDIGO utilise les règles suivantes :

*Si les 2 groupes de l'interaction amie ont plus de 2 connexions * indépendantes alors interaction amie*>[1]
*Si les 2 groupes de l'interaction amie ont 1 connexion * alors interaction amie **
*Si les 2 groupes de l'interaction amie ont 0 connexion * alors interaction amie* <[2]

Le lecteur remarquera l'analogie de ces règles (connexion*, interaction amie) avec les règles (yeux>, base de vie).

La figure *Groupe-amitié-interactions* donne des exemples d'interactions amies entre 2 groupes amis voisins avec la valeur de l'interaction.

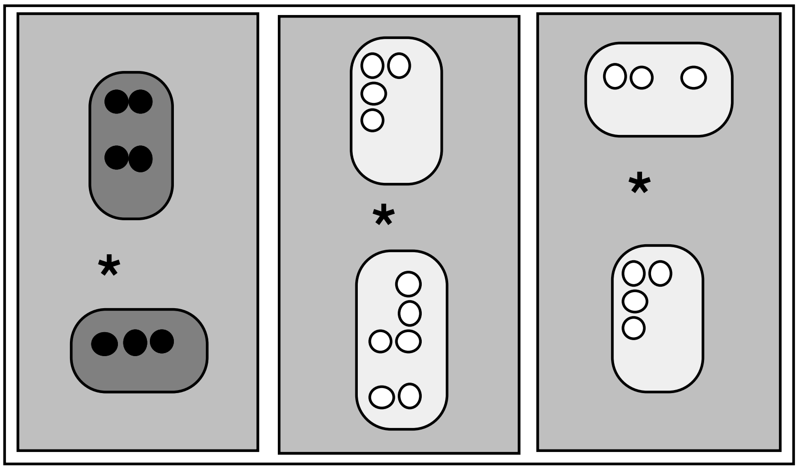


figure Groupe-amitié-interactions

Synthétiser la valeur des interactions en une seule valeur :

L’amitié d’un groupe synthétise l’état de l’ensemble des interactions amies du groupe de la manière suivante:

Si un groupe a plusieurs[3] *interactions amies alors amitié>.*
Si un groupe a quelques[4] *interactions amies alors amitié*.*
Si un groupe n’a pas d’interaction amie alors amitié<.

La figure *Groupe-amitié-valeur* donne à droite la liste des groupes reconnus avec la valeur de l'attribut amitié. A gauche se trouvent les coups conseillés par l'amitié. B signifie un coup Blanc, N

[1]Auquel cas, les deux groupes fusionnent en un seul groupe et on recommence.
[2]Cas impossible en pratique car si un groupe n'a pas de connexion * avec un autre, il n'en est pas voisin et l'interaction amie n'est pas construite.
[3]"plusieurs" est à fixer: 3, 4, 5 ? Se reporter à la discussion sur la synthèse de premier niveau en conclusion.
[4]"quelques" est aussi à fixer. Même remarque.

un coup Noir et A un coup pour les deux joueurs. Il faut noter que les coups indiqués sur la figure sont des coups de connexion ou de déconnexion.

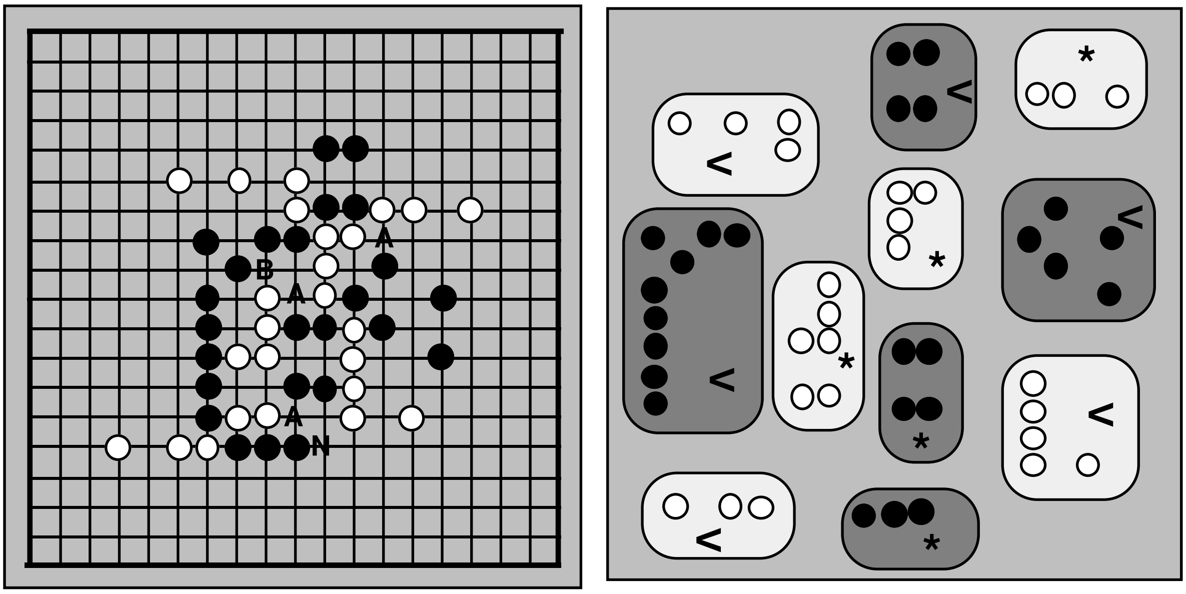


figure Groupe-amitié-valeur

L'amitié est l'attribut le plus prioritaire dans INDIGO. Par conséquent, sur la position exemple, INDIGO jouera les coups qui coupent ou connectent les groupes. Nous supposerons qu'INDIGO, jouant contre lui-même, jouera la séquence de droite de la figure *Groupe-amitié-séquences*. Une séquence plus normale que celle jouée par INDIGO correspond au diagramme de gauche de la figure *Groupe-amitié-séquences*.

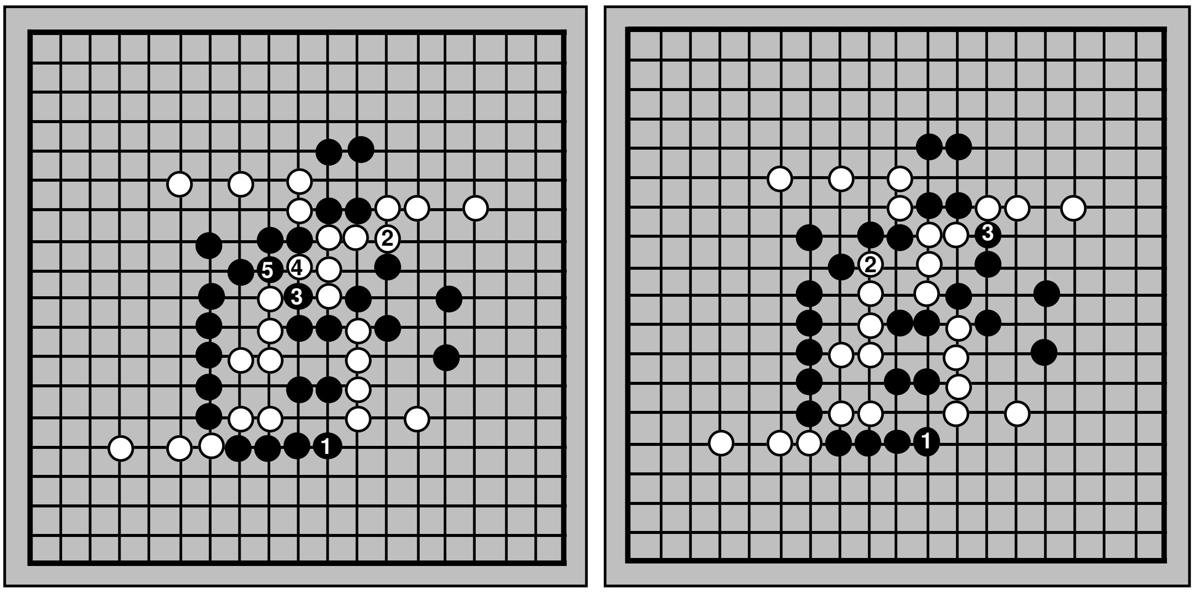


figure Groupe-amitié-séquences

INDIGO joue de cette manière car le niveau "itératif" est statique, il ne fait pas de calcul. Il ne voit pas, à ce moment là, que 2 serait meilleur en 3. Pour le comprendre, anticipons sur la description du niveau global qui sera faite plus loin.

Pour choisir le coup global, INDIGO affecte à un coup une valeur égale à la somme des tailles des groupes instables qui conseillent ce coup. Au moment de choisir le deuxième coup de la séquence, INDIGO affecte la valeur 6 + 4 = 10 au coup 2 (la somme des tailles des deux groupes conseillant de jouer en 2) et la valeur 4 + 3 = 7 au coup 3 (la somme des tailles des deux groupes conseillant de jouer en 3). Il joue donc le coup 2 au lieu de jouer 3... Au coup suivant, INDIGO joue logiquement 3. Puis, lors de l'interprétation qui précède le coup 4, il s'aperçoit enfin que le groupe blanc central créé est mort, mais il est trop tard...

Ce niveau est victime de l'effet horizon à horizon 0. Lorsque les machines seront plus puissantes, INDIGO pourra calculer au niveau "itératif". Ce niveau sera dynamique.

Pour la suite, nous prendrons pour exemple la position issue de la séquence jouée par INDIGO. Nous allons montrer comment l'attribut inimitié permet de savoir que le groupe blanc central est mort.

L'inimitié

L'attribut inimitié d'un groupe synthétise les résultats des interactions ennemies en un seul résultat.

<u>Information en entrée :</u>

Pour la suite nous prenons comme exemple la position de droite de la figure *Groupe-amitié-séquences* , jouée par INDIGO. A ce niveau, des informations ont déjà été calculées. Nous les résumons dans la figure *Groupe-inimitié-information-entrée.*

A gauche, nous rappelons la position prise comme exemple. A droite, nous donnons les groupes avec les valeurs des attributs base de vie (b), amitié (a) et vacuité (v).

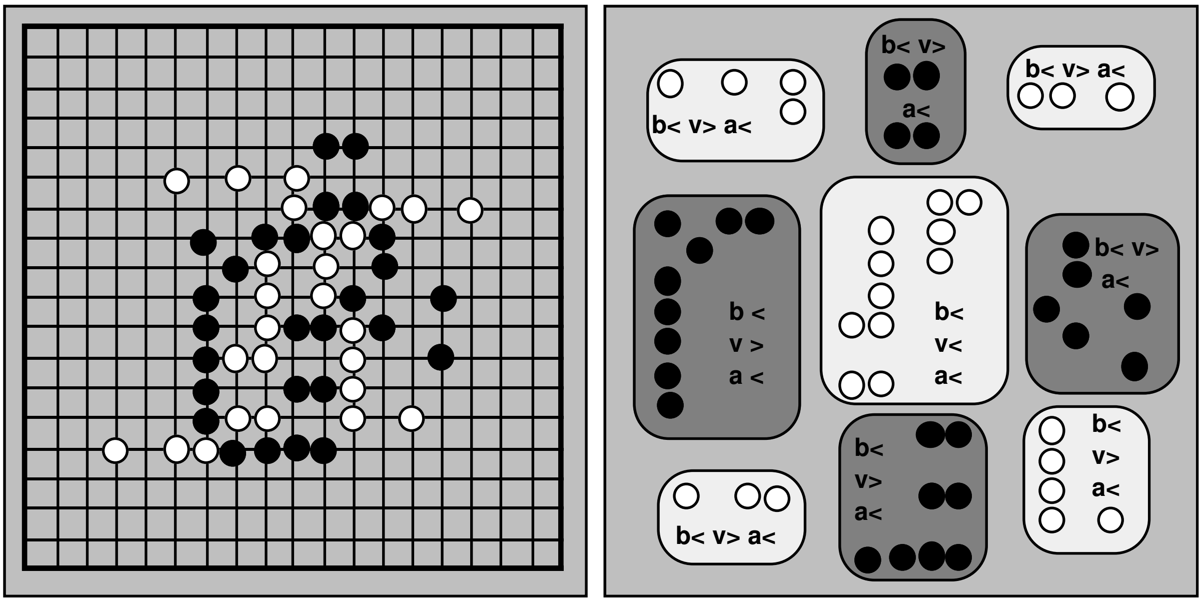


figure Groupe-Inimitié-information-entrée

L'information sur la base de vie, l'amitié et la vacuité des groupes ennemis voisins d'un groupe donné est nécessaire pour connaître la valeur de l'inimitié de ce groupe.

Connaître la valeur des interactions avec les groupes ennemis voisins :

Connaître la valeur d'une interaction ennemie n'est pas très simple. INDIGO utilise les règles suivantes :

les règles pour obtenir la valeur ludique de l'interaction

Soient A et B les deux groupes, i(A, B) leur interaction ennemie.
Soient b(x) la base de vie de x, v(x) la vacuité de x, a(x) l'amitié de x pour x = A ou B.

S'il existe x dans { A, B } et y dans {b, v, a} tel que y(x) = * alors i(A, B) = '*'
Si pour tout y dans {b, v, a} on a y(A)=y(B)='<' alors **_combat_**_._
Si pour tout y dans {b, v, a} on a y(A)=y(B) alors i(A, B)='='.
Si pour tout y dans {b, v, a} on a y(A)≥y(B) alors i(A, B)='>'.
Si pour tout y dans {b, v, a} on a y(A)≤y(B) alors i(A, B)='<'.
Sinon (A, B)= ☁.

Le symbole ☁ signifie "flou", non déterminable. Ce symbole est un symbole très pratique car il regroupe tout ce que l'on ne sait pas déterminer, c'est-à-dire beaucoup de choses!

Le combat, cas critique

combat est le cas où tous les attributs des deux groupes de l'interaction ennemie sont <: pas de base de vie possible, pas d'espace vide autour, pas d'ami. Dans ce cas, il faut utiliser des heuristiques sur le nombre d'yeux, le nombre de libertés communes, le nombre de libertés extérieures de chaque groupe et conclure [Müeller 1993].

Soient no(x) le nombre d'yeux de x, ne(x) le nombre de libertés extérieures de x pour x dans {A, B} et nc le nombre de libertés communes à A et B

Si un œil est *, le faire :

Si no(A)=1/2 alors i(A, B) = *
Si no(B)=1/2 alors i(A, B) = *

Si pas d'œil du tout pour A et B :

_Si no(A) = no(B) = 0_

_Si nc = 0 ou 1:_
_Si ne(A) = ne(B) i(A, B) = *_
_Si ne(A) > ne(B) i(A, B) = >_
_Si ne(A) < ne(B) i(A, B) = <_

_Si nc ≥ 2:_
_ne(A) > ne(B) + nc -1 i(A, B) = >_
_ne(B) > na(A) + nc -1 i(A, B) = <_
_ne(A) = ne(B) + nc -1 i(A, B) = { > | 0 }_
_ne(B) = ne(A) + nc -1 i(A, B) = { 0 | < }_
autres i(A, B) = 0

Si un œil contre un œil

_no(A) = no(B) = 1_

_nc = 0:_
_ne(A) = ne(B) i(A, B) = *_
_ne(A) > ne(B) i(A, B) = >_

$ne(A) < ne(B) \qquad i(A, B) = <$

$nc \geq 1:$

$ne(A) > ne(B) + nc \qquad i(A, B) = >$
$ne(B) > ne(A) + nc \qquad i(A, B) = <$
$ne(A) = ne(B) + nc \qquad i(A, B) = \{ > | 0 \}$
$ne(B) = ne(A) + nc \qquad i(A, B) = \{ 0 | < \}$
autres $\qquad i(A, B) = 0$

Si un œil contre pas d'œil

$no(A) = 1$ *et* $no(B) = 0$

$ne(B) > ne(A) + nc \qquad i(A, B) = <$
$ne(B) = ne(A) + nc \qquad i(A, B) = *$
$ne(B) < ne(A) + nc \qquad i(A, B) = >$

Dans le cas de notre exemple, ces règles ne sont pas utiles car l'exemple ne tombe pas dans le cas où deux groupes voisins sont avec leurs attributs base de vie, vacuité, amitié <. Nous verrons au paragraphe Domaines Voisins - IAD un exemple qui utilise largement ces règles.

Les valeurs ludiques de quelques interactions représentatives de notre exemple

La figure *Groupe-inimitié-interactions-1* donne la liste des interactions ennemies qui font intervenir le groupe blanc central avec, à chaque fois, la valeur de l'interaction.

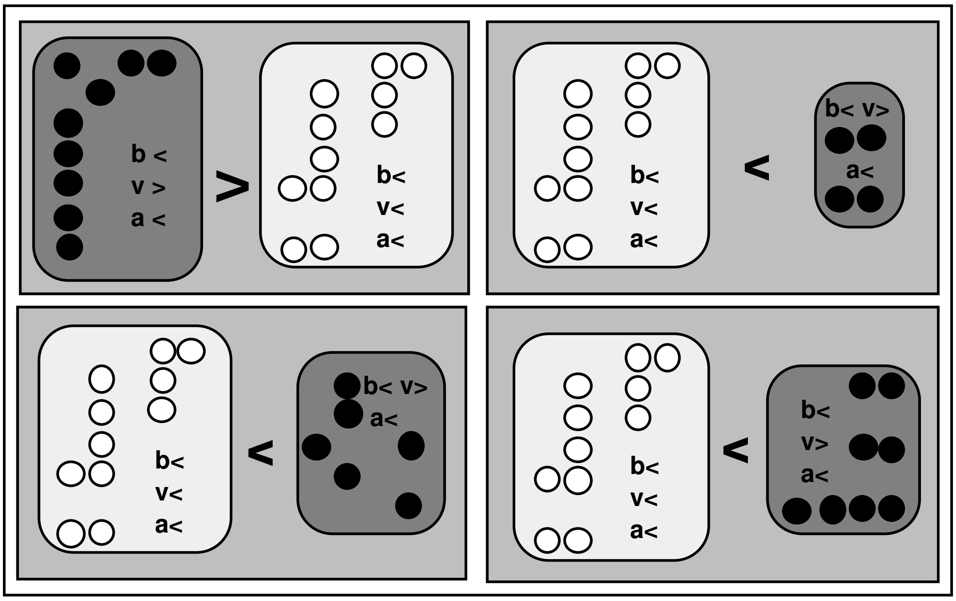


figure Groupe-inimitié-interactions-1

On observe que toutes les interactions du groupe blanc central sont < de son point de vue.
La figure *Groupe-inimitié-interactions-2* donne deux autres interactions ennemies.

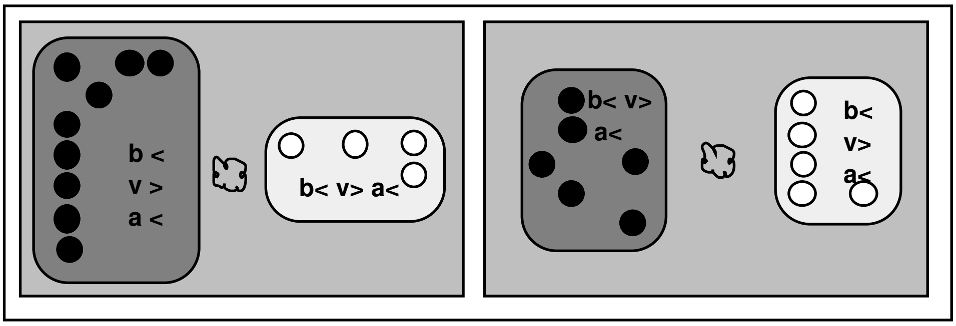


figure Groupe-inimitié-interactions-2

<u>Synthétiser la valeur des interactions en une seule valeur :</u>

L'inimitié d'un groupe synthétise l'état de l'ensemble des interactions ennemies du groupe de la manière suivante:

*Si au moins une interaction ennemie est * alors inimitié*.*
*Si pas d'interaction * et au moins une interaction ennemie est 0 alors inimitié 0.*
*Si pas d'interaction * et pas d'interaction 0 et au moins une interaction ennemie > alors inimitié>*
Si toutes les interactions ennemies sont < alors inimitié<.
Sinon interaction ennemie

La figure *Groupe-inimitié-valeur* donne la liste des groupes reconnus avec la valeur de l'attribut inimitié.

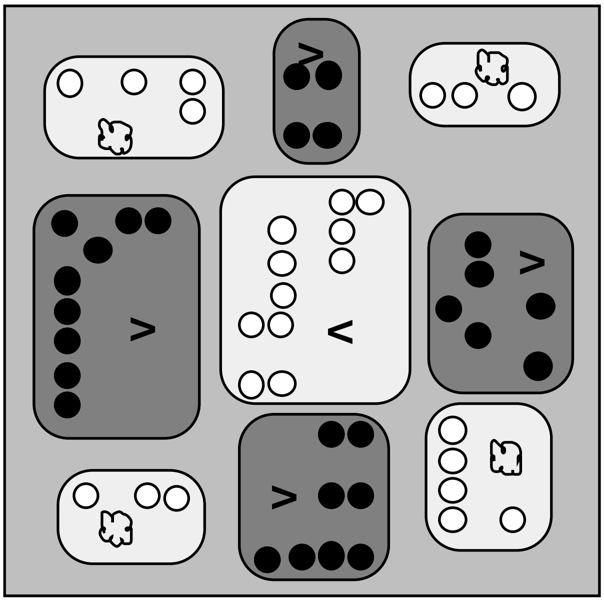


figure Groupe-inimitié-valeur

Il faut noter que le symbole est absorbant pour l'opération de synthèse inimicale.

La synthèse

L'ensemble des propriétés d'un groupe sont synthétisées à un deuxième niveau dans la propriété de *santé*. La santé est un des attributs de la classe groupe.

La santé

La synthèse des propriétés intrinsèques et interactives du groupe est faite de la manière suivante:

Si base de vie> alors santé>
Si base de vie< et amitié< et inimitié< et vacuité< alors santé<
Si base de vie< et amitié< et inimitié0 et vacuité< alors santé0
Si base de vie ou amitié* ou inimitié* ou vacuité* alors santé**
Si un groupe voisin a une santé< alors santé>[1]
Sinon santé

La figure *Groupe-santé-valeur* donne la valeur de la santé des groupes de la figure *Groupe-inimitié*

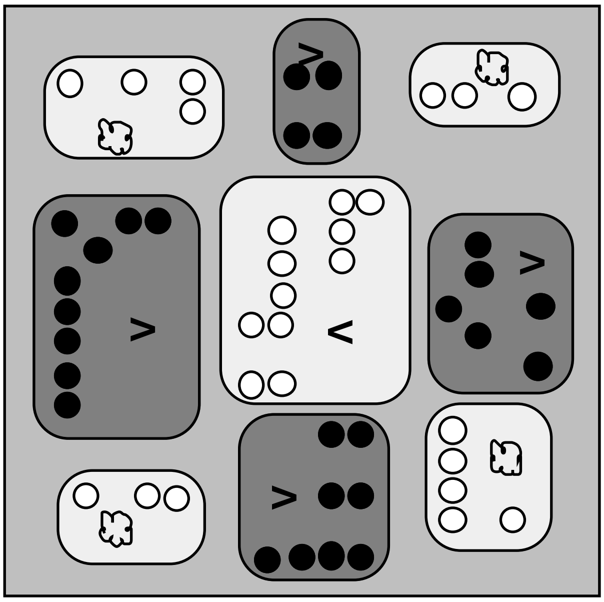


figure Groupe-santé-valeur

La catastrophe

Si la santé d'un groupe est négative, une catastrophe se produit. Quand une catastrophe se produit, les groupes ennemis "mangent" le groupe mort en fusionnant autour de lui pour former un groupe plus gros. Nous avons utilisé le terme "catastrophe" parce qu'à ce moment de la partie, la description du goban, le score de la partie (et la psychologie des deux joueurs !) subissent une discontinuité sensible.

La figure *Groupe-catastrophe* montre l'état des groupes après l'action de la règle de catastrophe.

[1]État transitoire car, dans ce cas, une catastrophe suit (cf. paragraphe suivant).

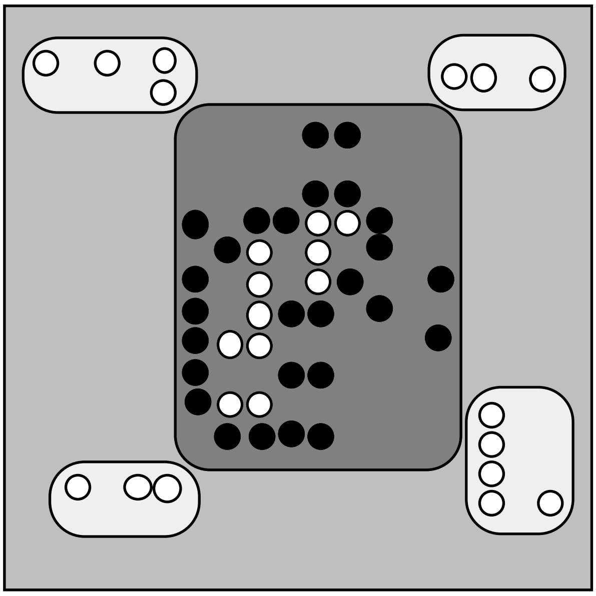


figure Groupe-catastrophe

On repart à zéro avec de nouveaux groupes, plus gros, sans valeur d'attribut. Le lecteur remarquera qu'un groupe peut être construit par connexions > mais aussi construit comme le produit d'une catastrophe comme nous l'avions annoncé au paragraphe sur la construction des groupes.

Tant que des catastrophes se produisent le modèle fait fusionner les groupes. Le nombre de groupes diminue à chaque catastrophe pour s'arrêter lorsqu'aucun groupe n'a de santé <.

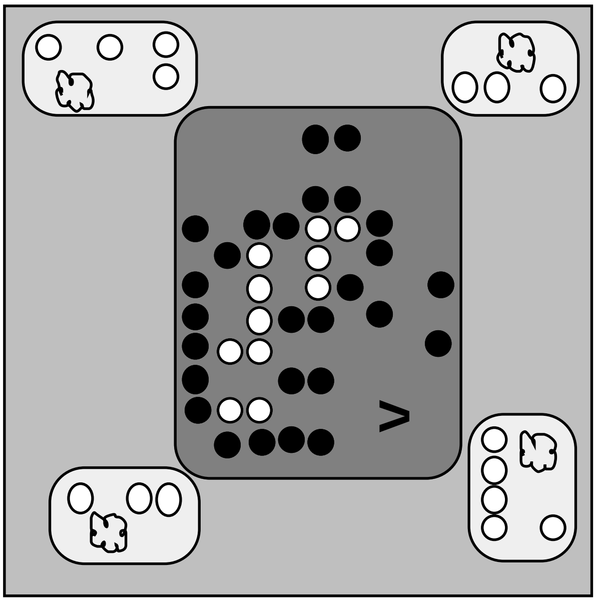


figure Groupe-fin-interpétation-goban

Conclusion

Nous avons présenté le concept de groupe et ses propriétés tel qu'il est représenté dans le système INDIGO. Nous résumons cette présentation en l'enrichissant, si possible, de correspondances avec les facultés humaines.

La connexion

La création du groupe s'appuie sur le concept de **connexion**.

Correspondance avec le degré de conscience humain

Chez un joueur humain, la notion de connexion est évidemment plus complexe que dans INDIGO. En plus de la connexion décrite ci-avant, un joueur humain utilise des connaissances générales, profondes et intuitives sur la connexion comme c'est représenté sur la figure *Correspondance-groupe-connexion*.

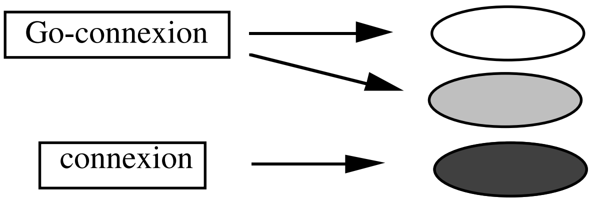


figure Correspondance-groupe-connexion

Nous pouvons faire la même remarque que celle faite au niveau élémentaire. Le processus d'apprentissage de la connexion d'un joueur de Go correspond non pas à la seule acquisition de connaissances nouvelles, inexistantes avant l'apprentissage, mais plutôt à une *découverte* de *connaissances générales* et *cachées*, *pré-existantes* à l'apprentissage. Le processus de découverte serait, en fait, une *adaptation au jeu de Go*, ou *spécialisation pour le jeu de Go*, des connaissances générales de regroupement sur le monde réel.

L'intérieur et l'extérieur

Les propriétés du groupe sont réparties en deux catégories : **intérieur** (base de vie) et **extérieur** (amitié, inimitié, vacuité).

Correspondance avec le degré de conscience humain

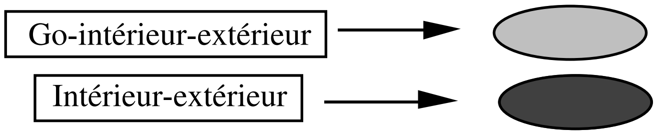


figure Correspondance-intérieur-extérieur

A propos de l'apprentissage de ce qu'est l'intérieur ou l'extérieur au Go, même remarque qu'au paragraphe précédent avec une différence : les joueurs de Go ont beaucoup de mal à expliquer comment distinguer pratiquement l'intérieur de l'extérieur; les connaissances correspondantes sont non conscientes, et même intuitives.

La synthèse

Les propriétés du groupe traduisent une **synthèse**. Synthèse à un premier niveau des interactions du groupe en propriété interactive et synthèse à un deuxième niveau des propriétés intérieures et extérieures en une unique propriété synthétique.

A posteriori, nous avons analysé comment nous avons effectué ces synthèses. Nous avons constaté que des nombres se retrouvaient constamment d'un attribut à l'autre : 2, 3 et 4. Pour présenter notre modèle, nous avons choisi de créer les constantes "**plusieurs**" et "**quelques**". Avec ces notations, les règles de synthèse expriment l'idée que si un groupe n'a que "quelques" possibilités suivant un point de vue, il est instable suivant ce point de vue. La synthèse exprime aussi que si un groupe a "plusieurs" possibilités suivant un point de vue, il est stable pour ce point de vue. Suivant la dépendance ou l'indépendance entre les éléments de la synthèse, les valeurs de stabilité sont pratiques ou théoriques.

<u>En théorie :</u>

	stabilité théorique
chaîne = f(liberté+)	3
base de vie = f(œil>)	2
IA = f(connexion*)	2
amitié = f(IA*)	3
vacuité = f(dilatation*)	3

Si les éléments sont indépendants, il est intuitif de penser que 3 représente la stabilité. Si l'adversaire supprime un élément, il en reste deux pour faire quelque chose. 2 est instable mais offre l'avantage de la symétrie (le concept de miai célèbre au Go), si l'adversaire prend un élément, on prend l'autre et on verra bien ce qui se passera. Transposé dans le cas où l'élément est la liberté, cela donne le shisho. 1 offre l'avantage de n'offrir qu'une seule possibilité et donc de ne pas réfléchir. Tout au long de notre modélisation, nous avons **théoriquement** essayé de suivre le principe qui dit que **3 représente la stabilité**. Ce principe permet d'éliminer les modèles où trop de calculs sont nécessaires.

<u>En pratique :</u>

La pratique pousse le nombre 3 dans l'instabilité. Même si deux libertés sont des éléments indépendants par définition (on ne peut pas supprimer 2 libertés en un coup), nous avons vu au niveau "chaîne" comment notre modèle est plus performant pratiquement avec un jeu de la chaîne à 4 libertés qu'avec 3. Au niveau "groupe", les éléments peuvent être dépendants (on peut tuer 2 yeux d'un coup, 2 ou 3 connexions d'un coup) et le seuil de stabilité d'un point de vue augmente sensiblement.

	stabilité pratique
chaîne = f(liberté+)	4
IA = f(connexion*)	4 ?
amitié = f(IA*)	4 ?
vacuité = f(dilatation*)	4 ?
inimitié = f(IE*)	4 ?
santé = f(élément*)	4 ?

4 représente la stabilité pratique. Cela permet d'obtenir de meilleurs résultats pratiques à court terme.

La similarité entre l'échelle des pierres et celle des groupes

Les propriétés interactives à **l'échelle des groupes reflètent** ce que nous voyons physiquement sur un goban à **l'échelle des pierres**.

L'image de gauche de la figure *Groupe-pierre-intérieur* est vue non seulement comme une photo prise à l'échelle des pierres mais aussi comme une photo prise à l'échelle des groupes. L'image de droite symbolise alors les propriétés intérieures au groupe (la base de vie de notre modèle).

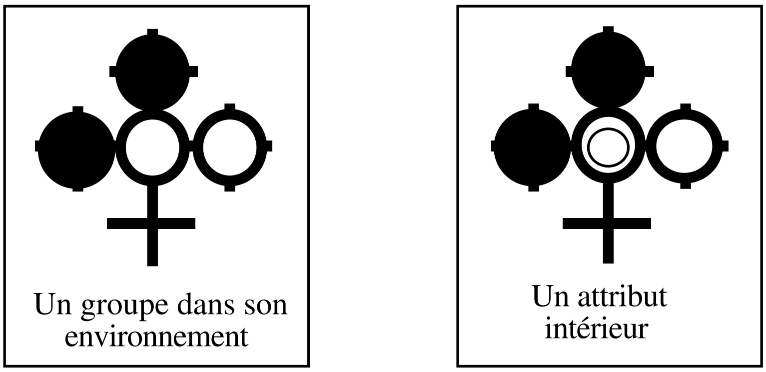


figure Groupe-pierre-intérieur

La figure *Groupe-interactions* symbolise alors les types d'interactions du groupe avec son environnement (amie, ennemie, vide).

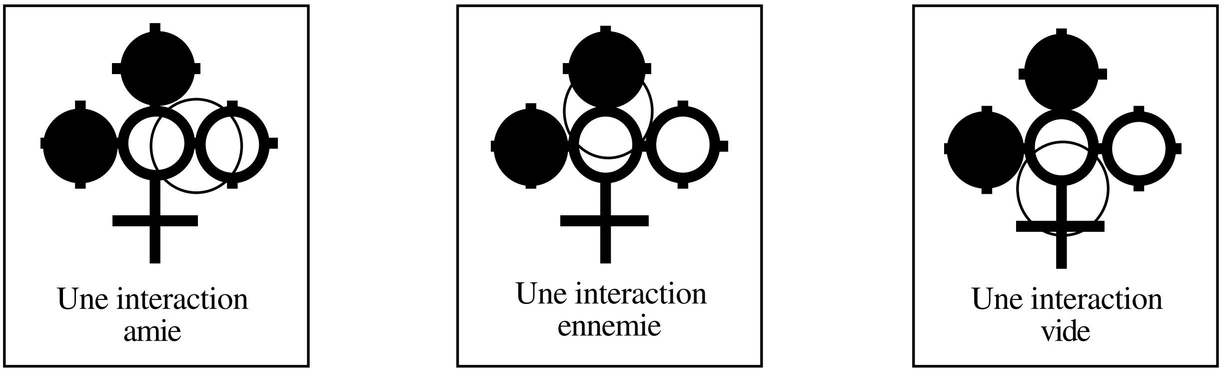


figure groupe-interactions

L'image de gauche de la figure *Groupe-synthèse* symbolise la synthèse des interactions d'un type d'interaction (amitié, inimitié, vacuité). L'image de droite symbolise la synthèse des propriétés intérieures et extérieures du groupe (la santé).

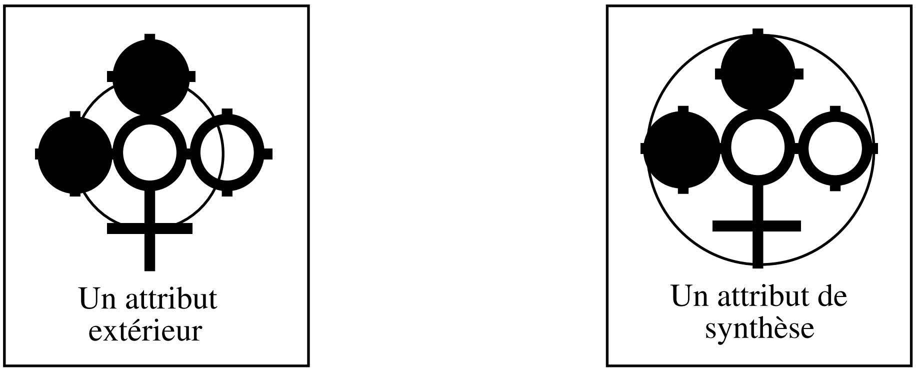


figure groupe-synthèse

Correspondance avec le degré de conscience humain

La faculté de manipuler des faits et des connaissances à différentes échelles de grandeur est une des facultés visuelles essentielles de l'être humain. Le joueur de Go, utilise cette capacité pour jouer au Go, mais il en est très peu conscient, ce que représente la figure *Correspondance-multi-échelle*.

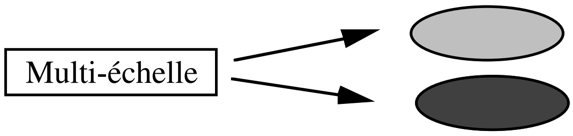


figure Correspondance-multi-échelle

La vie et la mort des groupes

Nous avons montré comment un groupe peut être **vivant** ou **mort.** La notion de vie et mort est ambiguë dans le jargon du Go. Certains joueurs appellent "mort" (respectivement "vivant") un groupe sans (respectivement avec) base de vie, d'autres appellent "mort" (respectivement "vivant") un groupe avec une santé négative (respectivement positive). La force de notre modèle sur les groupes avec la distinction intérieur-extérieur, est de **lever l'ambiguïté** avec les attributs base de vie, et santé qui peuvent être > ou <. **Nous avons explicité un modèle statique de la vie et de la mort des groupes, faisant intervenir l'interaction avec le voisinage du groupe.** Nous avons décrit la **catastrophe** qui se produit quand un groupe meurt.

Correspondance avec le degré de conscience humain

Selon le niveau du joueur, les connaissances sur la vie et la mort des groupes sont plus ou moins conscientes. En caricaturant, un joueur fort (5ème dan) possède ces connaissances sous forme plutôt non consciente. S'il fait l'effort de les conscientiser, ces connaissances seront éventuellement bien formalisées. Un joueur moyen (10ème kyu) possède ces connaissances sous forme consciente mais souvent mal formalisée. Un joueur faible (25ème kyu) ne possède pas ces connaissances du tout.

Les territoires

Introduction

Nous présentons dans ce paragraphe le concept de territoire. Ce concept est important car il permet d'évaluer une position en terme de points. Sa construction fait intervenir des concepts morphologiques. Les attributs d'un territoire sont assez simplistes actuellement.

La construction

Par la morphologie mathématique

La construction des territoires est un des points forts de notre modèle. Il faut se reporter au paragraphe Domaines Voisins et refaire le circuit Croissance Fractale -> Morphologie Mathématique Symbolique -> Numérique -> Multi-Echelle, pour comprendre cette construction. La croissance fractale nous a permis de faire une analogie entre les "fjords" de Mandelbrot et les territoires au Go. La morphologie mathématique classique nous a permis de créer des outils pour reconnaître les territoires sommairement. La morphologie mathématique floue nous a permis d'affiner cette reconnaissance à une échelle donnée.

Nous donnons des exemples de territoires reconnus par INDIGO au paragraphe Morphologie Mathématique et Logique Floue de la Partie 2 de ce document. En cours de jeu, nous avons fixé **le nombre de dilatations à 3 et le nombre d'érosions à 7**[1]. INDIGO pourrait reconnaître des territoires à plus grande échelle. Mais cela ne sert à rien car il ne sait pas défendre des territoires trop grands. Pour illustrer les concepts de territoire dans ce paragraphe, d'espace vide et de fraction aux paragraphes suivants, nous prenons pour exemple la position de la figure *TFV-exemple*.

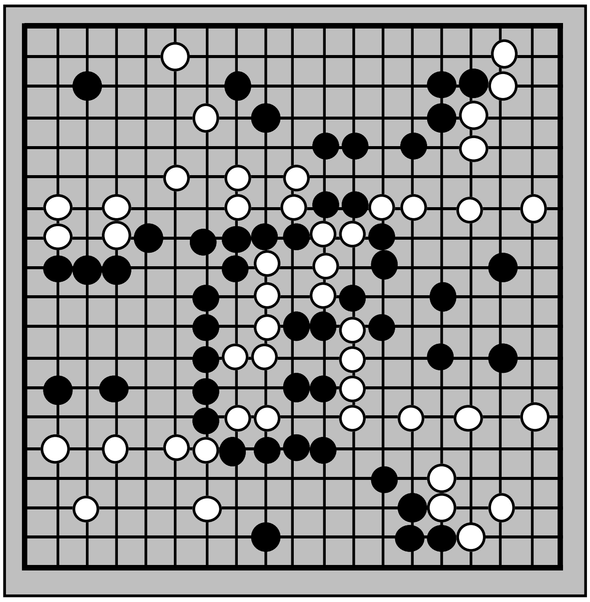


figure TVF-exemple

[1]Au paragraphe sur la morphologie mathématique et sur la logique floue, nous avons dit que le nombre d'érosions est égal à 1+n(n-1) où n est le nombre de dilatations.

La figure *Territoire-reconnu* montre les territoires reconnus à partir de l'exemple. Les ■ indiquent des intersections de territoire noir et les □ indiquent des intersections de territoire blanc.

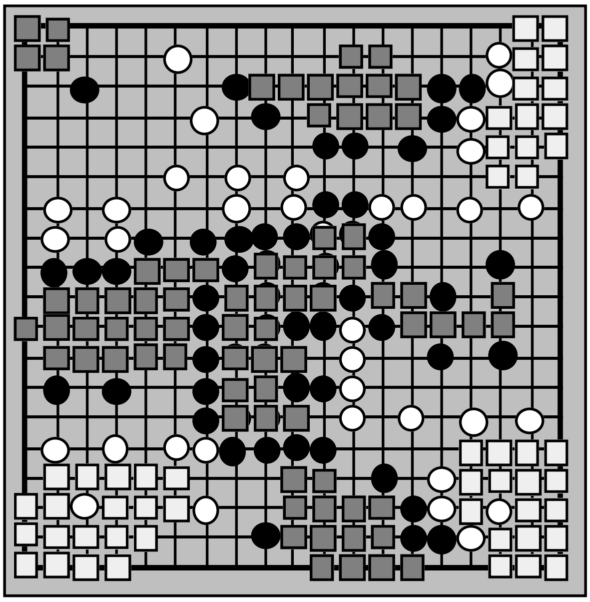


figure Territoire-reconnu

Le lecteur remarquera que tous les territoires sont reconnus par morphologie mathématique sauf le territoire noir central issu de la catastrophe du groupe blanc central.

Les attributs

Actuellement les actions possibles sur un territoire sont générales mais simplistes : érosion et dilatation. Nous aimerions modéliser les actions sur les territoires par analogie avec le modèle sur les groupes.

Un attribut intérieur

Cet attribut, analogue de l'attribut "base de vie" intérieur aux groupes, traduirait la possibilité d'envahir un territoire.

Un attribut inimitié

Cet attribut, analogue d'inimitié pour les groupes, traduirait la possibilité de réduire un territoire adverse en dilatant son propre territoire.

Un attribut vacuité

Cet attribut, analogue de vacuité pour les groupes, traduirait la possibilité de réduire ou dilater un territoire proche d'un espace vide.

Un attribut amitié

Cet attribut, analogue d'amitié pour les groupes, traduirait la possibilité de réduire ou dilater deux territoires de la même couleur.

Un attribut fermeture

Cet attribut, analogue d'"encerclement" pour les fractions, traduirait la possibilité de fermer ou dévaster un territoire. Cet attribut est hérité de l'attribut de séparation du concept de fraction[1]

Conclusion

La construction des territoires a fait appel à des **techniques de domaines voisins** au Go et à des analogies entre ces domaines. La qualification des territoires est encore embryonnaire avec les seules opérations de dilatation et d'érosion. En faisant une analogie avec notre modèle sur les groupes nous pensons spécialiser ces opérations et en trouver de nouvelles comme l'invasion, qui est une érosion par l'"intérieur". En faisant hériter la séparation du concept de fraction (cf. plus loin), nous pensons modéliser la fermeture du territoire et son inverse, l'ouverture. Le système INDIGO n'effectue actuellement aucun calcul arborescent sur cette description des territoires. En effectuant de la recherche arborescente, il pourrait améliorer son niveau.

Correspondance avec le degré de conscience humain

La construction des territoires dans INDIGO correspond chez le joueur humain a des connaissances visuelles sur la distinction entre l'intérieur et l'extérieur. Comme décrit dans la conclusion du niveau élémentaire sur les jeux topologiques ou morphologiques ou bien dans la conclusion sur les groupes, nous pensons que le joueur humain utilise des connaissances intuitives générales acquises dans le monde réel qu'il adapte au jeu de Go pour reconnaître les territoires. Ce fait est résumé par la figure *Correspondance-territoire*.

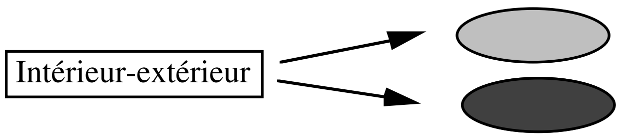


figure Correspondance-territoire

[1]Le concept de zone sera présenté plus loin comme un "ancêtre" du concept de territoire.

Les espaces vides

Introduction

Nous présentons dans ce paragraphe le concept de vide. La construction des espaces vides est analogue à celle des territoires (ouverture morphologique au lieu de fermeture morphologique). Le concept de vide est important car il permet de mesurer l'avancement de la partie, en particulier la fin pour passer.

Construction

A échelle 0, INDIGO considère une intersection incontrôlée (pas de territoire ni de groupe) comme appartenant à un espace vide. La figure *Vide-échelle-0* montre les intersections "vides" qui correspondent à la figure *TVF-exemple* vue précedemment.

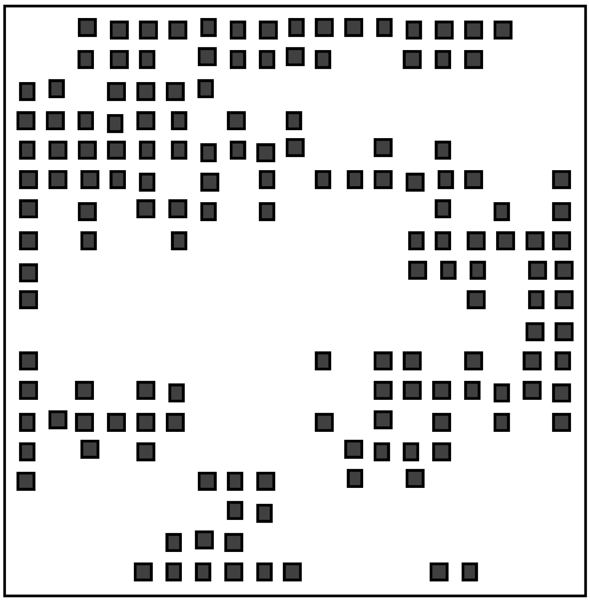


figure Vide-échelle-0

Cette figure est le "complémentaire" de la figure *Territoire-reconnu* au sens où un petit carré noir correspond exactement à une intersection vide de la figure *Territoire-reconnu*.

A échelle n, INDIGO utilise l'opérateur Y(n, m), issu de l'opérateur d'ouverture morphologique pour connaître les espaces vides.

Par exemple, la figure *Vide-échelle-1* montre les espaces vides à échelle 1 obtenu avec Y(1, 1).

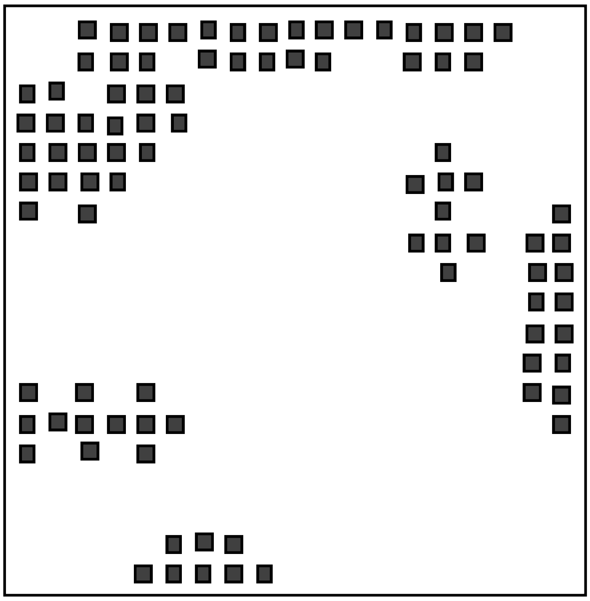


figure Vide-échelle-1

Et à échelle 2, obtenue avec Y(2, 2), il ne reste plus beaucoup d'espaces vides comme le montre la figure *Vide-échelle-2*.

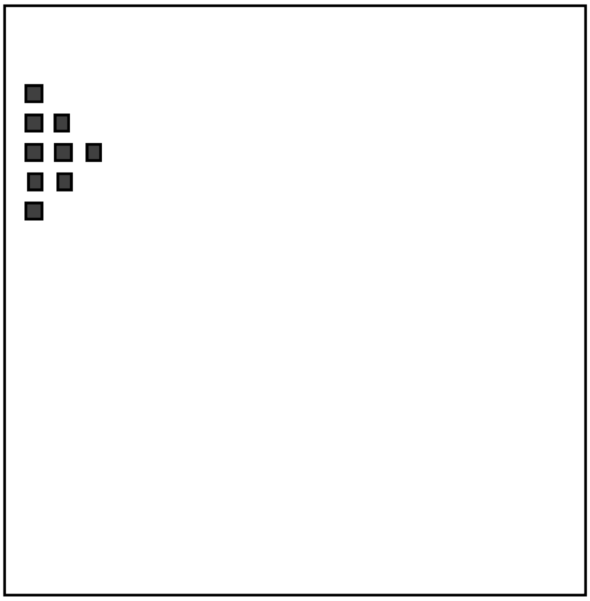


figure Vide-échelle-2

A échelle 3, il n'existe aucun espace vide.

Mesurer l'avancement de la partie

En observant la présence ou l'absence d'espaces vides aux différentes échelles, on peut mesurer l'avancement de la partie lorsque les objets sont indépendants.

Pas d'espace vide à l'échelle 0 : fin de partie au sens chinois :: tous les "dame" sont remplis.

Pas d'espace vide à échelle 1 : fin de partie au sens japonais : les derniers espaces vides restant sur le goban sont des "dame".

Pas d'espace vide à échelle x : fin du fuseki[1].

Espace vide à l'échelle maximale : tout début de partie.

Conclusion

L'espace vide à remplir est la base du jeu de Go. La reconnaissance des espaces vides pose les mêmes problèmes multi-échelle que celle des territoires. INDIGO mesure l'avancement de la partie avec les espaces vides. Les espaces vides sont utilisés pour choisir le coup du niveau global.

Correspondance avec le degré de conscience humain

La construction des espaces vides dans INDIGO correspond chez le joueur humain à des connaissances visuelles sur la distinction entre l'intérieur et l'extérieur, surtout en début et milieu de partie. Même remarque que pour les territoires. Ce fait est résumé par la figure *Correspondance-vide*.

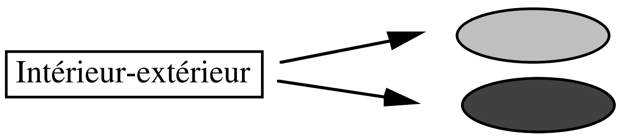


figure Correspondance-vide

[1]Le début de la partie au Go : environ vingt à trente coups.

Les fractions

Introduction

Nous montrons d'abord ce qu'est une fraction sur un exemple. Nous montrons ensuite comment une fraction est construite. Nous montrons comment **le concept de fraction permet de modéliser l'encerclement d'un groupe**.

Un exemple

La figure *Fraction-exemple* donne la position sur laquelle nous allons présenter le concept de fraction.

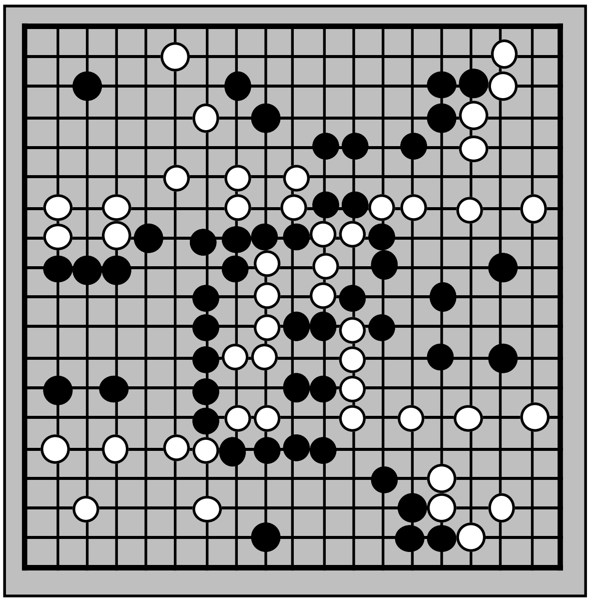


figure Fraction-exemple

Le goban de la figure *Fraction-exemple* est intuitivement interprété en terme de fraction **noire** de la manière représentée sur la figure *Fraction-noir* :

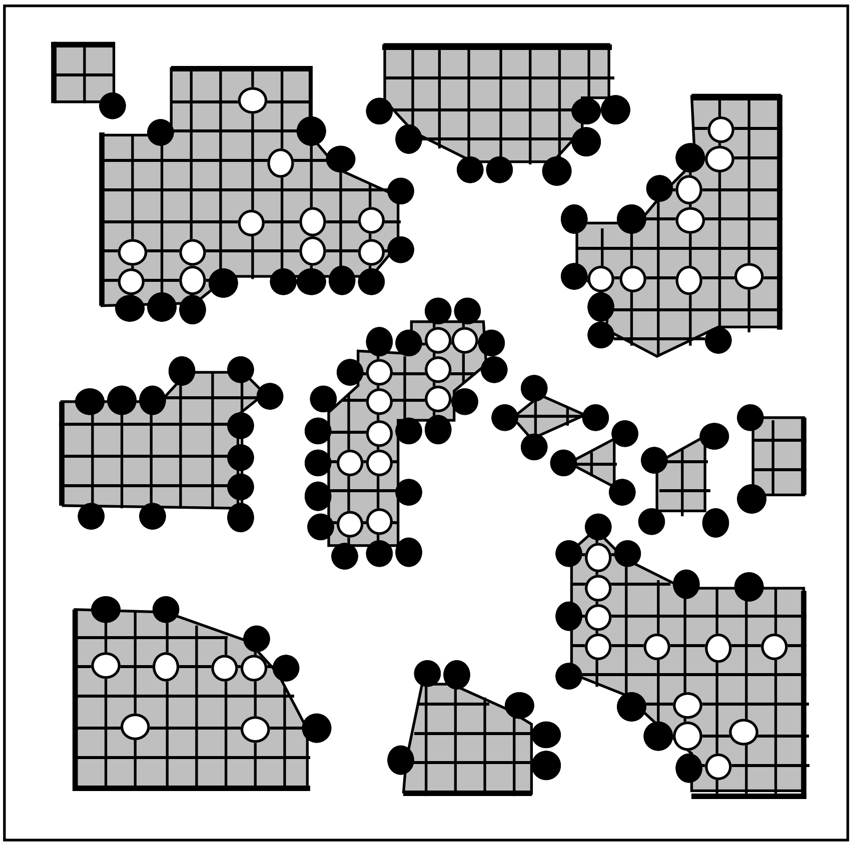


figure Fraction-noir

De même, le goban de la figure *Fraction-exemple* est intuitivement interprété en terme de fraction **blanche** de la manière représentée sur la figure *Fraction-blanc* :

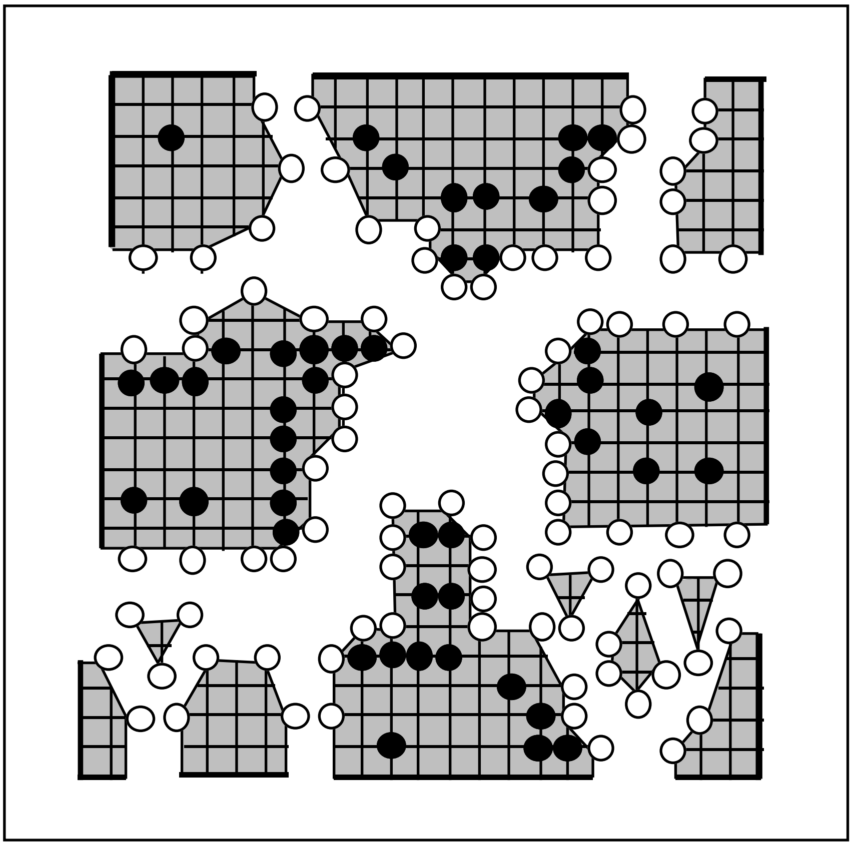


figure Fraction-blanc

Il faut noter que le goban peut être fractionné suivant le point de vue de Noir pour donner les fractions noires et celui de Blanc pour donner les fractions blanches.

Une fraction noire (resp. blanche) est limitée par des pierres noires (resp. blanche) et contient éventuellement un groupe - ou plusieurs - blanc (resp. noir) ou un territoire noir.

La construction

Pour construire les fractions du goban, nous avons utilisé une technique par itérations, analogue à celle utilisée pour construire les groupes. Une fraction est un ensemble maximal d'intersections limité par des séparations[1] >. Le lecteur pourra vérifier que l'on identifie par cette méthode les fractions noires et blanches des figures *Fraction-noir* et *Fraction-blanc* à partir de la position de la figure *Fraction-exemple*. INDIGO reconnaît les fractions de cette manière.

L'utilisation

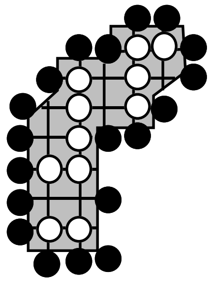
Nous montrons dans ce paragraphe comment le concept de fraction permet de modéliser l'**encerclement d'un groupe**.

Au départ d'une partie, il n'existe aucun groupe. Il existe virtuellement une fraction noire et une fraction blanche. Quand la partie avance, les pierres sont de plus en plus nombreuses et le goban est découpé en deux types de fractions, les blanches et les noires. La taille des groupes grossit et celle des fractions diminue. A tout moment de la partie, chaque groupe d'une couleur fait partie d'une fraction de l'autre couleur. Une fraction d'une couleur contient un ou plusieurs groupes de

[1]Les séparations sont définies au paragraphe sur le niveau intermédiaire.

l'autre couleur. Un groupe G contenu dans une fraction F, devient encerclé d'ordre n lorsque $D^n(G)$ contient F, où D est la dilatation morphologique.

Par exemple, le groupe blanc central de la figure *Fraction-groupe-encerclé* est encerclé d'ordre 1.



Fraction-groupe-encerclé

L'intégration dans notre modèle

Nous nous posons beaucoup de questions au sujet des fractions. Elles reflètent plusieurs choses.

La fraction du goban, concept dual du regroupement des intersections

Alors que les groupes sont construits de façon bottom-up[1], les fractions sont construites de façon top-down[2].. Lorsque la partie avance, la taille des fractions diminue et celle des groupes augmente. Au début de la partie, chaque groupe est inclus dans une fraction très grande et à la fin chaque groupe coïncide presque avec la fraction qui l'inclut.

Le triplet **(global, séparation, fraction) est dual du triplet (intersection, connexion, groupe)**.

La dualité (groupe, connexion) - (fraction, séparation) doit théoriquement nous pousser à créer un modèle de fractions dual de celui de groupe. Or nous n'avons pas pratiquement développé un modèle sur les fractions aussi complet que celui sur les groupes (avec qualification basée sur la synthèse et l'interaction). Ce serait trop lourd. Pratiquement, nous avons ajouté aux groupes un attribut nommé "encerclement" qui traduit la manière dont un groupe est encerclé dans sa fraction. Mais l'encerclement est un attribut interactif par essence et il est redondant avec les concepts de vacuité et d'inimitié... Que faire pratiquement ?

Les fractions et les territoires

Suivant la présence et l'états du (des) groupe(s) adverse(s) qui sont inclus dans la fraction et l'état des groupes qui la délimitent, une fraction peut être dans plusieurs états.

[1]les intersections sont associées par connexion.
[2]le goban est coupé en zones délimitées par des séparateurs.

<u>1 - S'il n' y a pas de groupe adverse à l'intérieur et si les groupes qui participent à la frontière ont une santé > , la fraction est un territoire</u>

La figure *Fraction-cas-1* montre des exemples de fractions dans le cas 1.

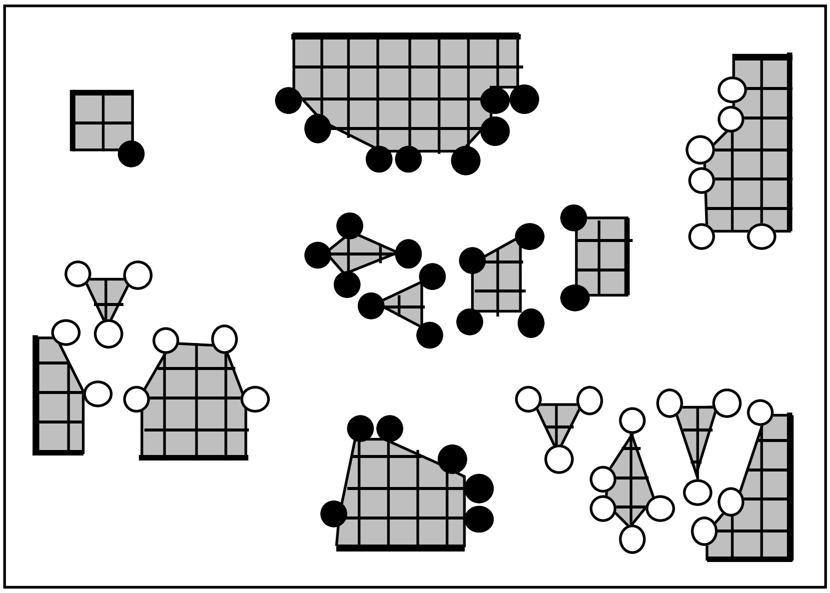


figure Fraction-cas-1

Un problème posé par ce type de fraction est clairement le fait que ces fractions sont trop fractionnées. Nous préférerions regrouper les fractions qui font partie du même territoire.

<u>2 - S'il n'y a pas de groupe adverse et si les groupes qui participent à la frontière ont une santé <, la fraction est un œil ou une forme morte.</u>

Ce cas n'existe malheureusement pas sur l'exemple que nous avons choisi pour illustrer le niveau itératif. Le lecteur imaginera que si le groupe blanc central mort avait un œil ou une forme morte, cet œil ou cette forme morte serait une fraction dont les frontières seraient constituées par le groupe mort lui-même.

<u>3 - Si un groupe adverse est présent et possède une santé >, la fraction est invisible et n'offre pas d'intérêt a priori.</u>

La figure *Fraction-cas-3* montre des exemples de fractions dans le cas 3.

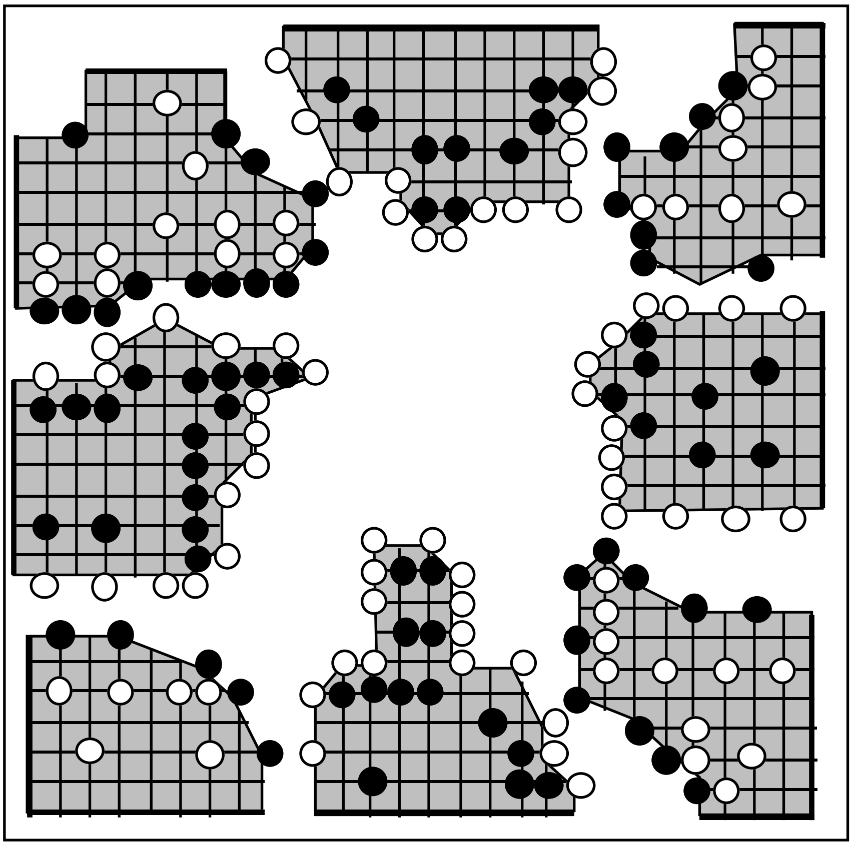


figure Fraction-cas-3

<u>4 - Si un groupe adverse est présent et possède une santé négative, la fraction est un territoire.</u>

La figure *Fraction-groupe-encerclé* présentée avant montre un exemple de fraction dans le cas 4.

<u>Pourquoi les joueurs de Go ne parlent pas des fractions.</u>

Dans le cas 1, on voit du territoire vierge. Dans le cas 2, on voit une forme morte. Dans le cas 3, on voit un groupe vivant. Dans le cas 4, on voit un territoire issu de la mort d'un groupe. Dans tous les cas les fractions sont cachées. Voilà pourquoi **les joueurs de Go ne parlent pas des fractions**.

En général, le groupe adverse ne possède ni une santé > ni une santé < mais il a une santé , une étude de la relation entre le groupe adverse et la fraction permet de dire si le groupe a une santé plutôt proche de > ou de <. C'est là l'intérêt du concept de **fraction**. Même si les joueurs de Go n'en parlent pas, **ils utilisent ce concept**. Nous pensons que beaucoup du talent d'un joueur de Go consiste en la finesse d'appréciation de la relation du groupe inclus et de sa fraction.

Les fractions et les groupes : l'encerclement est un attribut de la classe groupe dans INDIGO

Lors de la présentation du modèle sur les groupes nous n'avons pas parlé de l'encerclement pour alléger la présentation au lecteur. En réalité, l'encerclement est un attribut de plus de la classe groupe. Il a été défini précédemment (cf. paragrpahe sur l'utilisation des fractions). Comme les attributs interactifs, cet attribut donne une contrainte supplémentaire pour qu'un groupe ait une santé < et soit mort :

Pour refléter ce qui est implémenté dans INDIGO, il faut remplacer la règle :

Si base de vie< et amitié< et inimitié< et vacuité< alors santé<

par la règle :

Si base de vie< et amitié< et inimitié< et vacuité< et encerclement< alors santé<

Mesurer l'avancement du milieu de partie

Une fraction a une couleur, Blanc ou Noir. La caractéristique du milieu de partie est d'avoir beaucoup d'intersections où une fraction noire et une fraction blanche sont présentes. Cela traduit la dépendance qui existe entre les points de vue de Noir et de Blanc. Au début de partie, le fuseki, les fractions blanches et noires sont disjointes car peu de pierres ont été posées. En fin de partie, le yose, des fractions ont disparu sous l'effet de la domination de l'un des deux joueurs. Par contre, en milieu de partie, les fractions blanches et noires se recouvrent et cela traduit la dépendance entre Blanc et Noir.

Conclusion

Les fractions sont des parties du goban partiellement séparées les unes des autres. Pour les construire, on prend les ensembles d'intersections maximaux limités par des jeux de la séparation de l'adversaire >.

Il existe une **dualité (groupe, connexion) avec (fraction, séparation)**.

Les fractions sont **utilisées pour reconnaître l'encerclement des groupes**.

Les fractions sont un indicateur de l'avancement de la partie.

Correspondance avec le degré de conscience humain

L'analogie (groupe, connexion) avec (fraction, séparation), rend la correspondance entre notre modèle computationnel avec les connaissances du joueur humain, analogue à la correspondance faite en conclusion du paragraphe sur les groupes. Il existe des connaisances générales et intuitives de fraction chez l'être humain et il existe des Go-connaissances de Go-fraction chez le joueur de Go, qui sont des adaptations ou spécialisations des connaissances générales de fractions. La seule remarque qui diffère est le fait que la séparation est moins consciente que la connexion ne l'est. Ce fait est résumé par la figure *Correspondance-fraction-séparation* (pas d'ovale blanc) :

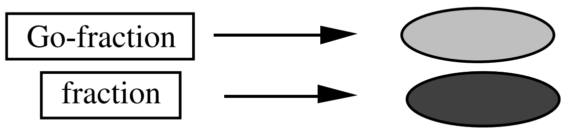


figure Correspondance-fraction-séparation

Les joueurs de Go ne parlent pas des fractions, ce concept est surtout **non conscient**.

Conclusion

<u>Le niveau itératif permet d'expliciter des concepts implicites pour l'homme, sinon d'identifier leur existence</u>

Les groupes : **regroupement, intérieur-extérieur, synthèse, similarité d'échelle.**

Les territoires : **intérieur-extérieur, multi-échelle**.

Les espaces vides : **multi-échelle**.

Les fractions : **séparation**.

La figure *Correspondance-niveau-itératif* résume les correspondances faites dans les conclusions de chaque niveau du modèle entre ce qui est présent dans le modèle cognitif d'INDIGO et les connaissances d'un joueur humain.

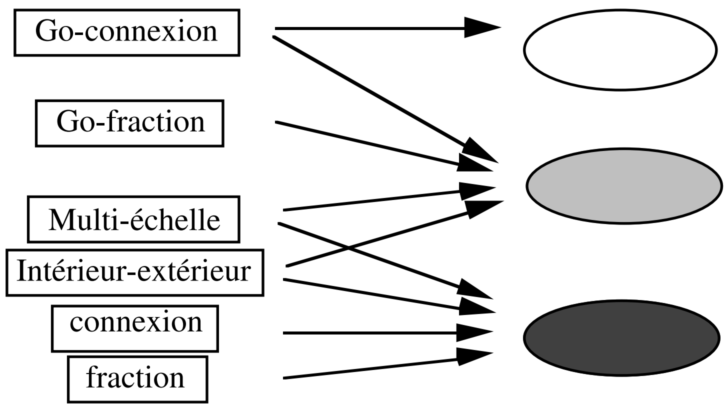


figure Correspondance-niveau-itératif

<u>La force de ce niveau est de contenir un modèle statique sur la vie et la mort des groupes qui tient compte du voisinage du groupe.</u>

A notre connaissance, cela n'a jamais donné lieu a des publications. Les programmes de tsumego comme Risiko de Thomas Wolf sont dynamiques mais ne tiennent pas compte du voisinage d'un groupe. Les auteurs des programmes de jeu de Go qui jouent une partie complète, utilisent peut-être ce type d'évaluation statique mais d'une part, ils ne le disent pas publiquement et d'autre part, le comportement de ces programmes ne le laissent pas penser.

<u>La force de ce niveau est aussi de contenir un algorithme de reconnaissance des territoires basé sur la combinaison d'opérateurs morphologiques.</u>

A notre connaissance, il n'existe aucun algorithme, explicite dans la littérature, de reconnaissance de territoires (par opposition à l'influence modélisée par tous les concepteurs de programmes de Go avec la dilatation morphologique).

Méthode

Pour élaborer ce niveau, notre connaissance sur le Go et sur les domaines "voisins" a été utile: **Théorie des jeux, Morphologie mathématique, Intelligence Artificielle Distribuée**.

Perspectives

Les conditions statiques (les coups engendrés et les états Gagné et Perdu) sur le jeu de la santé du groupe sont définis. Lorsque la puissance des machines que nous utilisons le permettra, INDIGO pourra faire des **calculs dynamiques** sur le jeu de la santé de chaque groupe. Le niveau de INDIGO augmentera sensiblement[1].

De plus ces calculs, un par groupe, sont distribués. Ils pourront être faits **en parallèle**, ce qui réduira le temps de réponse.

Bibliographie

[Boon 1991] - Mark Boon - Overzicht van de ontwikkeling van een Go spelend programma - Afstudeer scriptie informatica onder begeleiding van prof. J. Bergstra - 1991

[Bouzy 1994b] - B. Bouzy - Modélisation des groupes au jeu de Go - Rapport interne du LAFORIA - Septembre 1994

[Cazenave 1994] - T. Cazenave - Un système apprenant à jouer au Go - 2ème Rencontres Nationales des Jeunes Chercheurs en IA - Marseille 1994

[Chen 1989] - K. Chen - Group identification in computer go - Heuristic programming in Artificial Intelligence 1 - D.N.L. Levy & Beal D.F. (éds) Ellis-Horwood Prenctice-Hall 1989

[Fotland 1992] - D. Fotland - Many faces of go, documentation and playing algorithm - 1992

[Müeller 1993] - M. Müeller - Game Theories and Computer Go - Cannes Workshop Computer Go - 1993

[1]Tant l'information apportée par des calculs est grande, nous pensons qu'il gagnerait 5 pierres, mais il nous faudrait des machines 100 fois plus puissante !

Introduction

Dans ce paragraphe, nous présentons le niveau "global". Le but du niveau global est d'utiliser la description du niveau "itératif" avec les groupes, les territoires et les espaces vides pour **calculer le score** et **jouer LE coup**. La caractéristique du niveau global est l'aspect **multi-critère**.

Le score

Calculer le score est relativement simple connaissant les informations sur les territoires et les groupes et moyennant quelques hypothèses. Le score est calculé en règle chinoise: un point pour une couleur si l'intersection est contrôlée par cette couleur (occupée par un groupe ou un territoire de cette couleur). Si une chaîne ou un groupe est instable (*) elle ou il n'intervient pas dans le décompte mais l'incertitude augmente de la valeur de la chaîne ou du groupe. Si un groupe est incertain ( ) , il compte pour sa valeur dans le score.

Le score est utilisé pour savoir qui gagne. La partie s'arrête lorsque les deux joueurs passent. INDIGO passe lorsque toutes les intersections du goban sont contrôlées par une couleur. INDIGO n'utilise pas le score pour décider de sa stratégie. En outre, INDIGO ne possède pas de stratégies de type sécuritaire ou de type risqué, à utiliser en cas de grosse avance ou de gros retard.

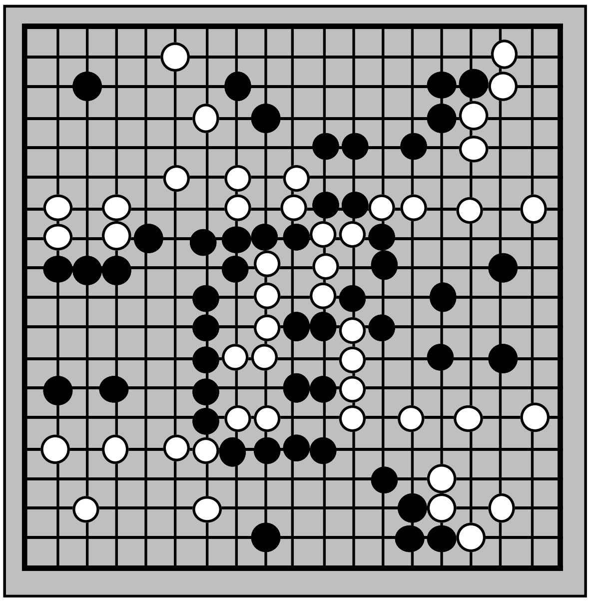


figure TVF-exemple

Sur la position de la figure *TVF-exemple* INDIGO voit les territoires indiqués par la figure *Territoire-reconnu* .

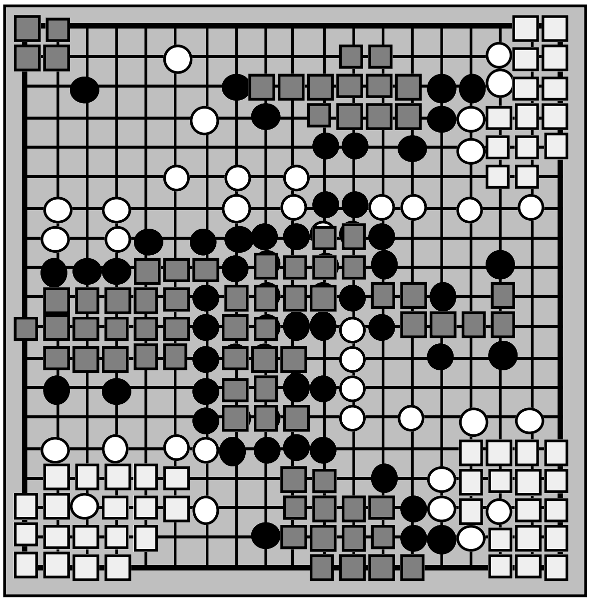


figure Territoire-reconnu

La figure *Territoire-reconnu* donne :

Territoires noirs 4 + 12 + 19 + 20 + 14 + 7 = 76
Groupes noirs 1 + 10 + 8 + 16 + 13 = 48
Total noir **124**
Territoires blancs 14 + 19 + 17 = 50
Groupes blancs 11 + 6 + 11 + 8 = 36
Total blanc **86**

Selon INDIGO, Noir gagne de 38 points. Cette appréciation est bonne mais statique. Le premier qui joue sur cette position peut gagner gros comme nous allons le voir.

Choisir LE coup

INDIGO choisit le coup en fonction de 3 types d'objets: les groupes, les territoires, les espaces vides. Trouver la valeur du coup joué pose deux problèmes : premièrement, comment affecter une valeur à un coup associé à un objectif donné: effectuer une action sur un groupe, sur un territoire ou sur un espace vide ? Deuxièmement, comment faire la somme de ces valeurs pour obtenir une valeur globale ?

La valeur d'un coup associé à un type d'objet donné

Premièrement, il est nécessaire de préciser que les objets stables (groupes vivants, territoires fermés) ne sont pas pris en compte. La valeur d'un coup associé à des objets stables est nulle. INDIGO joue pour stabiliser les objets instables. INDIGO ne joue pas de menaces sur des objets stables dans l'espoir de les déstabiliser ou de garder l'initiative.

Les groupes

Sur la position *TVF-exemple*, INDIGO reconnaît et qualifie les groupes avec leur santé comme indiqué par la figure *Global-groupe*.

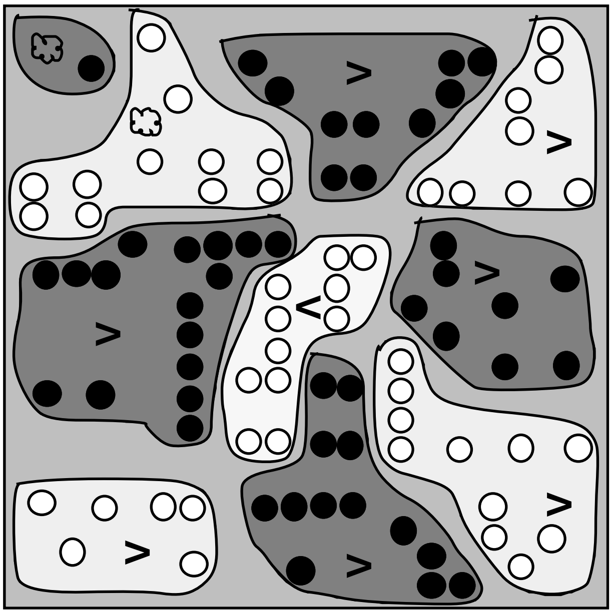


figure Global-groupe

A priori, la valeur d'un coup sur un groupe est proportionnelle à la taille du groupe. Plus un groupe est gros, plus il est intéressant de le stabiliser. Les paragraphes ci-dessous discutent de la valeur d'ordre 0 du coup sur le groupe (on ne tient pas compte des effets sur le voisinage) et de la valeur d'ordre 1 du coup sur le groupe (on tient compte des effets sur le voisinage proche).

La valeur d'ordre 0

La valeur d'ordre 0 d'un coup sur un groupe est proportionnelle à la taille du groupe. Il est difficile d'avoir une idée différente. Par contre, on peut l'affiner en observant qu'un groupe instable peut être * ou .

Si un groupe est dans un état *, l'effet du coup sur le groupe est directement prévisible. Si on joue on gagne le groupe, si l'adversaire joue, c'est lui qui gagne le groupe. Si des groupes sont dans un état , la valeur du coup est floue, elle possède une valeur moyenne égale à la taille du groupe avec un écart-type non nul. Ce genre de coup est parfois payant, parfois inefficace. Il est difficile de trouver des règles a priori qui permettent de trancher. INDIGO ne fait pas de différence entre les groupes * et les groupes .

Sur la position de la figure *TVF-exemple*, le niveau global d'INDIGO ne s'intéresse qu'aux deux groupes en haut à gauche. Les autres groupes ont une santé > et participent donc pour zéro à l'évaluation au niveau global. Les deux groupes n'engendrent pas de coups d'amitié (de connexion). Ils engendrent des coups de vacuité (de dilatation) et d'inimitié (d'opposition) qui ont une valeur égale à la taille des groupes.

La valeur d'ordre 1

La valeur d'ordre 1 d'un coup sur un groupe tient compte des effets sur les objets voisins. En effet, tuer un groupe peut rapporter plus que le groupe lui-même: stabilisation des groupes voisins, disparition des territoires voisins. Dans INDIGO, l'attribut inimitié d'un groupe tient compte des groupes ennemis voisins et permet d'avoir la valeur d'ordre 1. Mais cela ne permet pas de connaître les réactions en cascade possibles.

Les territoires

A priori, la valeur d'un coup sur un territoire est proportionnelle à la taille du territoire. En deuxième approximation , la valeur d'un coup sur un territoire est égale à la différence de taille du territoire si l'on joue en premier ou si l'adversaire joue en premier. Actuellement, INDIGO utilise la valeur a priori. Nous aimerions implémenter dans INDIGO la valeur d'un coup par différence. Une des faiblesses d'INDIGO est sa mauvaise évaluation pour affecter une valeur à un coup de territoire.

Les espaces vides

La valeur d'un coup sur un espace vide est proportionnel à la taille de l'espace vide et à son échelle de reconnaissance. En effet, en début de partie, il est important d'occuper le goban à grande échelle. Occuper un espace vide peut être plus important que de faire vivre un groupe (mais petit alors!). En fin de partie, certains espaces vides n'ont pas d'importance, ils sont neutres.

La valeur d'un coup global ou comment faire la somme ?

Pour un type d'objet donné, la valeur d'un coup est la somme des valeurs des coups conseillés par des objets de ce type. La valeur d'un coup conseillé par un objet de type donné est proportionnelle à la taille de l'objet, le coefficient de proportionnalité dépend du type de l'objet. Le coup joué pour un type donné est le coup qui a la valeur la plus grande.

Sur la position de la figure *TVF-exemple*, les coups qui sont à la fois des coups de dilatation pour les deux groupes ont une valeur égale à la somme de la taille de ces deux groupes : 5+11 = 16, les coups qui ne sont des coups de dilatation que pour un seul des deux groupes ont une valeur égale à la taille de ce groupe: 5 ou 11.

Pour choisir, on peut utiliser une méthode **lexicale** ou **compensatoire** [Pinson 1987]. Mais la décision au niveau global souffre des approximations faites aux niveaux du dessous.

Une méthode lexicale

La méthode lexicale suivant les types d'objet choisit d'abord suivant les groupes, puis suivant les territoires, puis suivant les espaces vides:

Choisir le coup parmi les groupes.
Sinon, choisir le coup parmi les territoires.
Sinon, choisir le coup parmi les espaces vides.

Un problème posé par cette méthode est que le programme joue pour stabiliser tous ses groupes, mêmes quand ceux-ci sont petits, avant de faire grossir des territoires ou d'occuper les coins vides!

Une méthode compensatoire

Une première méthode

Une première méthode compensatoire consiste à dire que tous les objets sont du même type. Un problème posé par cette méthode est que le programme occupe bien les espaces vides au début de la partie mais qu'après il préfère fermer des territoires que de tuer des groupes. Il se fait envahir ses territoires à grande échelle et ferme ses territoires autour de l'envahisseur sans le tuer.

Une amélioration

On affecte un poids différent à chaque type d'objet : Pg aux groupes, Pt aux territoires, Pv aux espaces vides. Le problème est que si on met Pg suffisamment grand, pour qu'il stabilise ses groupes avant les territoires et les espaces vides, on retombe dans la méthode lexicale. Il faudrait adapter les poids en fonction de l'avancement de la partie. Aucune méthode n'est donc vraiment satisfaisante. Mais nous avons choisi cette dernière méthode faute de mieux.

Sur la position *TVF-exemple* , INDIGO donne la même valeur (16) au coup ① et aux coups A de la figure *Global-coup* : la somme des tailles des groupes . Entre les coups de même valeur, INDIGO tire au sort. Par exemple, INDIGO jouera le coup ① .

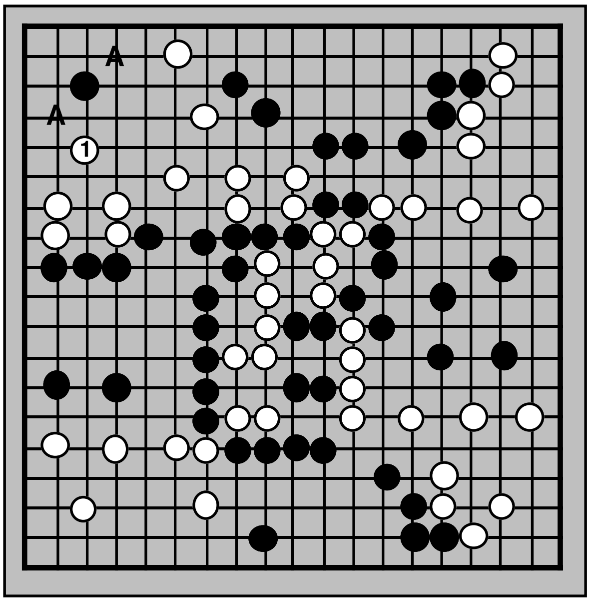


figure Global-coup

Ce coup est très bon. Les deux coups A aussi. Le joueur de Go remarquera que la valeur du coup est nettement supérieure à la valeur estimée par INDIGO (2 fois 16 = 32). Si Blanc tue le coin noir, il marque ≈ 55 points dans le quart Nord-ouest du goban (groupe et territoire confondus) et si Noir tue le groupe blanc il marque 55 points au même endroit. Le joueur de Go estimera donc la valeur à ≈ 100 points. Nous avons pensé donner la taille de la fraction dans laquelle est inclus le groupe au lieu de donner la taille du groupe comme valeur à un coup sur un groupe. Dans ce cas, cela donne une valeur plus proche : 50 + 32 ≈ 80 points. Mais la taille de la fraction est un indicateur trop sensible qui donne des résultats peu fiables. Nous avons préféré garder la taille du groupe comme valeur du coup associé à un groupe. Une perspective serait d'utiliser la valeur donnée par les fractions pour connaître une évaluation risquée et optimiste du coup et d'utiliser la valeur donnée par la taille des groupes pour connaître une évaluation sécuritaire et pessimiste du coup, et ainsi se rapprocher de l'algorithme B* de Berliner.

Les approximations faites aux niveaux inférieurs sont visibles au niveau global

Les recherches arborescentes des niveaux "zéro" et "élémentaire" sont faites avec la procédure alpha-bêta. Pour un jeu donné dans l'état *, cette procédure remonte UN coup pour Blanc et UN coup pour Noir. Quand on fait la somme au niveau global de tous les coups en fonction de tous les jeux *, il faudrait connaître, pour chaque jeu, TOUS les coups pour Blanc et TOUS les coups pour Noir. **Utiliser alpha-bêta au lieu de Minimax aux niveaux inférieurs devient un inconvénient au niveau global**.

Conclusion

Les connaissances stratégiques du niveau global

Au départ, ce niveau devait effectuer des choix sur des objets qui paraissaient hétérogènes : groupes, territoires et espaces vides. Notre travail a consisté à dégager des points communs entre ces objets pour que le processus de décision global soit facilité. Finalement, le niveau global est simple: il additionne et s'intéresse à ce qui est gros par la taille et instable.

Intelligence distribuée

Il est intéressant de remarquer que les formes globales jouées dans les parties d'INDIGO sont bonnes. La connaissance d'INDIGO est surtout locale et attachée aux objets du niveau itératif. **L'intelligence est répartie et distribuée sur chacune des entités locales existantes sur le goban.**

Niveau statique

Le niveau global est **statique** - comme le niveau itératif. Ce niveau souffre de **l'effet horizon avec un horizon 0** ! En effet sur la position de la figure *TVF-exemple*, INDIGO joue ① . Sans savoir si les groupes vont mourir ou vivre. Une séquence possible jouée par INDIGO contre lui-même serait par exemple celle de la figure *Global-séquence*.

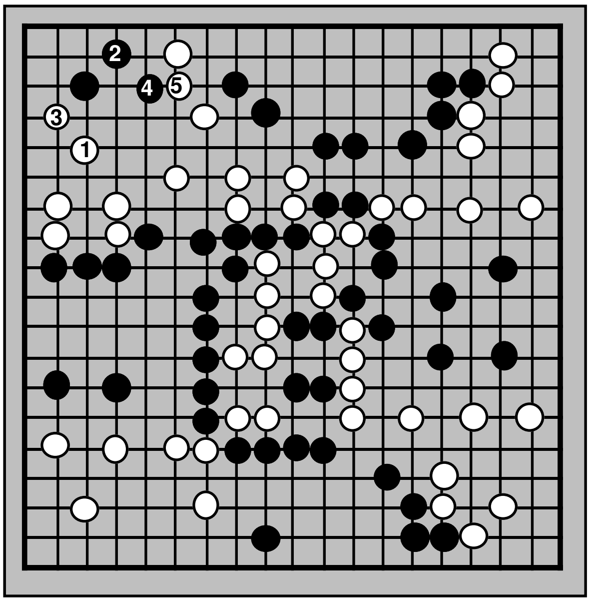


figure Global-séquence

INDIGO justifie les coups de 1 à 5 par la recherche de stabilisation de groupes .

<u>Correspondance avec le degré de conscience humain</u>

Les connaissances stratégiques d'un joueur humain sont conscientes comme le représente la figure *Correspondance-stratégie*.

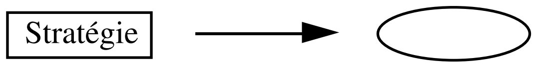


figure Correspondance-stratégie

Par contre, un joueur humain utilise une perception globale qui elle est intuitive et très difficilement conscientisable comme le représente la figure *Correspondance-global*.

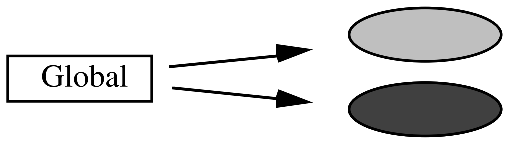


figure Correspondance-global

Les termes employés par les joueurs de Go (par exemple, le terme "*stratégie globale*" qui est un pléonasme) manifestent une confusion entre les concepts de stratégie et de globalité. Nous pensons que la perception du joueur est multi-échelle, locale ou globale, et que cette perception est ensuite utilisée par la stratégie qui, elle, est nécessairement globale. Stratégie implique global mais pas l'inverse.

Bibliographie

[Pinson 1987] - S. Pinson - Méta-modèles et heuristiques de jugement : Le système CREDEX. Application à l'évaluation du risque de crédit en entreprise. - Thèse de doctorat de l'Université Paris 6 - 1987

L'INCRÉMENTALITÉ

INDIGO peut fonctionner en mode **absolu** et en mode **incrémental**.

Le mode absolu

INDIGO procède par niveau en partant de celui du bas. INDIGO calcule d'abord tous les jeux de la chaîne, pour chaque chaîne du goban. Ensuite, il passe au niveau intermédiaire, où il calcule tous les jeux de ce niveau. Ensuite, il construit les groupes, au moyen des connexions, et valorise dans l'ordre : les amitiés, les territoires, les bases de vie, les encerclements, les vacuités, les inimitiés et les santés de chaque groupe. Si une santé est <, il crée une catastrophe, et ainsi de suite. Lorsqu'aucune catastrophe ne se produit il s'arrête et calcule les espaces vides. Enfin, INDIGO décide du coup global et le joue.

Le temps total pour jouer un coup (une interprétation du goban en mode absolu plus le choix du coup global) dépend de la taille du goban et du nombre d'objets à reconnaître et à qualifier. Sur une position de milieu de partie sur 19-19, au moment où beaucoup de groupes sont instables, INDIGO peut mettre 10 minutes pour jouer un coup en mode absolu sur une Sun SparcStation 10. Ce qui est intolérable.

Par quel moyen, relativement simple, nous avons pu diviser ce temps de réponse par 10 ?

Le mode incrémental

Introduction

En moyenne, un joueur de Go joue un coup en 30 secondes dans une partie normale[1]. Par contre, en partie rapide, la moyenne baisse à 5 secondes. Le temps est un facteur crucial. Une des facultés principales du joueur de Go, qui expliquerait la rapidité des joueurs humains, est de ne recalculer que ce qui est nécessaire après qu'une pierre est posée sur le goban. Souvent un coup, joué à un bout du goban, ne modifie pas l'état conceptuel à un autre bout. Mais cette règle n'est pas absolue. La règle du jeu nous le prouve. Une pierre posée sur le goban peut entraîner la capture physique d'une pierre adverse très loin de la pose initiale. Dans le jeu de la chaîne, le résultat d'un jeu peut dépendre d'un shisho qui traverse le goban.

Alors comment faire pour qu'un système jouant au Go joue à la même vitesse qu'un joueur humain et reproduise un semblant de fonctionnement incrémental ?

Nous avons suivi un principe simple. Nous avons profité de la large réification faite dans notre programme. Tout ou presque est objet. Les jeux sont des objets, les groupes et les attributs des groupes sont des objets. Tous les objets posés sur le goban font partie de la classe très générale Objet-posé.

La classe Objet-posé

La source

Un objet posé possède un attribut 'source' : l'ensemble des intersections sur lesquelles est posé l'objet. Par exemple, pour une chaîne de pierres, la source contient exactement les intersections où se trouve une pierre de la chaîne.

[1]En supposant qu'une partie dure une heure et que 120 coups sont joués par joueur.

L'empreinte

Un objet posé possède un attribut 'empreinte' : l'ensemble des intersections dont l'existence et l'état de l'objet dépendent. Cet ensemble s'appelle l'empreinte de l'objet posé. Par exemple, pour une chaîne de pierre, son empreinte est égale au lieu plus les intersections voisines.

Le principe

Si un coup est joué sur une intersection, INDIGO détruit tous les objets posés dont l'empreinte rencontre l'intersection. Lorsque l'on doit recalculer les objets posés d'un type donné sur une position, on ne recalcule que là où aucun objet posé de ce type n'existe.

L'avantage

Avec ce principe, le problème de l'incrémentalité devient un problème de spécification d'empreinte pour chaque type d'objet posé.

La spécification de l'empreinte

Pour certains objets, la spécification est simple, pour d'autres, elle est plus complexe. Tout dépend du niveau d'incrémentalité et de la fiabilité des calculs que l'on veut obtenir. Si on spécifie finement l'empreinte, on obtiendra une bonne incrémentalité avec une bonne fiabilité. Si on spécifie une empreinte trop petite, INDIGO ne détruira pas des objets posés qu'il aurait dû détruire, il ne les recalculera pas alors qu'il devrait le faire : les résultats seront faux. Si on spécifie une empreinte trop grande, INDIGO détruira et recalculera beaucoup d'objets posés mais obtiendra des résultats justes. Nous avons préféré privilégier la fiabilité des résultats en choisissant des empreintes suffisamment grandes.

Les spécifications de l'empreinte d'un objet posé statique sont assez simples: elles reposent sur le principe de dilater suffisamment de fois l'ensemble des intersections sur lequel repose l'objet. Quand un objet posé est dynamique, un jeu par exemple, on fusionne les empreintes calculées à chaque coup du calcul.

Pour un groupe, l'empreinte est la réunion des empreintes des connexions > et des chaînes et jeux de la chaîne associés. Pour une interaction, l'empreinte est égale à la réunion des empreintes des groupes. Pour un attribut interactif, l'empreinte est égale à la réunion des empreintes des interactions. Pour une santé, l'empreinte est la réunion de l'empreinte du groupe et des empreintes des attributs du groupe. Le lecteur remarquera que la santé d'un groupe dépend donc des intersections sur lesquelles sont posés les groupes voisins du groupe.

Spécifier correctement l'empreinte d'un objet posé d'une certain classe est plus complexe que de spécifier cette classe qui est déjà chaotique. Donc la spécification des empreintes est une affaire difficile qui ne peut se valider qu'expérimentalement. Pour le faire, le programmeur rencontre un objet bizarre : la bogue d'incrémentalité.

Définition d'une bogue d'incrémentalité

Position absolue

Nous appelons position absolue la dernière position calculée en mode absolu.

Position de la bogue

Nous appelons position de la bogue, la position où la bogue est apparente : le produit de l'exécution du programme n'est pas correct, il ne correspond plus à ce que le programmeur attendait, une cohérence a été perdue.

Bogue d'incrémentalité

Pour caractériser une bogue d'incrémentalité, nous partons d'une position absolue et nous jouons un coup, nous interprétons à nouveau incrémentalement, nous jouons un coup, etc...Si au bout de n coups joués, le résultat d'une interprétation incrémentale est différent de celui qu'aurait donné une interprétation absolue, nous disons que nous avons détecté une bogue d'incrémentalité d'ordre n.

Détection de la bogue d'incrémentalité

Détecter le type de la bogue

Chronologiquement, le programmeur détecte un résultat anormal : erreur d'interprétation de la position, mauvais coup, plantage, etc... Si aucune idée sur la cause de la bogue ne lui vient, il teste le type de la bogue : est-elle incrémentale ou non ? Il faut noter que les parties jouées utilisent l'interprétation incrémentale, donc il est assez fréquent de tomber sur une bogue incrémentale. Pour le savoir, on se place dans la position antérieure à celle où la bogue est apparue, on interprète de façon absolue, on joue le coup, on interprète à nouveau incrémentalement et on compare ce résultat à celui d'un interprétation absolue de la position. Si l'interprétation absolue est correcte et l'interprétation incrémentale est incorrecte, la bogue est classifiée incrémentale.

Détecter l'ordre de la bogue

Trouver l'ordre d'une bogue d'incrémentalité est important pour connaître un indice supplémentaire pour déterminer la cause de la bogue. Il faut repartir d'une position antérieure de m coups à celle où la bogue apparaît, comparer les résultats avec ceux de la partie (où l'on a fait le test avec m = la longueur de la partie); si le résultat est identique, il faut recommencer avec m plus petit; si le résultat est différent, il faut recommencer avec m plus grand. Par dichotomie, on obtient l'ordre de la bogue. Le coup qui a été joué à ce moment là peut être un indice pour cerner la cause de la bogue.

Le résultat

Finalement, la gestion incrémentale des objets posés sur le goban nous a permis de diviser par 10 le temps de réponse d'INDIGO. En mode incrémental, le temps de réponse ne dépend presque plus de la taille du goban. Il reste seulement sensible au nombre d'objets à reconnaître et au lieu du dernier coup à qualifier. **En mode incrémental, INDIGO joue un coup sur 19-19 en moins d'une minute (de 10'' à 1') sur une Sun SparcStation 10**.

Correspondance avec le degré de conscience humain

Les connaissances sur l'incrémentalité d'un joueur humain sont non conscientes, voire intuitives, comme le représente la figure *Correspondance-incrémentalité*.

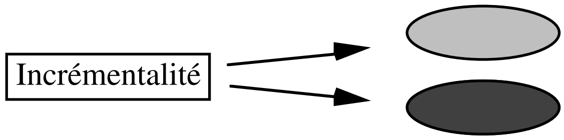


figure Correspondance-incrémentalité

Selon nous, c'est une des forces du système cognitif humain de gérer l'incrémentalité. Bien sûr, les joueurs de Go font aussi beaucoup d'erreurs causées par l'absence d'une réévaluation au bon moment. L'étude de l'incrémentalité est fondamentale pour améliorer les systèmes artificiels en général.

LA TAXONOMIE DES CLASSES

Nous donnons ci-dessous la taxonomie des classes dans INDIGO. Une tabulation signifie l'inverse de 'est-un'.

```
Objet
  Liste
  Ensemble-intersection
  Go-objet
    Partie
    Position
    Goban
    Intersection
    Règle
      Règle-topologique
      Règle-opposition
      Règle-dilatation
      Règle-vide
      Règle-point
      Règle-œil
      Règle-poison
        Règle-chaîne
        Règle-fraction
    Objet-posé
      Règle-reconnue
        Règle-reconnue-topologique
        Règle-reconnue-opposition
        Règle-reconnue-dilatation
        Règle-reconnue-vide
        Règle-reconnue-point
        Règle-reconnue-œil
        Règle-reconnue-poison
```

```
    Règle-reconnue-chaîne
    Règle-reconnue-fraction
Jeu
  Jeu-simple
    Jeu-intersection
    Jeu-chaîne
  Jeu-connexion
  Jeu-séparation
  Jeu-opposition
  Jeu-vide
  Jeu-point
  Jeu-séparation-territoire
  Jeu-œil
Chaîne
Groupe
Attribut-groupe
  Attribut-intérieur
    Base-de-vie
  Attribut-extérieur
    Encerclement
    Amitié
    Vacuité
    Inimitié
  Attribut-synthétique
    Santé
Fraction
Territoire
Vide
Global
```

CONCLUSION

Résumé du modèle INDIGO

MOOD INDIGO (My Object Oriented Design Is Now Designed In Good Objects) **reconnaît des objets** sur le goban, répartis en une cinquantaine de classes.

Il utilise le concept de jeu. U**n jeu est associé à une action sur un objet** reconnu sur le goban. **>** : le jeu est gagné même si l'adversaire commence, ***** : le premier qui joue gagne, **<** : le jeu est perdu même si on commence, **0** : le premier qui joue perd (impasse).

INDIGO est constitué par des **niveaux** d'abstraction croissante : le niveau **zéro**, le niveau **élémentaire**, le niveau **itératif**, le niveau **global**.

Le niveau zéro ou niveau chaîne est la base tactique d'un modèle sur le Go. Il est dynamique. Trouver la complexité adaptée de ce niveau pour qu'il s'intègre efficacement dans le reste du modèle est fondamental. **En théorie**, un seuil de stabilité des chaînes à **3 libertés** paraît correct mais **en pratique**, un seuil à **4 libertés** donne de meilleurs résultats.

Le niveau élémentaire regroupe une collection d'une **dizaine jeux élémentaires** qui doivent fournir un résultat précis, complet et rapide pour les niveaux supérieurs. Ces jeux utilisent un **demi-millier de règles** dont la particularité est de contenir des **patterns** dans leur partie gauche. L'état actuel de ce niveau a été difficile à obtenir à cause de l'hétérogénéité des concepts qu'il contient. Les concepts cruciaux et généraux, cachés derrière les concepts spécialisés, ont été difficiles à découvrir.

Le niveau itératif ou niveau groupe est le plus riche. Il contient **un modèle statique sur la vie et la mort des groupes qui tient compte du voisinage des groupes.** Le comportement d'un groupe est partagé en un comportement interne et un comportement externe. La reconnaissance des territoires a utilisé la **fermeture morphologique** des groupes. Ce niveau contient des concepts tels que : **regroupement, fraction, intérieur-extérieur, synthèse, multi-échelle.** Nous pensons que ces concepts ont un **degré de conscience faible** chez un joueur humain. Ce qui rend difficile la modélisation de ce niveau. Ce niveau est statique car limité par la puissance des machines.

Le niveau global est volontairement rudimentaire car nous pensons que le bon comportement global du modèle résulte plus du bon comportement des entités locales du niveau du dessous : les groupes. L'intelligence du modèle est **répartie et distribuée**. Ce niveau est statique.

La modélisation de l'**incrémentalité** basé sur le concept d'**empreinte** d'un objet posé sur le goban permet de réduire considérablement le temps de réponse pour jouer un coup : **10 secondes à 1 minute par coup** sur des stations de travail Sun Sparcstation10.

Les niveaux itératif et global sont statiques mais prévus pour devenir **dynamiques** dès que la puissance des machines le permettra. De plus, les calculs (un par intersection ou chaîne au niveau zéro, un par jeu au niveau élémentaire, un par groupe ou territoire au niveau itératif, un pour le niveau global) sont distribués. Ils pourront être faits **en parallèle** ce qui réduira le temps de réponse.

Correspondance entre le modèle computationnel avec le degré de conscience des connaissances humaines

Les figures *Correspondance-INDIGO-*conscient** résument l'ensemble des correspondances tirées tout au long de la partie 3 entre le modèle computationnel et le modèle cognitif.

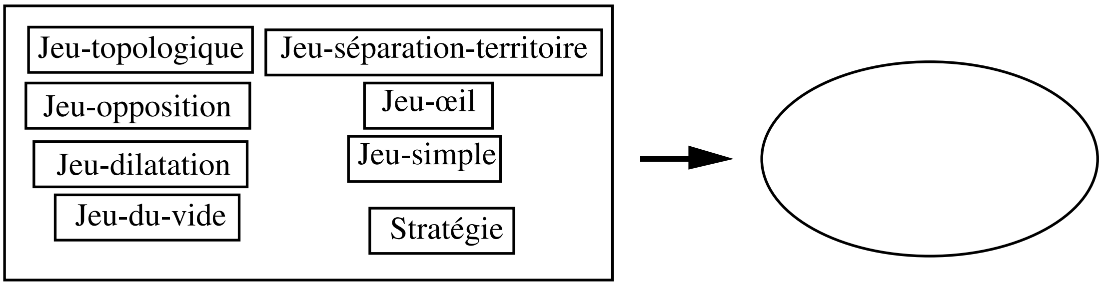


figure Correspondance-INDIGO-conscient

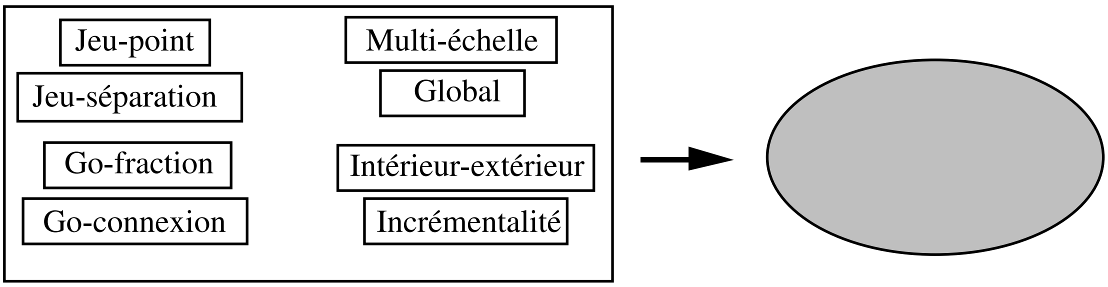


figure Correspondance-INDIGO-conscientisable

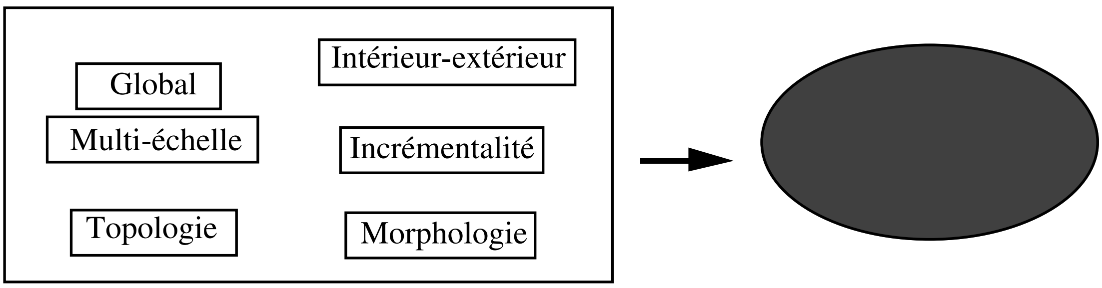


figure Correspondance-INDIGO-inconscient

Nous discuterons de cette classification au chapitre Correspondance avec les connaissances de l'être humain dans la partie 4.

PARTIE 4 : ÉVALUATION

Le but de cette quatrième partie est d'évaluer notre travail par rapport au but décrit dans la première partie : élaborer un modèle cognitif du joueur de Go pour expliciter des connaissances intuitives de l'être humain.

Dans la partie 2, nous avons montré comment la validation d'un modèle cognitif était favorisée par une réalisation informatique. Dans un premier chapitre, nous donnons **les résultats du programme INDIGO et les perspectives d'évolution**.

Au deuxième chapitre, nous rappelons **la méthode d'explicitation de concepts non conscients, basée sur une implémentation du modèle, la machine jouant le rôle de révélateur**. Nous mettons en **correspondance des concepts computationnels avec des concepts plus ou moins conscients d'un joueur de Go humain**. Pour ce faire, nous utilisons les conclusions intermédiaires énoncées en partie 3.

Enfin, au troisième chapitre, nous présentons une **réflexion sur notre travail au travers de trois domaines de l'Intelligence Artificielle : le Méta, l'Intelligence Artificielle Distribuée et la Logique Floue**.

RÉSULTATS DU PROGRAMME INDIGO ET PERSPECTIVES

Dans ce chapitre, nous présentons :
- les résultats des parties jouées contre des adversaires humains,
- les résultats des parties jouées contre d'autres programmes,
- le niveau du programme en kyu,
- l'aspect pratique,
- les perspectives envisagées pour la suite.

Contre des joueurs humains

INDIGO a joué 200 parties sur 9-9 ou 13-13 contre une quinzaine de joueurs humains débutants (de 30 ème kyu) à moyens (10 ème kyu). Contre tous les débutants, INDIGO a gagné les premières parties. Après, cela dépendait de la progression du débutant. En général, INDIGO restait plus fort dans le domaine purement tactique: les shichos et les captures de chaîne sont rarement des causes d'erreurs pour INDIGO face à un débutant. INDIGO ayant des connaissances statiques sur la vie et la mort des groupes, il tuait souvent des groupes aux débutants. Un point faible était sa focalisation sur des groupes instables de petite taille qu'il tuait en dépensant beaucoup de coups. **INDIGO a permis d'enseigner le Go à une quinzaine de joueurs débutants**. Son évaluation du goban en terme d'intersections contrôlées par les deux joueurs permettait au débutant de savoir ce qui appartenait à qui et ainsi de l'aider à choisir le coup. En annexe, nous donnons une partie jouée par INDIGO contre Hugh Bellemarre qui a été le partenaire humain de INDIGO le plus régulier.

Contre d'autres programmes de Go

INDIGO a joué contre plusieurs programmes :

- Many Faces of Go de David Fotland
- Poka de Howard Landman
- Gogol de Tristan Cazenave

Le nombre de parties jouées contre ces programmes dépend de l'**automatisation des parties**. INDIGO a joué une cinquantaine de parties contre Gogol entre Septembre 93 et Juillet 94 grâce à un outil de communication entre programmes de Go que nous avons développé sur Unix avec Tristan Cazenave. Ainsi, les parties étaient lancées le soir, jouées la nuit et analysées le lendemain matin. Sans cet outil, nous n'aurions pas pu faire jouer autant de parties à nos programmes. Quelques programmes jouent sur IGS (International Go Server). Nous avons donc développé avec Tristan Cazenave une passerelle qui permettait à nos programmes de se connecter automatiquement sur le serveur et de jouer avec Many Faces of Go. Cela a permis à INDIGO de faire trois parties contre Many Faces of Go en Mars 94. Malheureusement, notre outil ne gère pas les nombreuses coupures du serveur IGS. Par conséquent, trois parties seulement ont été commencées sans être terminées.

Avec les autres partenaires artificiels d'INDIGO, le nombre de parties jouées a été limité car jouées **"à la main"** : un programme joue un coup, on rentre le coup dans l'autre programme par l'interface homme-machine, etc... C'est faisable et très amusant pour une ou deux parties. Cela permet d'échanger des idées entre concepteurs de programmes, de prévoir quel coup va jouer son programme et expliquer pourquoi il l'a joué. Mais cela prend beaucoup de temps et limite le nombre de parties possibles.

Many Faces of Go

INDIGO a joué 3 fois contre Many faces of Go des débuts de parties de 150 à 200 coups en utilisant notre passerelle avec IGS. A chaque fois la partie a été interrompue par une coupure d'IGS. Lorsque le serveur redémarrait, Many Faces avait soit disparu, soit déjà commencé une partie avec un autre joueur... Les parties interrompues n'ont pas pu être continuées.

Nous donnons en annexe un des trois débuts de partie entre INDIGO et Many Faces of Go. Sur les trois débuts de partie, INDIGO a fait jeu égal sur l'un et était en retard sur les deux autres. Nous pensons qu'INDIGO est environ **2 ou 3 pierres plus faible** que Many Faces of Go.

INDIGO est combatif. Son modèle sur les groupes le rend capable de tuer des groupes à Many Faces of Go et de reconnaître ses groupes morts... Par contre, il ne construit pas de moyo comme est capable de le faire Many Faces of Go. INDIGO gère les territoires moins bien que Many Faces. INDIGO prend du retard et s'il ne tue pas de groupes, il perd.

Poka

INDIGO a fait une seule partie "à la main" avec Poka sur IGS, le programme de Howard Landman. Nous avons interrompu la partie alors qu'INDIGO devait avoir une vingtaine de points de retard. Nous pensons qu'INDIGO est environ **une pierre plus faible** que Poka. Le style de Poka est très différent de celui d'INDIGO. Poka construit de très grands moyos et laisse tomber des pierres investies en début de partie. Cette stratégie s'avère payante contre INDIGO qui dépense ensuite beaucoup de coups pour tuer ces pierres et enlever l'aji correspondant.

Gogol

La cinquantaine de parties entre Gogol et INDIGO a montré plusieurs choses intéressantes. INDIGO et Gogol ont beaucoup progressé entre Septembre 93 et Juillet 94 en jouant l'un contre l'autre. En jouant contre lui-même, un programme ne montre pas ses vraies faiblesses car personne ne les sanctionne. Par contre, en jouant contre un programme conçu autrement, le programme met à jour les faiblesses de l'autre programme et réciproquement. Ainsi nous avons pu déboguer nos programmes plus rapidement. En septembre 93, INDIGO était environ 25ème kyu et Gogol 30ème kyu. En juillet 94, on peut dire que Gogol est 18ème kyu et INDIGO 19ème kyu environ. Pour INDIGO, les améliorations dues à l'analyse de parties Gogol-INDIGO sont :

La définition, la programmation et l'intégration du jeu de l'intersection.

Contre Gogol, INDIGO a perdu des parties gagnées en bouchant des intersections neutres. Rajouter le jeu de l'intersection et l'intégrer dans la décision du coup global a permis de vérifier le coup joué et d'améliorer nettement le niveau du programme.

Le renforcement des conditions sur la mort d'un groupe

INDIGO possédait des contraintes trop faibles sur la mort d'un groupe. Il pensait avoir tué des groupes, ce qui était vrai. Mais Gogol ne sachant pas que ses groupes étaient morts jouait pour les renforcer. INDIGO, très zen, regardait cela avec amusement jusqu'au moment où les groupes revivaient ! En rajoutant ce qui correspond aujourd'hui à la vacuité dans INDIGO, INDIGO est devenu plus rigoureux sur la reconnaissance de la mort d'un groupe.

L'amélioration du jeu de la chaîne de 3 à 4 libertés.

Gogol possède un jeu de la chaîne performant. Gogol voyait des captures de chaîne sur des chaînes à 3 libertés qu'INDIGO ne voyait pas. Passer à 4 libertés a permis à INDIGO de voir plus de captures de chaîne.

Gogol est plus tactique qu'INDIGO notamment sur le jeu de la chaîne[1] et sur le jeu de la connexion. INDIGO, par contre, possède une modélisation plus poussée sur les groupes. Au total, INDIGO est environ **une pierre à deux pierres plus faible** que Gogol. Le style de Gogol est combatif comme celui d'INDIGO. Selon le recul que Tristan ou moi prenions en regardant les parties, celles-ci pouvaient être soit éprouvantes pour les nerfs, soit très amusantes et sources d'éclats de rire. Nous pensons que INDIGO et Gogol peuvent encore s'améliorer et arriver au niveau de Many Faces of Go.

Le niveau du programme INDIGO

Les parties contre des joueurs humains et artificiels ont montré qu'INDIGO était un peu meilleur que 20ème kyu, peut-être 18 ou 19ème kyu. Toutefois, il faut émettre des réserves sur un niveau trop précis. D'une part il existe une différence sensible de comportement entre un programme et un joueur humain et d'autre part la faiblesse de ces niveaux les rend vagues et imprécis. Ces réserves nous poussent à utiliser un chiffre rond et dire qu'**INDIGO est environ 20ème kyu**.

L'aspect pratique

La validation pratique du modèle cognitif est fondamentale dans notre approche. Nous avons choisi de le faire en implémentant notre modèle sur machine sous forme du programme INDIGO. L'implémentation de notre modèle a consommé 50% de notre temps. 90% des idées qui étaient dans notre tête au départ ont été invalidées par la machine. En effet, nous pensons que le processus qui va de l'idée au programme implémenté et testé sur machine est imprévisible dès que le domaine est complexe. 90% des idées qui sont actuellement validées dans notre programme proviennent de cycles répétés idée-modèle-programme-tests. Pour réduire les 50% de temps de programmation, il a été important de bien choisir nos outils.

Le choix du langage

Différents critères interviennent pour choisir un langage de développpement.

La rapidité à run-time

Un des facteurs qui limitent la force des programmes de Go est la puissance des machines. De ce point de vue il est obligatoire d'utiliser un langage rapide. Le C est idéal.

La rapidité de développement

Un des facteurs qui limitent la force des programmes de Go est aussi le temps de développement. De ce point de vue, il est conseillé d'utiliser un langage interprété comme Lisp. Cela accélère la mise au point. Si le programme s'est trompé sur une position et que l'on veut modifier une fonction du programme pour voir son effet, il est beaucoup plus pratique d'avoir un langage interprété : on modifie la fonction, on relance le programme sur la position et on regarde. Si l'on a un langage compilé dans ce type de cas, il faut arrêter le programme, modifier la fonction, recompiler, relancer le programme pour le replacer dans la position où il s' était trompé, puis regarder. La rapidité du développement pousserait à utiliser un langage de type Lisp. Ce qui est dit

[1]En jouant l'un contre l'autre, Gogol et Indigo ont fini par avoir beaucoup de points communs. Entre autres, le fait de gérer des jeux au sens de Conway et d'avoir un ensemble de jeux assez similaire.

doit être atténué, car il est parfois possible de charger dynamiquement un module compilé dans une application[1].

La complexité du domaine

La complexité du jeu de Go entraîne l'utilisation d'une conception orientée objet, voire une programmation orientée objet. En effet un goban est comme une base de données que le programme interroge pour savoir QUOI se trouve OU.

La nature visuelle du Go

Le jeu étant visuel, la connaissance acquise par le programme est essentiellement picturale (deux dimensions). Cela a eu deux conséquences dans notre travail.

D'abord, nous ne concevons pas un programme de Go sans **base de "formes"** (règles dont les prémisses contiennent des patterns). Puisqu'aucun langage à deux dimensions n'existe qui fasse un pattern-matching efficace, il est nécessaire d'en construire un adapté au Go.

Ensuite, il est plus facile de mettre au point le programme avec un **outil de mise au point** avec les deux dimensions. Autrement dit, il faut une interface ou un langage qui permette de savoir *Quoi* (un groupe, un territoire, une chaîne, une connexion, un œil, etc.) est *Où* (sur quelle intersection) de façon visuelle.

Nous avons construit ces outils nous-même. Que le langage initial soit C ou Lisp, le programmeur de Go doit construire un langage de règles et un outil de mise au point qui intègrent les deux dimensions spatiales du Go.

Conclusion

Les critères de temps de développement et de temps de réponse vont à contresens. C est beaucoup plus rapide que Lisp mais on développe plus vite en Lisp qu'en C. Nous pensons que le temps de réponse est plus important et nous préférons utiliser un langage du type de celui de C. De toute façon, le langage de départ n'a pas d'importance pour la partie visuelle déclarative du programme. La complexité du domaine pousse à utiliser Smalltalk si l'on a privilégié la rapidité de développement et à utiliser C++ si l'on a privilégié le temps de réponse. Dans notre cas, **nous avons utilisé C++**.

Le développement

Pour développer un programme de Go, il y a des parties incontournables : gérer la règle du jeu, avoir une interface homme-machine, avoir un pattern-matcher, avoir un module associé au concept de jeu. On peut réutiliser des modules faits par d'autres ou les développer soi-même. Nous avons choisi de tout développer nous-même par souci d'homogénéité. De toute façon ces parties du programme sont petites devant le reste où pour chaque concept du Go, il faut avoir une classe ou un module qui le représente. En général, le premier jet du développement donne un programme très lent dont il faut améliorer le temps de réponse. Nous avons beaucoup travaillé sur la non-lenteur du programme en mettant au point un mécanisme d'incrémentalité[2].

---

[1]Avec Unix et C, il est possible de le faire à condition de gérer des pointeurs vers les fonctions modifiées au lieu de gérer les fonctions directement.

[2]Ce mécanisme est décrit dans la description du modèle dans partie 3 du document.

La mise au point

En phase de mise au point, il est important d'avoir des jeux de tests unitaires. Ceux-ci permettent de vérifier que le programme sait toujours interpréter correctement certaines positions et résoudre certains problèmes.

Pour trouver les faiblesses du programme, il est important de le faire jouer contre des adversaires. Le plus pratique est d'avoir un adversaire artificiel et un outil automatique qui fait jouer les adversaires l'un contre l'autre. En effet, organiser à la main une partie entre deux programmes coûterait beaucoup de temps. Donc la nuit, notre programme a joué contre lui-même ou contre Gogol, le programme de Tristan Cazenave. Cela a permis de trouver beaucoup de bogues - le programme ne fait pas ce que le concepteur a prévu. Le matin, il fallait analyser la ou les parties jouées. Il fallait trouver les positions boguées et refaire l'interprétation de ces positions. Cela pouvait être long car, pour une position donnée, il fallait interpréter en mode absolu au lieu d'interpréter en mode incrémental.

Perspectives

Les perspectives pour améliorer notre programme sont les suivantes.

Numériser les attributs d'un groupe en utilisant des paramètres existants mais inutilisés

Actuellement, les attributs d'un groupe sont symboliques : >, *, <, 0, ?, ✿. L'état ✿ d'un groupe n'est pas assez discriminant. En fait, beaucoup de groupes sont dans cet état en cours de partie. INDIGO choisit ses coups au hasard lorsque deux groupes de taille égale sont en concurrence dans l'état ✿. Il nous faut préciser ce symbole en le **numérisant** en utilisant des paramètres comme la force de l'encerclement du groupe, la distance d'encerclement du groupe, les directions d'encerclement du groupe, la taille du territoire contrôlé par le groupe qui existent déjà (ou presque) dans notre programme.

Découpler les attributs d'un groupe pour le choix du coup en associant une urgence aux coups

Pour choisir un coup associé à un groupe INDIGO ordonne de façon figée les attributs du groupe. Sans donner les détails, INDIGO dit que l'amitié (la connexion) est prioritaire sur l'encerclement qui est prioritaire sur la base de vie qui est prioritaire sur la vacuité qui est prioritaire sur l'inimitié. Cet ordre offre un avantage statistique. Puisqu' il est difficile de décider statiquement sur une position donnée si l'un ou l'autre de ces objectifs est prioritaire, nous avons choisi de dire aux groupes d'INDIGO de toujours se connecter, puis encercler, puis vivre sur place, puis occuper l'espace, puis affaiblir les groupes ennemis en se renforçant. Dans beaucoup de positions, INDIGO se trompe en jouant des coups de connexions inutiles. Dans beaucoup de positions INDIGO se connecte et cela évite la catastrophe. Statistiquement, cette approche est défendable. Mais INDIGO pourrait faire beaucoup mieux. En fait, pour chaque objectif, INDIGO dispose de coups engendrés par des règles. En associant une **urgence** à ces règles, nous pensons qu'INDIGO jouera des coups beaucoup plus adaptés aux positions qu'il rencontrera et diminuera la part de hasard de son comportement.

Mieux utiliser les territoires pour choisir le coup global

Notre programme reconnaît les territoires et les utilise au sein du modèle sur les groupes mais le niveau global ne les utilise pas. Il nous faut **rajouter le jeu de l'érosion-dilatation d'un territoire** et l'utiliser au niveau global. Jusqu'à maintenant nous avons supposé que la notion de territoire est moins prioritaire que celle de groupe. Nous construisons les territoires une fois que les groupes sont stables. En effet, un territoire limité par un groupe en mauvaise santé n'est pas vraiment un territoire. Il en résulte que le concept de territoire est un concept coûteux en temps machine. Nous pensons faire un module de territoire allégé qui permettra de faire des calculs pour connaître la valeur d'un coup en termes de points de territoires gagnés ou perdus.

Amorcer la modélisation du "ko"

INDIGO respecte la règle du "ko" (cf. Annexe A règle du jeu). Une perspective séduisante serait de rajouter la gestion du ko dans le module de jeu existant. Il faut alors enrichir l'ensemble des états dynamique d'un jeu de l'état Ⓚ (cf. Berlekamp) et trouver un mécanisme qui simule ce que font les joueurs humains : prendre le ko, faire une menace de ko, répondre à la menace de ko ou gagner le ko. Cela paraît simple à faire[1], et peut améliorer significativement le niveau d'INDIGO si cela réussit..

Attendre que la puissance des machines augmente et qu'elles deviennent parallèles !

Nous avons vu qu'INDIGO effectue des calculs aux niveaux chaîne et élémentaire et que les niveaux "itératif" et "global" sont statiques. Si la puissance des machines augmente beaucoup (cent fois), le **niveau "groupe" pourra devenir dynamique**. Si la puissance des machines augmente encore (cent fois), **le niveau "global" pourra devenir aussi dynamique**. De ce point de vue, nous avons vu général et le modèle d'INDIGO a tout à gagner d'une augmentation de la puissance des machines. En attendant que les machines soient plus puissantes, nous pouvons encore enrichir notre modèle et l'optimiser.

La nature distribuée du jeu de Go et l'architecture de notre programme facilitent une **évolution vers le parallèlisme**. Chaque calcul sur chaque jeu d'un niveau peut se faire indépendamment des autres calculs.

Conclusion

INDIGO a joué des parties contre plusieurs programmes de Go : Many Faces of Go, Poka, et Gogol, et contre de nombreux joueurs humains de notre laboratoire d'accueil, le LAFORIA. Nous pensons que **son niveau se situe au dessus de 20 ème kyu**.

L'aspect pratique, fondamental pour obtenir des résultats évaluables, limite ces résultats. **Choisir, construire et utiliser des outils qui allègent la charge pratique** a été nécessaire.

A court terme nous envisageons des améliorations simples de notre programme : **préciser l'état incertain d'un groupe, faire calculer sur le jeu de la dilatation-érosion d'un territoire**.

A plus long terme, le temps travaille pour nous car notre architecture est prévue pour passer facilement d'une interprétation statique à **une interprétation dynamique** du goban dès que la puissance des machines le permettra. Notre modélisation orientée objet permettra aussi de passer plus facilement sur des **machines parallèles** dès que la technologie du parallélisme sera banalisée.

[1]Beaucoup de choses à faire, simples en apparence, sont devenues complexes lorsque nous avons entrepris de les faire...

CORRESPONDANCE ENTRE LES CONCEPTS DU MODÈLE ET LES CONCEPTS D'UN JOUEUR DE GO

Introduction

Dans ce chapitre nous évaluons notre modèle computationnel suivant le point de vue du degré de conscience d'une connaissance d'un joueur de Go humain. Le modèle computationnel regroupe un ensemble de concepts dont nous ignorions, soit l'existence, soit les modes de reconnaissance ou les modes d’emloi, au début de notre thèse. Notre travail, basé sur l'implémentation sur machine a permis d'expliciter des concepts non conscients chez le joueur humain, la machine a joué le rôle du révélateur. Nous passons en revue les concepts de notre modèle en leur attribuant un degré de conscience (conscient, conscientisable, intuitif).

En aucun cas, nous ne discutons de la conscience des connaissances par une machine ou par un modèle computationnel. La machine a apporté de la rigueur à notre validation mais il serait maladroit d'aller plus loin en parlant de conscience des connaissances par le modèle computationnel. Il est possible de dire que, pour une machine, les connaissances déclaratives d'un système sont "conscientes", et que les connaissances procédurales ne le sont pas, mais nous ne nous engagerons pas dans ce débat.

Enfin, nous relions certains concepts de notre modèle, que nous supposons non conscients chez un joueur humain, avec certaines verbalisations de joueurs humains. Avant tout, nous commençons par discuter de la conscience du temps et de l'espace chez un joueur en train de jouer au Go.

Le plan de ce chapitre est le suivant :

La conscience du temps et de l'espace chez un joueur de Go humain
La machine révélatrice de concepts non conscients
Les concepts non conscients chez un joueur de Go humain
Des associations entre les concepts et les verbalisations
Vue synoptique

La conscience du temps et de l'espace chez un joueur humain

Nous avons été frappé par la sobriété des commentaires de très forts joueurs de Go lorsque nous sommes allés dans des clubs de Go au Japon. Un soir, autour d'un goban, cinq ou six joueurs japonais, 7 ème ou 8 ème dan (en équivalent de dan européen) commentaient une partie de Go. Les commentaires donnaient à peu près ceci :

- "*Koko ima !*", et une séquence de coups suivait.
- "*Koko ima !*", encore et une autre séquence de coups suivait, etc.

Les joueurs communiquaient au moyen de ces seuls deux termes. **Koko** signifie **ici** et **ima** signifie **maintenant.** Tout le Go est résumé par ces deux mots concepts. **Quand** et **où** faut-il jouer un coup ?

Nous pensons que tous les joueurs de Go, débutants ou forts ont une conscience d'une position qui se décompose en une composante spatiale et une composante temporelle. Suivant sa capacité à mettre en relation des faits spatialement voisins - avec des voisinages d'autant plus grands que le joueur est fort - et suivant sa capacité à imaginer les positions qui succèdent à la position courante - d'autant plus loin que sa conscience temporelle le permet - le joueur de Go est capable de savoir **où** et **quand** jouer. Ces composantes sont très dépendantes. Nous ne savons pas comment cette dépendance est faite, c'est ce qui a rendu notre recherche passionnante.

La machine révélatrice de concepts non conscients

Avant de discuter du degré de conscience des concepts humains, il est nécessaire de donner un **nouvel éclairage** sur ce que nous entendons par conscient, conscientisable et intuitif et de montrer comment **la machine nous a servi de révélateur** de concepts non conscients. Les degrés de conscience d'un concept sont flous en réalité et il faut arrondir les définitions ci-dessous. Pour simplifier, nous supposons qu'un concept possède un *mode de reconnaissance* et un *mode d'emploi.* On peut définir le degré de conscience pour un concept ou pour ses modes. Dans la suite de ce paragraphe, nous employons le terme concept au sens du concept mais aussi au sens du mode de reconnaissance et au sens du mode d'emploi.

Retour sur les définitions du degré de conscience

Conscient :

Un concept (ou un de ses modes) est conscient s'il est présent dans les verbalisations de joueur humain sous forme d'un terme *sans effort particulier* de la part du joueur.

Conscientisable :

Un concept (ou un de ses modes) est conscientisable s'il est présent dans les verbalisations de joueur humain sous forme d'un terme *moyennant un effort particulier* de la part du joueur et *un apport extérieur*. Cet effort est un effort de conscientisation. Nous entendons le terme conscientisation dans un sens un peu différent de celui de Pierre Vermersch [Vermersch 1991b]. Celui-ci suppose que l'apport extérieur est un psychologue qui pose de bonnes questions (comme ENFANDELEFAN[1]) du genre de : "*Quand tu te trouvais dans l'état où tu étais, tu pensais à quoi ?*". Nous supposons que l'apport extérieur peut être la machine.

[1]cf. Partie 2 verbalisations, histoire comme ça

Intuitif :

Un concept (ou un de ses modes) est intuitif s'il reste absent dans les verbalisations de joueurs humains *même moyennant un effort important* de la part du joueur et *un apport extérieur*.

<u>La machine révélatrice</u>

Reprenons un historique rapide de notre travail. Nous sommes un fort joueur de Go. Nous avons élaboré un premier modèle cognitif sur le Go. Nous n'avons pas fait d'effort particulier pour conscientiser des concepts. Nous avons obtenu un premier modèle cognitif :

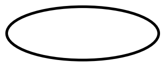


figure révèle-0-ovale

Nous avons créé un modèle computationnel :

figure révèle-0-rectangle

Les résultats de la machine étaient nettement inférieurs à ceux d'un joueur humain :

figure révèle-<<<

Nous les avons étudiés pour savoir ce qui manquait à la machine. Nous avons rajouté des concepts dans la machine :

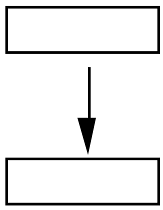


figure révèle-1-rectangle

Cette étude nous a montré que les concepts que nous avions ajouté étaient visibles dans les verbalisations. Nous avons supposé que ces nouveaux concepts mis dans la machine correspondaient à des concepts conscientisables car nous avons fait un *effort important* et les mots correspondant aux concepts étaient présents dans les verbalisations :

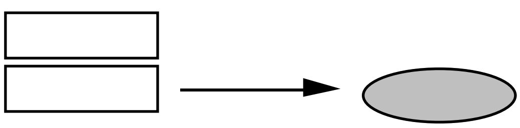


figure révèle-1-ovale

La machine nous a servi de révélateur de concepts conscientisables. En parallèle, nous avons fait un *effort important* d'introspection pour conscientiser d'autres concepts :

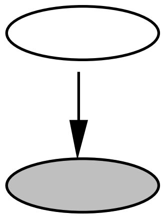


figure révèle-1-ovale-bis

Les résultats du modèle computationnel étaient moins mauvais qu'avant, mais pas encore acceptables :

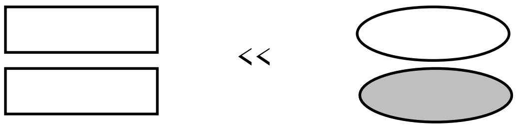


figure révèle-<<

Nous avons encore conscientisé des concepts avec l'aide de la machine :

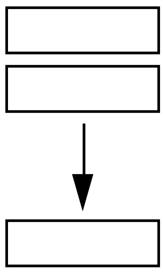


figure révèle-2-rectangle

Nous avons cherché une correspondance entre ces nouveaux concepts et des mots dans les verbalisations de joueurs humains. Nous n'avons trouvé que des termes qui désignaient des concepts mais pas leur mode de reconnaissance ni leur mode d'emploi. Nous avons alors fait l'hypothèse que ces nouveaux concepts computationnels correspondaient à des concepts cognitifs intuitifs :

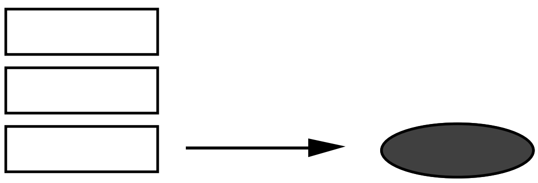


figure révèle-2-ovale

La descente directe par introspection n'était plus possible :

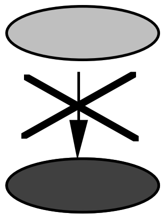


figure révèle-2-interdit

Les résultats commençaient à ressembler aux résultats de joueurs humains mais restaient faibles :

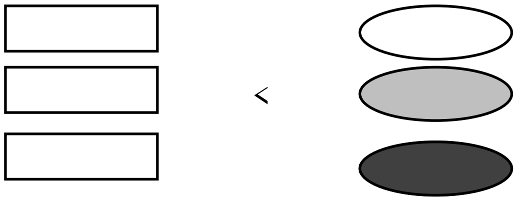


figure révèle-<

Nous avons mis en correspondance les concepts computationnels avec les concepts cognitifs :

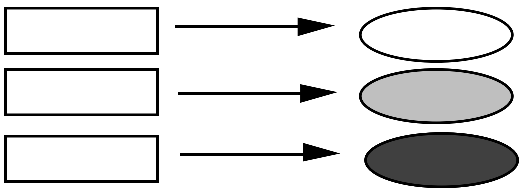


figure révèle-2-correspond

Finalement, le chemin pour accéder aux concepts conscientisables est double : il est possible d'introspecter directement ou bien d'implémenter un modèle computationnel, le valider et enfin faire une correspondance[1] :

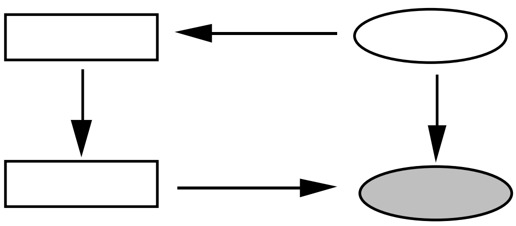


figure révèle-chemin-conscientisable

Par contre pour "accéder" aux concepts intuitifs, un seul "chemin" a été identifié dans notre travail : implémenter un modèle computationnel, le valider, et enfin faire une correspondance des concepts computationnels vers des concepts cognitifs, dont les modes d'emplois et mode de reconnaissance sont absents des verbalisations. L'introspection ne marche pas pour expliciter ces concepts :

[1]Cela ressemble à un trajet en métro. Pour des concepts souterrains cela tombe bien !

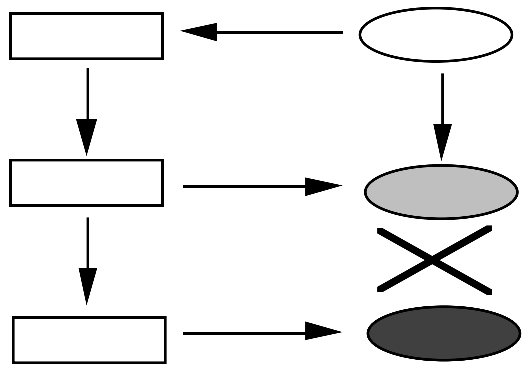


figure révèle-chemin-intuitif

Finalement, nous estimons que **la machine a joué un rôle de révélateur de concepts non conscients**. La machine nous a permis d'**expliciter des concepts conscientisables** et de **faire correspondre des concepts intuitifs chez l'homme avec des concepts computationnels** (indispensables à la cohérence du modèle computationnel).

Les concepts non conscients chez un joueur de Go

Il est possible d'extraire de notre modèle une liste de concepts que nous avons explicités au cours de notre travail. Pour chacun de ces concepts, nous faisons une correspondance avec des connaissances humaines et un degré de conscience associé. Ces correspondances sont approximatives pour deux raisons. D'abord, elles sont faites indépendamment de la progression d'un joueur. Ensuite, les frontières entre conscient, conscientisable et intuitif sont vagues en réalité, contrairement à notre formalisme.

Après la correspondance entre la liste de concepts et les connaissances humaines, nous discutons du degré de conscience des connaissances humaines en fonction du niveau et de la progression du joueur.

<u>Le jeu</u>

Chez l'homme

Un joueur de Go est avant tout un joueur et il en est **conscient**. Le concept le plus important au Go est le jeu.

Dans INDIGO

Un programme de jeu de Go comme INDIGO utilise la notion de jeu de façon intensive. Les jeux se décomposent en sous-jeux. Un jeu "appelle" des sous-jeux comme un programme appelle un sous-programme. Nous avons implémenté la partie "jeu" d'INDIGO sous forme procédurale (appels de sous-programmes C++). Nous pourrions imaginer reprogrammer cette partie "jeu" d'INDIGO dans un "langage de jeu". Avec un tel langage, nous définirions les jeux avec leurs propriétés et leurs relations décrites dans le paragraphe sur la théorie des jeux de Conway et dans le paragraphe métajeu.

<u>Le regroupement</u>

Chez l'homme

Le joueur de Go utilise une notion de regroupement complexe. Elle s'appuie en première approximation sur un regroupement intuitif visuel identifié par la théorie de la Gestalt [Kohler 19]. Cette notion de regroupement de bas niveau est **intuitive** et **générale**, elle est utilisée par le système visuel dans le monde réel. Elle serait indépendante des connaissances de haut niveau sur le domaine visualisé [Marr 1982]. Le joueur de Go débutant utilise cette vision intuitive pour amorcer la reconnaissance des groupes au Go. Quand il progresse, il mémorise des formes de connexion liées au Go et construit une notion de regroupement proche de celle qui existe dans notre modèle. Ce nouveau type de regroupement est acquis par l'expérience et est **conscient**, ou s'il ne l'est pas il est **intuitif mais conscientisable**. Quand il progresse encore, sa notion de regroupement s'enrichit en fonction des concepts "voisins" du regroupement : capture de chaînes ou groupes ennemis.

Dans INDIGO

INDIGO utilise une notion de regroupement strictement liée à la notion de jeu de la connexion. Il utilise aussi des regroupements par capture de groupes ou chaînes ennemis. Il n'utilise pas de regroupement au sens de la Gestalt. A un moment donné de notre recherche, nous avions essayé de simuler ce type de regroupement, intuitif chez l'homme, avec les opérateurs X(m,n) de morphologie mathématique (comme nous l'avons fait pour les territoires). Cela construisait des "groupes". Malheureusement, il manquait les connaissances d'utilisation de ces "groupes". Par rapport aux autres programmes, INDIGO avait un comportement catastrophique quand il utilisait ces "groupes". Nous avons donc abandonné cette voie et repris la notion correspondant au jeu de la connexion.

La fraction

Chez l'homme

Le joueur de Go utilise une notion de fraction très complexe. Nous pensons que cette notion se découpe en deux (comme la notion de regroupement). D'une part, une notion **intuitive et générale** du système visuel humain qui permet de segmenter ou fragmenter une image en régions. D'autre part, une notion de séparation proche de ce qui existe dans INDIGO (cf. la description des fractions dans le modèle). Cette notion est **intuitive et difficilement conscientisable**. En effet les joueurs de Go n'en parlent pas (comme nous l'avons précisé dans le paragraphe sur les fractions dans le chapitre qui présente notre modèle).

Nous pensons que le joueur de Go utilise les fractions pour plusieurs choses. D'abord, les fractions sont utilisées pour détecter les encerclements des groupes. Ensuite, le raisonnement du joueur de Go est apparemment basé sur les groupes mais aussi sur les fractions. Quand une fraction contient plus d'un groupe, le joueur *regroupe les groupes* d'une même fraction et raisonne alors sur la fraction plutôt que sur chacun des groupes de la fraction. En ce sens, une fraction est un regroupement. Enfin, la sémantique d'une fraction contient une notion d'indépendance : des intersections de fractions distinctes ont un avenir relativement indépendant. Le joueur humain utilise certainement les fractions pour raisonner indépendamment et donc incrémentalement. Il faudrait mieux préciser cette indépendance utilisée par l'homme pour améliorer notre modèle.

Dans INDIGO

La notion de fraction dans INDIGO est construite avec la notion de séparation présentée dans le paragraphe sur la morphologie mathématique. La notion de fraction permet de modéliser l'encerclement des groupes.

L' interaction

Chez l'homme

L'être humain utilise largement la notion d'interaction dans le monde réel. Le joueur de Go en fait autant dans ses parties de Go. Dans les commentaires de parties de Go, cela n'est pas directement apparent : "*Le groupe blanc est faible mais le groupe noir voisin est aussi faible, donc le groupe blanc n'est pas vraiment faible*". Cependant, ce type de phrase contient le concept d'interaction. Au premier degré, le groupe blanc est faible. Si l'on tient compte de l'ensemble de ses interactions avec ses voisins, il est de la même force que l'un deux, donc il n'est pas faible. Nous pensons que l'interaction est à la base de l'intelligence, chaque neurone du cerveau interagissant avec beaucoup de neurones. L'interaction est présente à **tous les degrés de conscience** du système cognitif humain.

Dans INDIGO

Dans INDIGO, la notion d'interaction est présente au niveau des groupes. Chaque groupe possède une santé qui synthétise l'état des interactions du groupe avec ses voisins. Elle est évidemment présente au niveau des intersections où chaque intersection connaît ses voisines.

La dualité intérieur-extérieur

Chez l'homme

Un être humain sait facilement reconnaître l'intérieur de l'extérieur des objets. Il n'en parle pas. C'est l'évidence pour lui. Les connaissances qu'il utilise pour le faire sont **intuitives**.

Dans INDIGO

Dans INDIGO, la dualité intérieur-extérieur apparaît à deux endroits. D'abord, un territoire est la fermeture morphologique d'un groupe de pierres. Il représente l'intérieur du groupe. L'influence représente l'extérieur d'un groupe. Ensuite, les propriétés d'un groupe sont partitionnées en propriétés intérieures et extérieures.

La catastrophe ou mort des groupes

Chez l'homme

L'homme est étonné par ce qui le choque. Appelons ceci des catastrophes. Le système cognitif humain intègre les catastrophes pour qu'elles deviennent normales et ne soient plus des catastrophes. Dans le cas de la mort des groupes au Go, le système cognitif est choqué au départ par l'effet des groupes morts. Visuellement, un groupe mort garde sa couleur. Par contre, le groupe prend conceptuellement la couleur opposée. Si le débutant est choqué par une telle inversion de couleur entre le niveau physique et le niveau conceptuel, le joueur qui progresse l'est de moins en moins. Un joueur fort n'est pas du tout choqué. Chez un joueur fort les connaissances de reconnaissance de groupe mort sont devenues des automatismes. Il peut expliquer ces automatismes, ses connaissances sont **conscientisables** ou **conscientes**. Chez un joueur moyen, ces connaissances sont **conscientes** et chez un joueur débutant elles n'existent pas encore.

Dans INDIGO

Dans INDIGO, une catastrophe est un groupe mort. INDIGO n'est aucunement choqué ! L'équivalent du choc humain dans INDIGO est le temps machine passé à reconnaître la catastrophe ou groupe mort. La connaissance pour reconnaître la mort des groupes n'est pas un automatisme.

L'aspect multi-échelle

Chez l'homme

L'homme synthétise les informations perçues à différentes échelles. Les connaissances correspondantes sont sûrement **intuitives**.

Dans INDIGO

INDIGO reconnaît les objets comme les territoires à l'échelle qui lui est spécifiée. Il ne gère pas d'aspect multi-échelle au sens où il trouverait lui-même l'échelle des objets qu'on lui présenterait.

L'incrémentalité

Chez l'homme

Lorsqu'une pierre est posée sur le goban, le joueur de Go ne recalcule pas ce qu'il a déjà calculé. Il sait que ce mécanisme existe chez lui mais ne sait pas expliquer comment il fait. Pour la bonne raison qu'il ne sait même pas expliquer comment il joue tout court. Les (méta)connaissances qui permettent d'interpréter incrémentalement le goban sont **intuitives**. Ce comportement incrémental est aussi un inconvénient car un joueur peut oublier de recalculer quelque chose et perdre une partie à cause de cela.

Dans INDIGO

Le mécanisme d'incrémentalité d'INDIGO est simple (basé sur la notion d'empreinte) et décrit dans la présentation de notre modèle dans la partie 3.

La multiplicité du choix ou la stratégie

Chez l'homme

La particularité du Go est d'être multi-jeu, multi-choix, multi-critère. Le processus de décision chez un joueur de Go est complexe. Il en parle, il est a priori **conscient** du produit de ce processus.

Dans INDIGO

Le processus de décision du coup au niveau global d'INDIGO est assez simple.

L'apprentissage du joueur humain

Chez l'homme

Au paragraphe décrivant le niveau élémentaire de notre modèle en partie 3, nous avons tiré des correspondances entre les jeux morphologiques et le degré de conscience des joueurs novice complet, débutant, faible et moyen. Nous pensons que le jeu de Go, par ses caractéristiques morphologiques et topologiques ressemblant au monde réel, rend le processus d'apprentissage au Go particulier.

Le processus d'apprentissage d'un joueur de Go correspond non pas à la seule acquisition de connaissances nouvelles, inexistantes avant l'apprentissage, mais plutôt à une *découverte* de *connaissances générales* et *cachées*, *pré-existantes* à l'apprentissage. Le processus de découverte serait, en fait, une *adaptation au jeu de Go*, ou *spécialisation pour le jeu de Go*, des connaissances générales sur le monde réel.

Dans INDIGO

Il n'existe pas de module d'apprentissage dans INDIGO. Certaines bases de règles ont été engendrées par programme.

Conclusion

Il est très difficile de conclure sur un sujet aussi difficile que la **découverte ou explicitation de concepts non conscients**. La première difficulté réside dans le **paradoxe qu'un concept non conscient explicité n'est normalement plus non conscient !** En plus, **la classification varie suivant le niveau et la progression du joueur**.

Cependant la liste de concepts suivante nous paraît significative du jeu de Go et correspondant à des **concepts majoritairement intuitifs** :

Le regroupement
La fraction
L'interaction
Intérieur-extérieur
Catastrophe
Multi-échelle
Incrémentalité

Les associations entre les concepts et les verbalisations :

Identifier des concepts sous les mots

Le premier réflexe du concepteur est d'**associer un mot à un concept**. C'est l'idée générale, elle est bonne mais elle doit être utilisée avec précaution.

Les associations faciles

Les termes "chaîne", "groupe", "territoire", "global", "jeu" se retrouvent dans la description de notre modèle. Il aurait été difficile de faire d'autres associations terme-concept que celles-ci.

Les associations problématiques

Souvent nous n'avons pas réussi à associer un concept à un mot :

Chasser le "*naturel*".

Le terme "*naturel*" est très fréquent chez les joueurs de Go. Nous ne savons pas encore lui associer un concept précis car il est ancré trop profondément dans l'être humain. Ce concept est topologique et morphologique. Il est intuitif et difficilement définissable. Nous avons chassé le "naturel" des verbalisations de joueurs mais il est toujours revenu !

Prendre le point "*vital*".

Le terme "*vital*" est également fréquent dans les verbalisations. Il est souvent associé à un concept visuel et spatial lié à la forme des pierres. Il est aussi associé à la notion de jeu : pour gagner un jeu, les deux joueurs doivent chacun jouer le premier sur le point "vital" (cf. plus loin).

Qu'est-ce qu'une "*attaque*" ?

Le terme "*attaque*" employé dans les verbalisations est dissociable en deux concepts. Attaquer peut signifier avoir l'initiative dans un jeu, (i.e. commencer) ou bien jouer des coups qui affaiblissent (attaquent) l'adversaire. Ces deux concepts sont cachés dans le même terme car normalement on essaie de bien jouer : affaiblir l'adversaire quand on a l'initiative !

Identifier des relations entre les concepts

Supposons que l'on ait la verbalisation suivante : *"Le groupe x est stable car il peut se connecter au groupe y, il peut se connecter car il peut déstabiliser le groupe voisin z"*. Comme nous l'avons précisé au paragraphe précédent, on associe un concept du modèle à chaque terme clef de la verbalisation : concepts de groupe G, de stabilité d'un groupe S et de connexion entre groupes C.

Les relations de qualification d'un concept par un autre concept

On associe une relation de qualification entre des concepts à chaque liaison par un verbe (ici être et pouvoir) :

S qualifie G *"Le groupe x **est** stable"*
C qualifie G, "*il **peut** se connecter au groupe y"*

Les relations d'utilisation entre les concepts

On associe une relation d'utilisation entre des concepts à chaque conjonction de coordination *"car"*, ce modèle contient les relations suivantes :

S utilise C, *"Le groupe x est stable **car** il peut se connecter au groupe y"*
C utilise S *"il peut se connecter **car** il peut déstabiliser le groupe voisin z"*

Les bouclages créés par ces assocations faites au niveau du langage

Notre travail a permis d'identifier un obstacle intéressant : **les bouclages apparents des relations d'utilisation entre concepts associés à des termes présents dans des verbalisations**.

Le bouclage des relations d'utilisation (S utilise C qui utilise S dans l'exemple précédent) est très gênant pour construire un modèle computationnel. Les bouclages sont très nombreux au jeu de Go où tous les concepts dépendent les uns des autres.

Pour se sortir de cette difficulté, nous avons choisi de paramétrer les concepts. Dans notre exemple, on amorce le concept S avec une simplification S(0) où S(0) ne dépend pas de C; par exemple, un groupe est S(0)-stable s'il possède au moins n libertés (4 par exemple). De même, on amorce le concept C avec une simplification C(0) où C(0) ne dépend pas de S; par exemple un groupe est C(0)-connecté à un autre groupe s'il possède au moins m libertés communes (2 par exemple). Ensuite, on améliore le concept S avec S(1) où un groupe est S(1)-stable s'il est S(0)-stable ou s'il peut se C(0)-connecter à deux groupes distincts. On améliore le concept C avec C(1) où deux groupes sont C(1)-connectés s'ils sont C(0)-connectés ou s'ils ont en commun une même intersection et un même groupe ennemi S(0)-stable-*, ou enfin s'ils ont en commun un groupe S(0)-stable-<. Bref, on applique la méthode décrite dans la partie 2 de ce document où les concepts utilisent des concepts du niveau inférieur.

Finalement, pour notre exemple, on n'aura pas de bouclage : S(1) utilise C(0) et S(0), et C(1) utilise S(0) et C(0).

Certains concepts sont paramétrables facilement :

le jeu de la chaîne avec le seuil de stabilité d'une chaîne, 3 ou 4,
la reconnaissance de l'influence avec le nombre de dilatations 1, puis 2 puis 3 puis 4
la reconnaissance de territoire avec les couples dilatation-érosion 1-1, 2-3, 3-7, 4-13, etc.
la taille des formes reconnues 3-3, 5-5, etc.
la distance de voisinage ami ou ennemi,
la distance d'encerclement,
les formes avec l'urgence des coups joués.

La difficulté est d'abord de trouver un paramétrage des concepts et ensuite de trouver l'agencement des concepts paramétrés vis-à-vis de la relation d'utilisation. Le problème principal est la rigidité de ces choix de conception. Ils sont en amont et toute la programmation qui suit contient ces choix. Si l'on se trompe dans l'agencement des concepts, modifier la conception du programme est coûteux. La puissance des machines fixe et limite le nombre de niveaux du modèle.

L'être humain n'est pas gêné par ces problèmes et en ignore même la présence. Si le système cognitif humain résout ce problème de circularité des relations d'utilisation des concepts, apparente dans les verbalisations des joueurs, il le fait inconsciemment.

Démonter, décompiler le langage naturel

Des proverbes tels que

"Si 3 diagonales sur 4 sont amicales, l'œil est vrai."

ont dû être démontés. Les "3 diagonales sur 4" du jeu de l'œil ont dû être démontées en : "Aucun jeu du point n'est perdu". Le jeu du point, plus général, a permis de reconstruire les règles valables pour les yeux de taille 1 mais aussi de taille 2 et pourra servir pour les territoires de taille n. Le langage naturel compile des connaissances et les cache. Premièrement, il faut oser poser la question de savoir si une phrase contient une information compilée et ensuite il faut la décompiler.

Lever des ambiguïtés

Vie et mort

Le terme "*vivant* " a dû être séparé en "*vivant sur place*" et "*vivant par voisinage*".

santé > ou 0 ≈ "vivant"

base de vie > ≈ "vivant sur place"

Le terme "*mort* " a été séparé

au sens de l'état intérieur et extérieur d'un groupe (la santé), un groupe "*mort*"

santé < ≈ "mort"

au sens de l'état intérieur seul d'un groupe (la base de vie), un groupe "*mort sur place*"

base de vie < ≈ "mort sur place"

au sens de la forme de la chaîne capturée, une "*forme morte*"

au sens de la forme du territoire, une "*forme morte*".

jeu séparation du territoire en 2 < ≈ "forme morte"

jeu séparation du territoire en 2 > ≈ "forme vivante"

Préciser le non-dit

le "*sente*"

Le terme "*sente*", qui signifie l'initiative, est très fréquent dans les verbalisations. En effet, le jeu de Go étant un multi-jeu, l'initiative joue un rôle fondamental : un joueur joue une séquence locale en pensant qui va terminer la séquence. Obliger l'adversaire à répondre à ses coups permet de jouer le premier dans les différentes situations. Dans un commentaire de partie, le joueur se dit : "*Est-ce sente ?*". Implicitement, le joueur pense à un complément d'objet qui reste non dit dans la question. Il devrait dire "*Est-ce sente sur quoi ?*". La réponse au contenu du quoi implicite possède plusieurs degrés. De façon la plus riche, le complément d'objet implicite est la partie complète : si l'adversaire ne répond pas comme prévu à ce coup, il perd la partie :

quoi = partie complète

Mais souvent, il se pose la question localement et dans ce cas :

quoi = un jeu local

Le gain du jeu local est suffisant pour "faire le break" dans le jargon du Tennis.

Associer un concept là où rien n'est dit : "   "

Une différence est apparente entre le programme qui voit un jeu < à un endroit et le joueur humain qui n'en voit pas (il n'en parle pas). Par exemple, un jeu peut-être < parce un objet n'a pas été reconnu (i.e. la chaîne a disparu et le jeu de la chaîne est <, aucune forme de connexion n'est reconnue et le jeu de la connexion est <, le groupe n'a pas d'ami et son amitié est <).

Parfois, < ≈ " "

Si le joueur humain ne dit rien dans ce type de situation, il reconnaît peut-être des objectifs non atteignables.

Identifier des concepts "transversaux" correspondant à des mots ou suffixes indépendants du domaine.

Les suffixes du langage naturel cachent généralement des concepts généraux transversaux (orthogonaux aux concepts à tout le domaine étudié, mais qui permettent de les qualifier).

Un suffixe général comme **-able** (dans connect**able**, tu**able** par exemple) a dû être remplacé par * pour exprimer la dichotomie possible vers deux états stables opposés. Par exemple, la phrase : "*Le groupe est tuable en un coup*" se traduit dans le modèle INDIGO par tel groupe possède une santé *. Ou encore : "*Les deux groupes sont connectables*" par l'interaction amie entre les deux groupes est *.

Pour un joueur moyen, * ≈ "-able"

Un joueur moyen dit aussi : "*Les deux groupes sont connect**és***" pour signifier que l'adversaire ne peut plus déconnecter. Le jeu de la connexion est >.

Pour un joueur moyen, > ≈ "-é"

Pour un joueur faible, le suffixe **-able** signifie plutôt > par opposition à 'gagné'. "*Les deux groupes sont connectables*" signifie pour lui que les groupes ne sont pas connectés au sens de la règle du jeu. Le jeu de la connexion n'est pas dans l'état statique 'gagné' mais seulement dans l'état dynamique >. Le joueur qui emploie -**able** dans ce sens n'a pas encore maîtrisé la notion de jeu en général ou bien qui n'a pas la connaissance ou la capacité de calcul suffisante permettant de constater que le jeu est > et dire que "*les groupes sont connect**és***".

Pour un joueur débutant, > mais non gagné ≈ "-able"

Le terme "vital" est aussi associé à l'état dynamique * quand le lieu du coup à jouer dans un jeu * est le même pour les deux joueurs.

*** ≈ "vital"**

Vue synoptique de la correspondance

Finalement, nous regroupons les correspondances entre les concepts de notre modèle et les degrés de conscience des connaissances chez un joueur humain dans trois figures. Les ovales représentent les parties consciente (en blanc), conscientisable (en gris clair) et intuitive (en gris foncé). Les rectangles représentent les concepts de notre modèle. Les flèches représentent les correspondances.

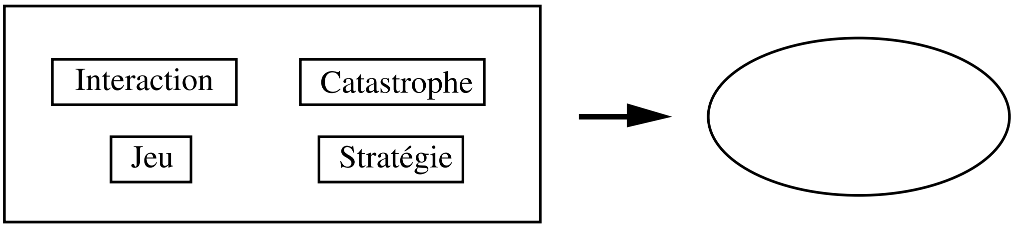


figure Evaluation-conscient

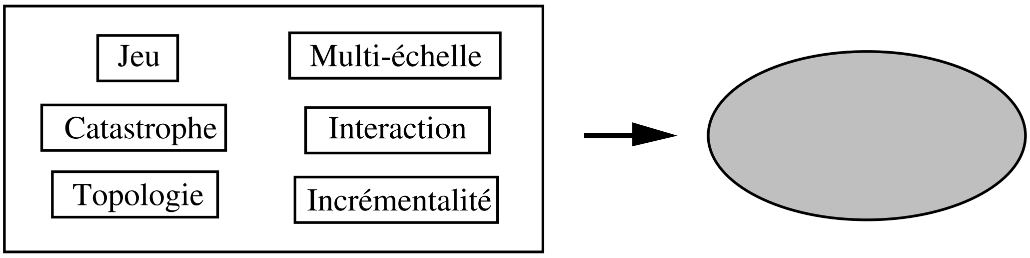


figure Evaluation-conscientisable

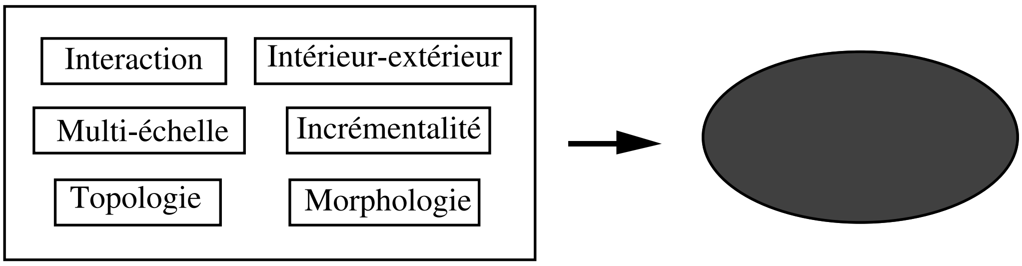


figure Evaluation-intuitif

La machine a servi de révélateur pour expliciter des concepts non conscients chez le joueur de Go.

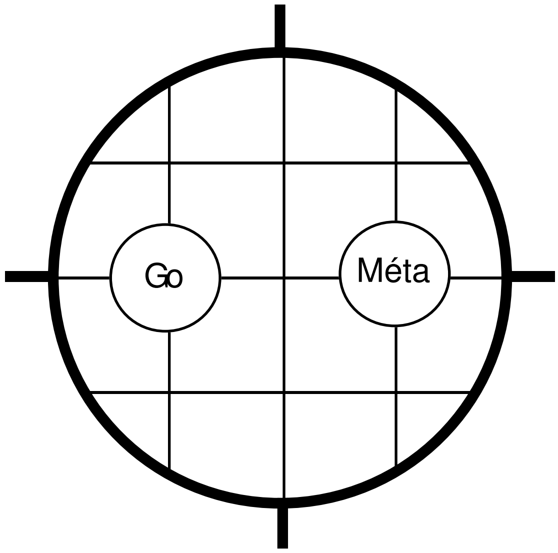
NOTRE TRAVAIL AU TRAVERS DE DOMAINES VOISINS DU JEU DE GO

Notre travail, présenté dans la partie 3, a utilisé des analogies faites entre le jeu de Go et des domaines de l'IA ou des mathématiques. Nous avons appelé ces domaines, les domaines "voisins". L'utilisation des domaines voisins a été présentée dans la partie 2.

Le but de ce chapitre est d'illustrer en retour certains domaines voisins au travers de notre travail : le **Méta**, la **Logique Floue** et l'**Intelligence Artificielle Distribuée**.

Le Méta

Dans ce paragraphe, nous montrons des points communs entre le "Méta" et le Go.



Ce que nous entendons par Méta

Notre travail a été fortement influencé par la notion de Méta étudiée par Jacques Pitrat et [Pitrat 1990] nous a servi de référence. Il est possible de donner plusieurs définitions du concept de "Méta". De façon générale, *nous dirons que quelquechose est un métaquelquechose lorsque celui-ci s'applique à des quelquechoses du même type*.

Exemple:

Une métaconnaissance est une connaissance qui s'applique à des connaissances.

Le but de ce paragraphe sur le "Méta" est de **montrer les concepts *quelquechose* dans notre travail qui sont ou peuvent être affinés en concepts *métaquelquechose***.

Nous avons identifié deux concepts candidats : le jeu et le pattern.

La notion de métajeu

Introduction :

Le but de ce paragraphe est de définir un concept de *métajeu* et de voir dans quelle mesure la modélisation du jeu de Go est liée à ce concept. Le concept très général de "Méta" trouvera alors un exemple d'application dans le domaine des jeux et du Go.

Il est indispensable de se reporter au paragraphe sur la théorie des jeux pour savoir ce que nous entendons par *jeu*.

Nous n'entendons pas Métajeu au sens de la généralité comme cela est fait dans Métagame [Pell 1992]. Metagame est un système général de jeux ressemblant au jeu d'Échecs, où l'on donne au système les règles du jeu. Metagame utilise des heuristiques générales de jeu pour jouer et des techniques d'apprentissage pour progresser.

La notion de métajeu au Go suppose l'existence d'une structure conceptuelle[1] et d'une évaluation statique[2] pour toute position. Différentes structures ludiques peuvent être appliquées sur la structure conceptuelle selon l'approche choisie. Nous montrons tour à tour ces approches. Enfin nous cernons en quoi ces approches utilisent la notion de *métajeu*. et nous discutons de l'avantage de chacune d'elles.

Notre discussion est illustrée par un exemple.

Qualification statique :

Une structure conceptuelle :

Nous prenons en exemple une position décrite comme suit. Premièrement, la position globale est composée de 4 groupes de pierres comme sur la figure *Méta-goban-4-groupes* .

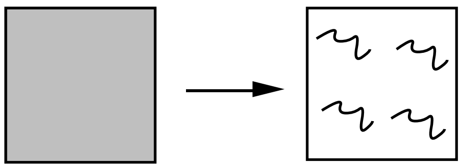


figure Méta-goban-4-groupes

Deuxièmement, chaque groupe de la position a 2 yeux comme sur la figure *Méta-groupe-2-yeux* .

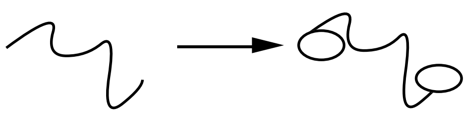


figure Méta-groupe-2-yeux

Enfin, chaque oeil est un élément terminal de la structure qui est centré sur une intersection vide entourée de pierres de sa couleur. La structure de la position est résumée par la figure *Méta-structure*.

[1]La partie 3 a montré que le choix de la structure est une des principales difficultés de la programmation du jeu de Go.
[2]L'évaluation statique d'une position est une autre difficulté.

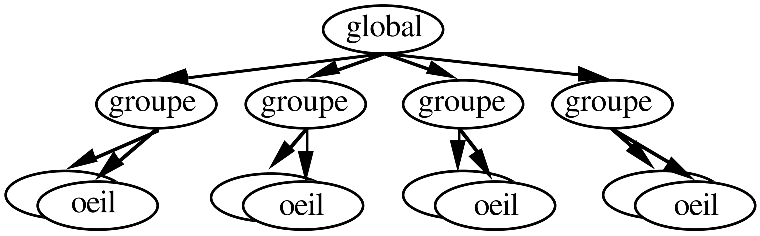


figure Méta-structure

Une évaluation statique :

La position structurée comme sur l'exemple précédent est évaluée statiquement suivant les règles suivantes.

Évaluation d'un oeil

Les éléments terminaux de l'exemple (les yeux) sont évalués suivant trois valeurs statiques: 'gagné', 'perdu' et 'autre'. Un oeil est gagné si 3 des 4 intersections diagonales sont de sa couleur. Il est 'perdu' si les 4 intersections diagonales sont occupées sans être dans le cas gagné. Il est 'autre' dans les autres cas. La figure *Méta-yeux-1* donne des exemples d'yeux.

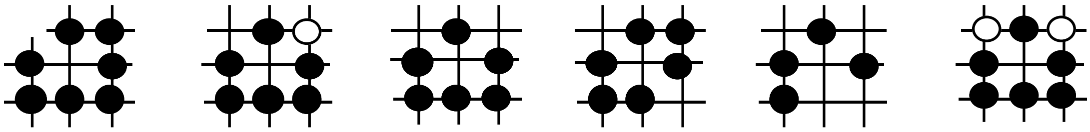


figure Méta-yeux-1

Évaluation d'un groupe

L'évaluation d'un groupe dépend de l'état de ses deux yeux. Si le groupe a deux yeux 'gagné', l'évaluation du groupe est 'gagné'. Si le groupe a un oeil 'perdu', l'évaluation du groupe est 'perdu'. Dans les autres cas, l'évaluation du groupe est 'autre'.

Évaluation globale

L'évaluation de la position globale pour une couleur C dépend des contributions des groupes. Si un groupe est 'gagné' (respectivement 'perdu') et de la couleur C (resp. autre couleur que C) ou 'perdu' (resp. 'gagné') et de l'autre couleur que C (resp. couleur C), sa contribution est +1 (resp. -1). L'évaluation de la position globale pour une couleur C est égale à la somme des contributions de chaque groupe.

Qualification dynamique :

L'approche-zéro :

Cette approche est celle que tout le monde utilise pour montrer que la force brute utilisée aux Échecs ne peut pas marcher au jeu de Go. Cette approche n'utilise pas la structure statique du jeu ni le paragraphe précédent. Il y a un jeu unique, le jeu global. La fonction d'évaluation est simple : +1 (respectivement -1) si l'intersection est occupée par une pierre noire (resp. blanche) ou vide mais entourée par des voisines noires (resp. blanches). Les coups générés sont ceux de la règle du jeu (en gros les intersections vides). La taille de l'arbre du jeu global est environ 361! sur un 19-19. La figure *dramatique* explique ce fait sur un goban 9-9...

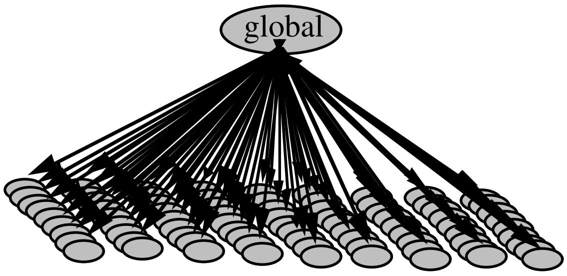


figure dramatique

L'approche-une :

Cette approche utilise un peu la structure du jeu. Il y a toujours un jeu unique, le jeu global. La fonction d'évaluation est définie à partir de la structure : +1 (respectivement -1) si l'intersection est occupée par un groupe noir (resp. blanc) "vivant". Les coups générés sont ceux générés par les feuilles terminales de la structure. La figure *Méta-approche-un* illustre ce fait.

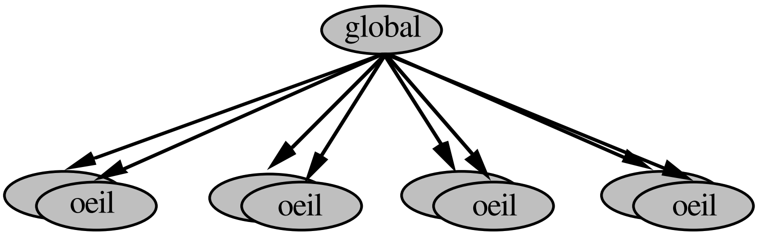


figure Méta-approche-un

Si chaque oeil peut engendrer en moyenne 4 coups, la taille de l'arbre est environ $(4.2.4)! \approx 32!$ Ce qui est beaucoup. Aux erreurs près liées à la structure conceptuelle, le résultat du jeu global est précis.

L'approche-deux :

Cette approche utilise la structure du jeu. Il y a un jeu global décomposé en 4 jeux du groupe vivant, eux-mêmes décomposés en 2 jeux de l'oeil. La structure ludique ou dynamique suit exactement la structure statique ou conceptuelle. La figure *Méta-approche-deux* illustre ce fait.

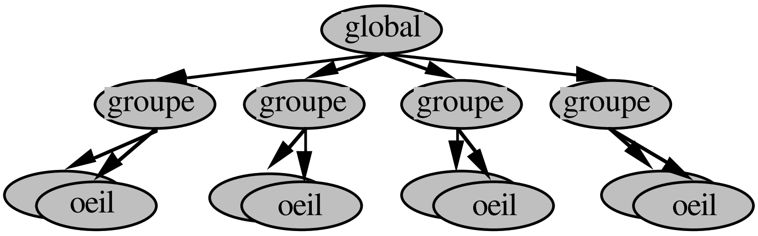


figure Méta-approche-deux

Chaque jeu non terminal est donc décomposé en sous-jeux. Pour connaître l'état d'un jeu terminal, on calcule en effectuant une recherche arborescente. Pour générer les coups d'un jeu non terminal, si des sous-jeux sont *, on prend les coups conseillés par ces sous-jeux, sinon on utilise des *règles de recomposition dynamiques* qui permettent de connaître l'état du jeu à partir de l'état des sous-jeux. Ces règles dynamiques ressemblent aux règles de recomposition statiques citées dans la paragraphe 'évaluation d'un groupe' ci-dessus. Si le groupe a deux jeux de l'œil '>', l'évaluation du groupe est 'gagné'. Si le groupe a un jeu de l'œil '<', l'évaluation du groupe est 'perdu'. Dans les autres cas, l'évaluation du groupe est 'autre'. La figure *Méta-règles-dynamiques* donne des exemples où ces règles s'appliquent.

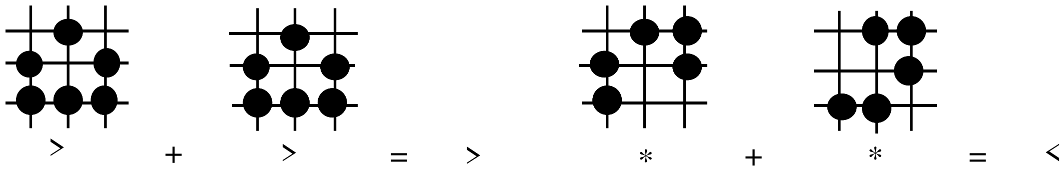


figure Méta-règles-dynamiques

Le coût est 4.2.4! ≈ 8.4! (4 parce que 4 groupes, 2 parce que 2 yeux par groupe et 4! si l'on considère un arbre de profondeur 4 pour le calcul du jeu de l'œil). Les règles de recomposition sont approximatives car elles sont *fausses* si les sous-jeux sont *dépendants*.

L'approche-trois :

Cette approche utilise la structure du jeu. Il y a toujours un jeu global décomposé en 4 jeux du groupe vivant, eux-mêmes décomposés en 2 jeux de l'oeil. La structure ludique ou dynamique suit encore exactement la structure statique ou conceptuelle. Cette approche diffère de l'approche deux car elle lève le cas ou les règles dynamiques de recomposition sont fausses comme sur les exemples de la figure *Méta-règles-dynamiques-fausses*.

figure Méta-règles-dynamiques-fausses

Pour connaître l'état d'un jeu terminal, on calcule en effectuant une recherche arborescente. Pour engendrer les coups d'un jeu non terminal, si des sous-jeux sont *, on prend les coups conseillés par ces sous-jeux, sinon si les sous-jeux sont indépendants, on utilise les règles de recomposition de l'approche-deux, s'ils sont dépendants on prend les doubles menaces sur chaque sous-jeu - on les engendre en prenant l'intersection des ensembles d'intersections vides sur lesquelles reposent les deux sous-jeux. La figure *Méta-approche-trois* explique graphiquement ce fait avec des doubles flèches qui indiquent que les sous-jeux sont dépendants.

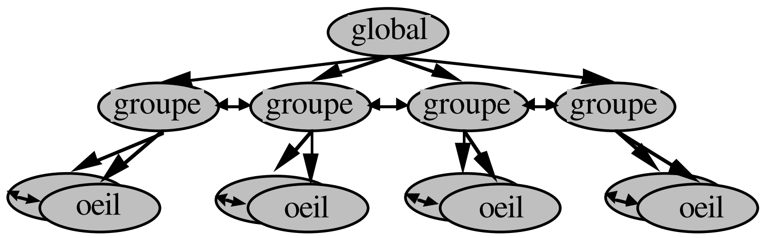


figure Méta-approche-trois

Le coût est intermédiaire entre celui de l'approche-une et celui de l'approche-deux Le résultat est beaucoup plus précis. La difficulté réside dans la mise en œuvre.

Conclusion :

Pour discuter des avantages et des inconvénients de ces approches, il est nécessaire de rappeler les approximations faites.

Simplifier pour modéliser

Toute structure conceptuelle proposée est une simplification de la réalité.

Simplifier pour expliquer

La structure conceptuelle proposée est un exemple simple présentant notre réflexion.

Ne pas scier la branche de l'arbre sur laquelle on repose

Le fait de jouer des coups pour calculer les différents états dynamiques de sous-jeux supposent que les coups joués ne modifient pas la structure conceptuelle statique. La réalité est plus complexe : chaque coup modifie la structure statique. En toute rigueur à chaque coup joué, la position change, il faudrait vérifier que la structure est toujours identique. Notre exemple ne met pas en évidence cette difficulté.

Jouer des coups ou jouer à des jeux ?

Oublions l'approche-zéro qui n'est présentée que pour servir de repère.

La particularité de l'approche-une est de calculer le jeu global en jouant des coups engendrés par les feuilles de la structure conceptuelle et de l'évaluer globalement ensuite. On pourrait qualifier cette approche de primaire et bête au sens où l'on agit et on regarde après. On joue des coups.

L'approche-deux joue sur le mécanisme d'abstraction aux deux sens du terme. Premièrement, on cache les coups du niveau du dessous, au niveau considéré. Deuxièmement, on conceptualise un peu plus à chaque passage d'un niveau au niveau supérieur. On pourrait qualifier cette approche de secondaire au sens où l'on réfléchit avant d'agir. On joue à des jeux.

La troisième approche se situe entre la première et la seconde. Le problème est de savoir quelle approche utiliser.

Si les sous-jeux d'un niveau sont indépendants, on a vu que les règles de recomposition dynamiques s'appliquent et que l'approche secondaire est recommandée.

Si les sous-jeux sont "très" dépendants, c'est-à-dire si leurs empreintes sont très similaires, l'approche deux est carrément fausse. Il faut l'abandonner. L'approche trois oblige à effectuer n calculs (un pour chaque sous-jeu) pour déterminer son résultat (>, <, *, 0). Comme les empreintes sont très similaires, on fait n calculs très similaires. Ensuite, on n'utilise pas les règles dynamiques de recomposition, donc on fait encore un calcul supplémentaire encore très similaire. L'empreinte est le lieu d'au moins n+1 calculs. Il est alors légitime d'abandonner aussi cette approche mixte et d'utiliser l'approche primaire qui ne fera qu'un seul calcul.

Si les sous-jeux sont "peu" dépendants, l'approche trois est recommandée.

Où est le métajeu ?

Il est possible de donner une définition faible du métajeu : **un métajeu est un jeu où l'on joue à des (sous-)jeux** au lieu de jouer des coups. L'approche deux définit des **métajeux** en ce sens. Un métajeu en ce sens nécessite d'avoir une description extérieure de ses sous-jeux.

Il est aussi possible de donner une définition plus forte du métajeu : **un métajeu est un jeu où l'on joue à des sous-jeux en interagissant sur la méthode de résolution de chacun des sous-jeux**, l'approche trois définit des **métajeux** en ce sens.

Dans INDIGO, l'approche deux est utilisée. L'approche trois est une perspective pour améliorer notre programme. Comprendre mieux les mécanismes des différentes approches pour évaluer une position au Go passe par une modélisation de plus en plus fine de la notion de métajeu, ce qui est une illustration intéressante pour le Méta.

Les patterns et les niveaux Méta

Cadre général :

Pour programmer le go, les programmeurs utilisent des règles dont la particularité est de contenir des "patterns" dans leur partie gauche. Pour simplifier, nous dirons qu'une règle est de la forme indiquée par la figure *Méta-règle-initiale* .

figure Méta-règle-initiale

Un des problèmes de la programmation du jeu de Go est d'enrichir automatiquement une base de règles.

Suivant le degré de formalisation atteint par un programmeur, il pourra enrichir la base de règles à la main ou bien, si le formalisme est clair, enrichir automatiquement la base de règles. Dans notre cas, nous avons plusieurs sous-classes de règles qui dépendent du point de vue de cette sous-classe : (liaison, vitalité, opposition, chaîne, zone, ...). Certaines sous-classes voient leur formalisme se figer et le nombre d'instances augmenter de façon telle qu'il n'est pas envisageable de maintenir la base de règles de cette sous-classe à la main. Cela nous est arrivé lorsque nous avions observé que notre modèle sur les groupes n'avait pas le comportement que nous attendions. Pour des groupes de taille petite (une pierre ou deux), le modèle donnait des comportements inattendus. Par exemple, le modèle ordonnait de sauver systématiquement des groupes en atari. Or il existe des coups au Go qui sont joués dans un esprit de sacrifice, surtout quand ces coups font partie de chaînes ou groupes de pierres de taille 1 ou 2, et d'autre coups où il ne faut absolument pas sacrifier de pierres, les pierres de coupe. Nous avons décidé de cacher le comportement de ces petits groupes, que nous appelions des "miettes", dans des patterns et qu'une miette incluse dans un pattern aurait le comportement spécifié par la règle. Le problème était alors d'engendrer automatiquement ces règles.

La génération comprend trois phases : à partir d'anciennes règles créer de nouvelles règles par spécialisation de la partie gauche, puis avec ces parties gauches calculer la partie droite par un moyen quelconque, enfin généraliser les règles obtenues. Dans ce paragraphe nous nous intéressons uniquement à la phase de spécialisation des patterns.

La spécialisation des patterns :

Pour manipuler les règles, un programme de Go doit utiliser une forme plus explicite de règle que celle de la figure *Méta-règle-initiale*. Certaines intersections n'ont pas d'importance. On utilise un symbole supplémentaire '#' qui peut valoir tous les autres. La règle exemple de la figure *Méta-règle-initiale* peut être exprimée explicitement comme sur la figure *Méta-règle-explicite* .

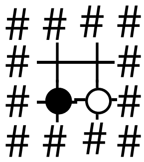
SI  ALORS blablabla

figure Méta-règle-explicite

Pour spécialiser des parties gauches de règles, un moyen simple est de dilater le pattern. C'est ce qu'exprime la métarègle de la figure *Méta-métarègle-dilatation* .

SI une prémisse de règle R contient un pattern P ALORS spécialiser la règle R avec les patterns dilatés de P

figure Méta-métarègle-dilatation

Cette métarègle de spécialisation appliquée à la règle de la figure *Méta-règle-initiale* crée des règles filles. La figure *Méta-règle-engendrée-1* montre un exemple de règle fille.

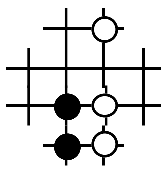
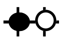
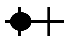
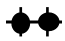
SI  ALORS blablabla

figure Méta-règle-engendrée-1

Cette métarègle est malheureusement trop générale. Pour la règle exemple, elle génère $3^8 \approx 10^4$ règles filles. Ceci n'est pas pratiquement envisageable. Surtout, la plupart de ces règles filles ne sont pas pertinentes.

Dans le cas de la base de règles associée à la gestion des "miettes", il était plus intéressant de ne spécialiser ou dilater que dans certaines directions privilégiées selon l'aspect du pattern. Par exemple, dans le cas des miettes, il était plus intéressant de dilater le pattern près des zones du pattern qui contiennent de l'information conflictuelle que près des zones non conflictuelles. Par exemple :  est plus conflictuel que  qui est plus conflictuel que .

Au lieu d'utiliser la métarègle de la figure *Méta-métarègle-dilatation* , nous avons préféré utiliser la métarègle de spécialisation par dilatations par morceaux de la figure *Méta-métarègle-dilatation-par-morceaux* .

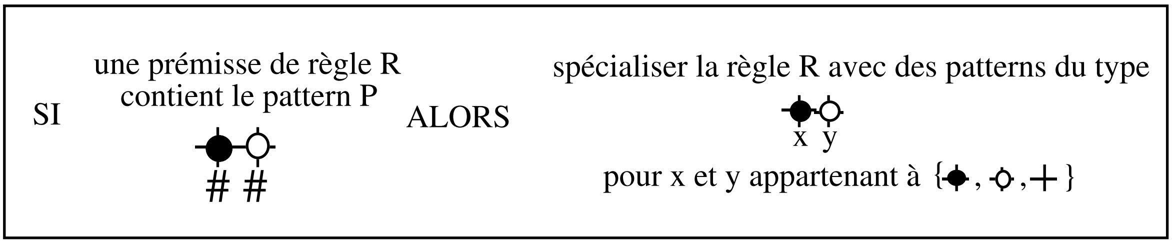


figure Méta-métarègle-dilatation-par-morceaux

La figure *Méta-règle-engendrée-2* montre une règle générée par la métarègle de la figure *Méta-métarègle-dilatation-par morceaux* à partir de la règle de la figure *Méta-règle-initiale* . Il est fondamental de remarquer que pour exprimer ces métarègles de façon déclarative, il est nécessaire d'utiliser la notation explicite du niveau de dessous, c'est-à-dire d'utiliser le symbole #. D'autre part, deux autres symboles ont été introduits, x et y qui sont des variables.

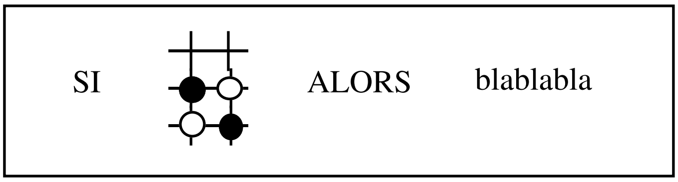


figure Méta-règle-engendrée-2

Dans ce cas, la métarègle engendre 9 règles filles, ce qui est gérable. Nous avons engendré automatiquement une centaine de règles destinées à la gestion des "miettes".

Les niveaux Méta :

Nous pouvons représenter avec des niveaux Méta, les différents aspects de la modélisation des patterns du Go.

Niveau 2	● ○ + # x y	métaprogrammation
Niveau 1	● ○ + #	programmation
Niveau 0	● ○ +	goban, partie, jeu
Niveau -1	● ○	bol

Le niveau 0 est évident pour tous les joueurs de Go : au cours d'une partie, les intersections vides du goban + peuvent devenir noires ● ou blanches ○. Ce que l'on peut écrire :

$$+ \approx \{ \bullet \mid \circ \}.$$

Le niveau 1 est évident pour les programmeurs de Go : au cours de l'algorithme de pattern-matching, les symboles # d'un pattern peuvent devenir +, ● ou ○. Ce que l'on peut écrire :

$$\# \approx \{ + \mid \bullet \mid \circ \}.$$

Le niveau 2 est évident pour ceux qui utilisent la métaprogrammation : au cours de l'algorithme de génération de patterns, les symboles x ou y des métapatterns peuvent devenir ◆ , ◇,, + ou #. Ce que l'on peut écrire :

x ≈ { ◆ | ◇ | + | # }.

Ainsi, à chaque niveau Méta supplémentaire, un ou plusieurs symboles sont rajoutés par rapport au niveau inférieur. Les nouveaux symboles d'un niveau, x, # et +, sont des variables qui peuvent prendre pour valeur, les symboles du niveau inférieur. Pour être cohérent, il est donc nécessaire de créer un niveau -1 contenant les deux seuls symboles • et ∘ . Ce niveau est un niveau "Antiméta" où n'existent que des pierres inanimées, • ou ∘, dans des bols[1].

A propos de * et de +

La différence entre le niveau 0 et le niveau -1 est l'arrivée du goban avec ses intersections vides +. Avec lui, il est possible de *jouer* au Go. Sans goban, pas de *jeu* de Go. Conway écrit :

{ ◆ | ◇ } = *.

Ainsi, il existe une analogie entre * et + que l'on peut écrire :

* ≈ +

A propos de # et +

Un joueur de Go, qui donne une explication à un autre joueur de Go faisant intervenir des "formes" ou patterns, utilise un goban pour poser la forme explicative. Par exemple, un joueur qui veut parler de la forme de la coupe posera des pierres sur le goban et cela donnera physiquement :

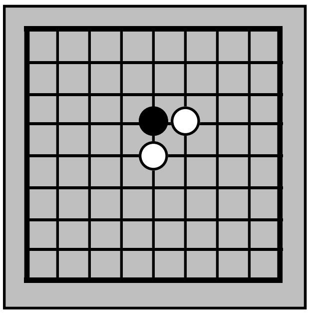


figure Méta-goban

Le joueur de Go qui écoute, ne sait pas s'il doit comprendre la forme en un sens assez strict :

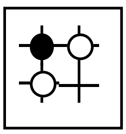


figure Méta-strict-implicite

ou bien en un sens plus large :

[1]Le nom du réceptacle où les pierres sont rangées.

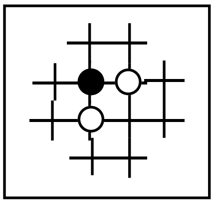


figure Méta-large-implicite

Jusqu'où s'arrête la "forme" (le pattern) ?

En utilisant la notation explicite du niveau Méta 1, et pour exprimer la même idée mais autrement, il ne sait pas s'il doit comprendre en un sens strict :

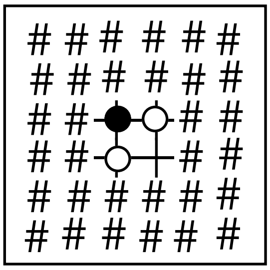


figure Méta-strict-explicite

ou bien :

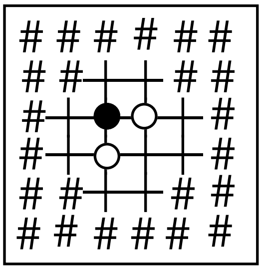


figure Méta-large-explicite

Jusqu'où doit-on mettre des + et où doit-on mettre des # ?

Le programmeur essaie de trouver une réponse là où il n'est peut être pas nécessaire d'en donner. En effet, le joueur humain n'est pas gêné par ce flou. En effet, les joueurs se comprennent sans avoir besoin de préciser la frontière exacte de la forme. Pour un humain, les niveaux Méta ne sont pas séparés nettement. La frontière entre les niveaux Méta 0 et 1 est vague. L'exemple présenté montre la similarité qui existe entre + et # pour un humain. Ce que l'on peut écrire :

+ ~ #.

Bibliographie

[Pitrat 1990] - J. Pitrat - Métaconnaissance - Hermès 1990

[Pell 1992] - B. Pell - Metagame : A New Challenge for Games and Learning - Heuristic programming in Artificial Intelligence 3 : The Third Computer Olympiad - H.J. van den Herik & L.V. Allis, editors - Ellis Horwood - 1992

Logique floue

Introduction

Un joueur de Go possède une vision floue et vague des objets reconnus sur le goban et effectue un raisonnement flou sur ces objets. INDIGO n'utilise pas de logique floue à proprement parler mais il nous paraît intéressant de présenter ce qui ce rapproche de la logique floue. La reconnaissance des territoires utilise de la morphologie mathématique que nous appelons floue car c'est une numérisation de la morphologie mathématique symbolique. D'autre part, le raisonnement ludique d'INDIGO utilise des symboles "imprécis". Ces symboles permettent de limiter les calculs coûteux pour ne pas perdre de temps. Ils permettent aussi de réduire le nombre de cas possibles lorsque des synthèses sur des résultats de jeux sont effectuées.

Le but de ce paragraphe est de montrer comment la logique floue intervient dans INDIGO dans le raisonnement ludique. La vision floue du goban a été présentée au chapitre sur l'utilisation de domaines voisins dans la partie 2.

Le raisonnement ludique

Nous recensons actuellement 3 cas "flous" dans INDIGO.

Le symbole ?

Les contraintes pratiques de temps de réponse de INDIGO nous ont poussé à simplifier INDIGO. Lorsque INDIGO effectue des calculs, l'arbre de recherche est limité en taille. Si une recherche n'aboutit pas, le résultat du calcul est '**?**'.

Le symbole

La complexité théorique des résultats de jeux, synthèses de résultats de sous-jeux[1] est trop lourde à gérer. Nous avons caché cette complexité en créant le symbole "fuzzy" ou [2].

La différence entre * et est que :

*** = { > | < }** (Si on joue, on gagne)

alors que l'on peut avoir :

= { | } (Si on joue, on peut ne pas savoir ce qui va se passer...)

Le symbole * pour les combats[3]

Lorsque des combats interviennent entre deux groupes voisins, INDIGO utilise des heuristiques sur le nombre de libertés des groupes pour savoir qui gagne le combat et connaître l'état de l'interaction ennemie sans poser physiquement les pierres.

---

[1]Les attributs d'un groupe comme inimitié et santé sont des synthèses "complexes".
[2]Voir en particulier la synthèse produite dans l'inimitié d'un groupe.
[3]Voir la synthèse d'une interaction ennemie.

Par exemple, en simplifiant, ces heuristiques sont du type :

si le groupe a N libertés son état est *,
si il a plus de N libertés, son état est >,
si il a moins de N libertés, son état est <.

Ces heuristiques marchent bien tant que les libertés sont claires. Des libertés peuvent ne pas être claires pour deux raisons. Certaines libertés sont de **fausses libertés** au sens où l'adversaire peut les supprimer en gardant l'initiative : atari ou menace de déconnexion. D'autres libertés sont des **libertés inaccessibles en un coup**, il faut des coups d'approche pour y accéder. Le nombre de libertés reconnues statiquement doit être ajusté par ces deux considérations optimiste et pessimiste. Comme le principe d'utilisation de ces heuristiques est de ne pas poser physiquement les pierres pour connaître le résultat, il nous a fallu trouver un moyen de les utiliser lorsque les libertés ne sont pas claires. Le principe a été d'**élargir le nombre N à un intervalle flou [N-A,N+B]** où A est le nombre de fausses libertés et B la somme, pour chaque liberté inaccessible en un coup, des coups supplémentaires pour accéder à cette liberté.

L'heuristique est devenue:
si le groupe possède n libertés avec N-A≤n≤N+B sont état est *,
si il possède plus de N+B libertés, son état est >,
si il possède moins de N-A libertés, son état est <.

Cela permet à INDIGO d'élargir le nombre de cas où il considère un combat comme *. Pour INDIGO, mieux vaut rajouter un coup inutile de temps en temps et ne pas perdre de combats gagnants que de ne jamais rajouter de coup et perdre des combats gagnants.

L'utilisation de ces symboles

L'utilisation de * est claire car, si on joue, on gagne, et si l'autre joue, on perd. Le niveau global d'INDIGO peut quantifier le nombre de points rapportés par un coup. Par contre ? et  sont plus difficilement utilisables par le niveau global. Il manque dans INDIGO quelque chose qui permette d'y voir plus clair dans . ? est inutilisable, on sait que l'on ne sait pas calculer, c'est tout. Le lecteur intéressé pourra se reporter à la théorie des jeux de Conway [Conway 1982] où l'auteur donne plusieurs exemples à ce propos sur des jeux autres que le Go :

page 16: what is a game ?
page 30: positive, negative, zero, fuzzy
page 32: how big is a fuzzy game ?
page 71: a closer look at the stars
page 119: switch games
page 120: cashing cheques
page 122: temperature
chapitre 6: the heat of the battle

<u>Bibliographie</u>

[Conway 1982] - J. Conway, E.R. Berlekamp, R. Guy - Winnings ways - tome 1 & 2 - Academic press - 1982

L'Intelligence Artificielle Distribuée

Introduction

Le but de ce paragraphe est de montrer comment INDIGO interprète la position *IAD-exemple* que nous avons présentée au paragraphe Méthode - Utilisation des domaines voisins - Intelligence Artificielle Distribuée.

Comment INDIGO interprète une position caractéristique de la nature chaotique du jeu de Go

Sur la position *IAD-exemple-A-B*,

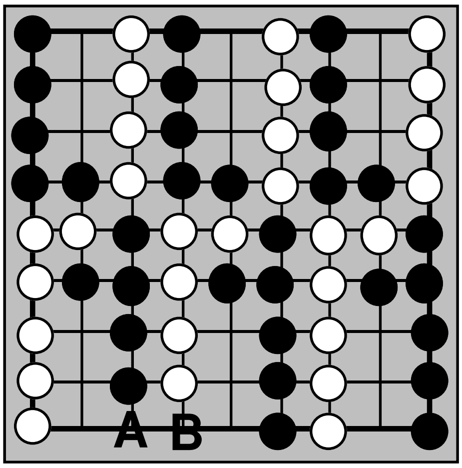


figure IAD-exemple-A-B

INDIGO reconnaît les groupes comme indiqué sur la figure *IAD-exemple-groupe.*

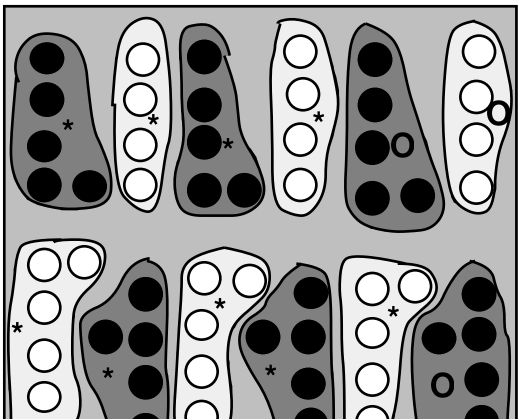


figure IAD-exemple-groupe

En effet, sur la figure *IAD-exemple-A-B*, les connexions * en A ou B pour Blanc ou Noir entraînent que les groupes concernés par ces connexions ont une amitié * et donc une santé *. Les groupes voisins de ces groupes ont une inimitié * par le fait d'être voisin d'un groupe avec une amitié * et ont donc une santé *. Tous ces groupes conseillent de jouer A ou B selon que c'est à Noir ou Blanc de jouer. Les groupes restants ont une inimitié 0 et donc une santé 0.

Noir joue le premier

Si c'est à Noir de jouer, INDIGO joue en 1 sur la figure *IAD-noir*.

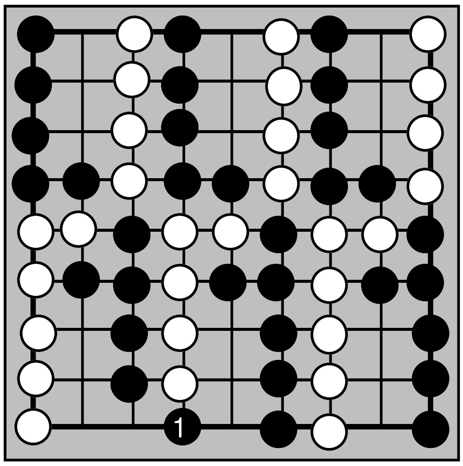


figure IAD-noir

Les groupes reconnus sont ceux indiqués par la figure *IAD-noir-groupe*.

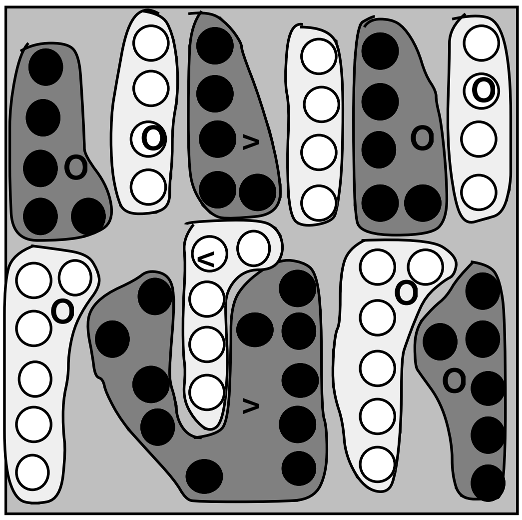


figure IAD-noir-groupe

Un groupe blanc a une santé <. Les groupes noirs voisins de ce groupe ont une santé >. Une catastrophe se produit. Les autres groupes ont une santé 0 car ils possèdent tous au moins une interaction ennemie 0 et donc une inimitié 0 et donc une santé 0.

Une catastrophe se produit avec création d'un territoire noir comme indiqué sur la figure *IAD-noir-1-catastrophe*.

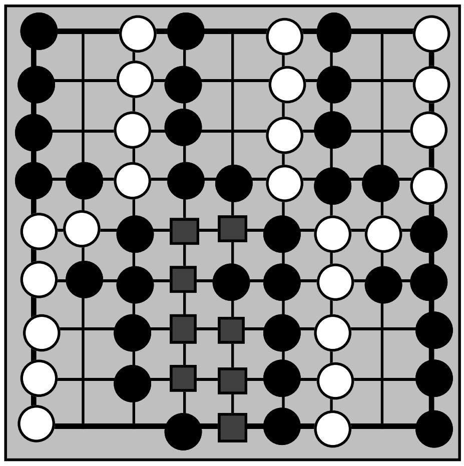


figure IAD-noir-1-catastrophe

Les groupes sont alors ceux de la figure *IAD-noir-groupe-1-catastrophe*.

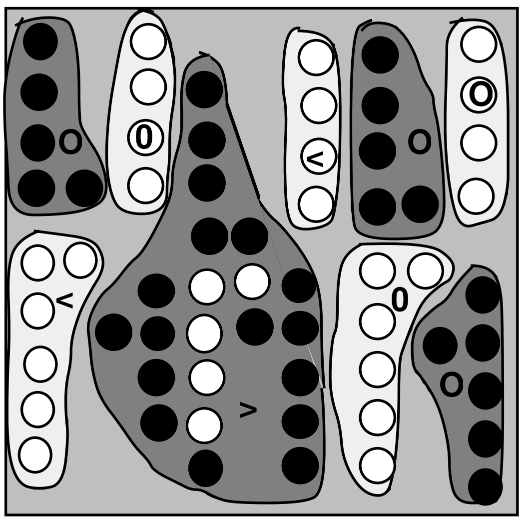


figure IAD-noir-groupe-1-catastrophe

Le gros groupe noir a une santé >. Les groupes voisins blancs ont une santé 0 ou < selon que des libertés communes existent ou non.

Deux groupes ont donc une santé <. Deux catastrophes se produisent avec création de deux territoires noirs supplémentaires comme indiqué sur la figure *IAD-noir-2-catastrophe*.

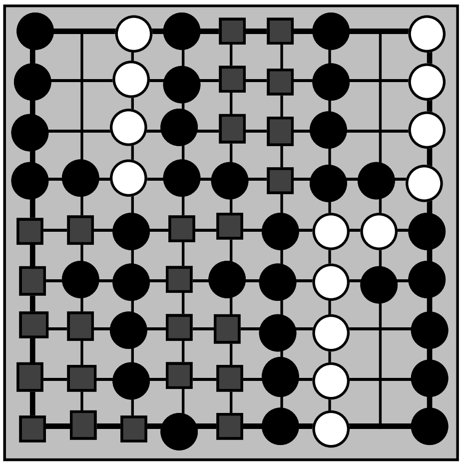


figure IAD-noir-2-catastrophe

Les groupes sont alors ceux de la figure *IAD-noir-groupe-2-catastrophe*.

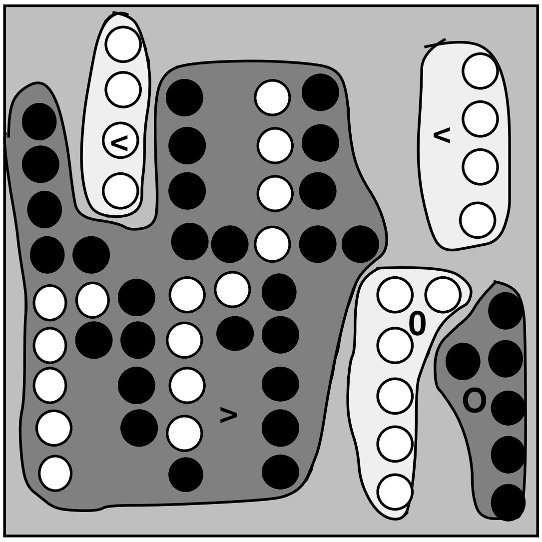


figure IAD-noir-groupe-2-catastrophe

Le gros groupe noir devient encore plus gros. Il a une santé >. Les groupes voisins blancs ont une santé < ou 0 selon que des libertés communes existent ou non.

Deux groupes ont donc une santé <. Deux catastrophes se produisent encore avec création de deux territoires noirs supplémentaires comme indiqué sur la figure *IAD-noir-3-catastrophe*.

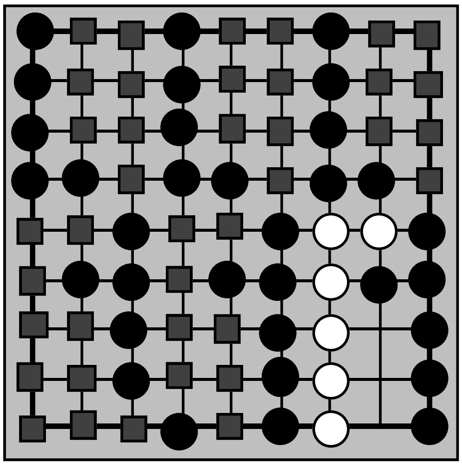


figure IAD-noir-3-catastrophe

Les groupes sont alors ceux de la figure *IAD-noir-groupe-3-catastrophe*.

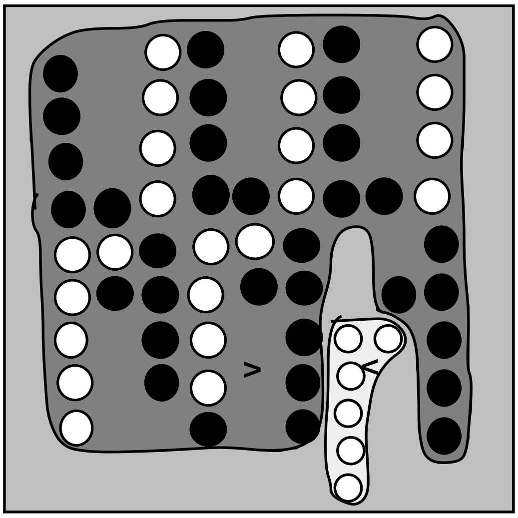


figure IAD-noir-groupe-3-catastrophe

Le gros groupe noir devient toujours plus gros. Il a une santé toujours >. Le groupe blanc voisin une santé <.

Une dernière catastrophe se produit avec un dernier territoire noir comme indiqué sur la figure *IAD-noir-tout*.

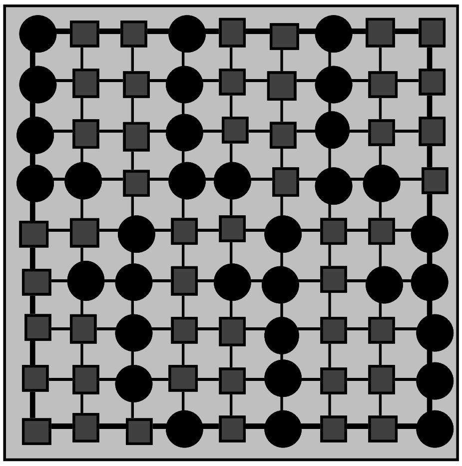


figure IAD-noir-tout

Tout le goban appartient à Noir.

Blanc joue le premier

Si c'est à Blanc de jouer sur la position de la figure *IAD-exemple-A-B*, INDIGO joue comme indiqué par la figure *IAD-blanc*.

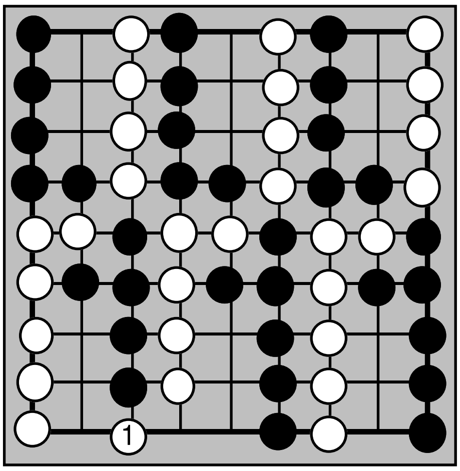


figure IAD-blanc

De manière analogue à ce qui se passe quand Noir joue le premier, INDIGO interprète la position de la figure *IAD-blanc* avec une avalanche de catastrophes qui aboutissent à l'interprétation de la figure *IAD-blanc-tout* .

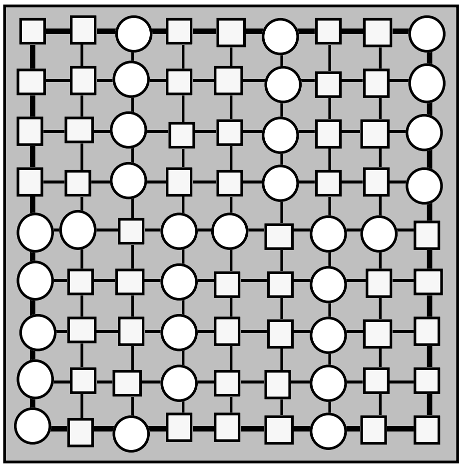


figure IAD-blanc-tout

Tout le goban appartient à Blanc.

Conclusion

Finalement, nous pouvons faire les remarques suivantes :

L'état du jeu associé à la position *IAD-exemple-A-B* est { +81 | -81 }.

Un petit changement sur la description physique - poser une pierre - peut produire de gros changements sur la description conceptuelle (162 points ici).

Les changements sont reconnus par une avalanche de catastrophes.

Une catastrophe est locale. Elle se produit si des contraintes ne sont plus respectées au niveau d'une entité locale, le groupe.

Le groupe possède un comportement local simple qui peut produire de gros changements au niveau global.

L'intelligence de INDIGO est distribuée sur chacun des groupes.

Le jeu de Go se prête idéalement à une modélisation distribuée.

En particulier, il se prêterait à une implémentation sur machine parallèle.

CONCLUSION

Dans cette dernière partie, nous évaluons si le but annoncé en première partie du document a été atteint. D'abord, **élaborer un modèle cognitif du joueur de Go**, ensuite **expliciter des connaissances non conscientes**. Nous donnons aussi deux résultats importants liés à notre démarche pratique : d'abord, **développer un programme qui joue au Go**, ensuite **illustrer des domaines de l'IA par le jeu de Go**.

Élaborer un modèle cognitif

A la fin de la première partie, nous avons défini un modèle cognitif en suivant sans problème les critères de la "source", du "quoi", du "comment" et du "combien" car ils se rapportent tous à l'être humain. Par contre, le critère du "support" a été discuté. D'une part, nous voulions **valider le modèle** suivant les quatre premiers critères. La machine devenait alors un outil indispensable. Mais, le modèle cognitif allait-il donc être un **modèle computationnel** ? Nous avons décidé d'avoir un modèle computationnel pour valider pratiquement notre travail **et** une **correspondance** entre le modèle computationnel et le modèle cognitif pour évaluer celui-ci. Nous savions que nous ne pourrions pas valider le modèle cognitif en rapport avec la nature de son support, le cerveau humain. En utilisant une machine, nous avons donc eu l'*avantage* de *valider empiriquement* notre modèle cognitif mais nous avons eu l'*inconvénient* de ne pas le valider en liaison avec la *nature* du cerveau. Ceci étant précisé, nous pouvons évaluer notre modèle suivant les critères de la "source", du "quoi", du "comment" et du "combien".

Le critère "source" est totalement atteint : la source d'inspiration pour construire le modèle cognitif a été le **joueur de Go humain**.

Le critère du "quoi" est presque totalement atteint. *Totalement* parce que le programme INDIGO, implémentation du modèle, joue une partie de Go complète comme un être humain. *Presque* parce que son niveau n'est situé qu'**entre 15ème et 20ème kyu**, c'est-à-dire comparable à celui d'un joueur faible et en dessous de notre ambition de départ.

Le critère du "comment" est presque totalement atteint. *Totalement* parce que le programme INDIGO est capable de **donner les raisons d'un coup** en suivant les concepts reconnus à chaque niveau, les règles déclenchées et les résultats des sous-jeux calculés pour jouer ce coup. Tous ces concepts correspondent à des concepts utilisés par des joueurs humains. *Presque* parce qu'il n'existe pas de module dans le programme qui traduise la trace en langage naturel, comparable aux **verbalisations** des joueurs humains.

Le critère du "combien" est respecté : il joue un coup **sur 19-19 entre 10" et 1'**.

Globalement, nous estimons que **le but d'élaborer un modèle cognitif est atteint**.

Illustrer des domaines de l'IA

Notre travail s'est déroulé dans un laboratoire d'IA, le LAFORIA, et nous avons été largement influencé par les différents courants existant en IA.

En retour, la présentation de notre modèle au travers de ces domaines permet de les illustrer.

Le **Méta** avec la décomposition du jeu global en sous-jeux que l'on peut considérer comme des **métajeux** sous certaines conditions. L'adaptation au jeu de Go de la théorie des jeux de Conway va de paire avec la définition d'un concept de métajeu.

L'**IAD** avec l'architecture de notre modèle qui suit la **nature distribuée** du jeu de Go. La perspective d'utiliser une machine à **architecture parallèle** est réelle. La vie et la mort des groupes de pierres est faite en utilisant le concept d'**interaction** entre les groupes. Le programme sait interpréter certaines **positions chaotiques** où l'effet global est le seul résultat de l'application de règles simples sur l'interaction entre les groupes voisins.

La **Logique Floue** avec la **reconnaissance floue** des territoires et l'**imprécision modélisée** pour cacher la complexité inhérente au jeu de Go.

La **Vision** avec la nature visuelle du jeu de Go. Nous avons utilisé la **morphologie mathématique** pour reconnaître les territoires et la théorie computationnelle de la **vision de Marr** pour structurer notre modèle.

Ces illustrations serviront les chercheurs en IA en quête d'un domaine complexe d'application de leurs outils.

Développer un programme de Go

Le programme INDIGO, implémentation du modèle computationel, est conçu suivant un formalisme **objet**, il utilise les **jeux de Conway**, il est structuré en **niveaux**.

Le niveau inférieur comprend deux jeux de base : le jeu de la chaîne et le jeu de l'intersection. Le niveau élémentaire comprend une dizaine de jeux ou actions élémentaires et environ 500 règles. Le niveau itératif est important car il construit les objets nécessaires pour évaluer une position globale : **les groupes** avec une **reconnaissance statique de la vie et de la mort** en fonction d'une propriété interne mais aussi et surtout en fonction de propriétés d'**interaction** avec le voisinage des groupes, les **territoires** reconnus avec des outils de **morphologie mathématique**. Le niveau global est simple : il est basé sur l'**intelligence distribuée** sur les objets du niveau itératif. Enfin un mécanisme d'**incrémentalité** basée sur l'empreinte des objets posés sur le goban a été mis au point.

Le niveau atteint est très acceptable pour un programme de Go. Mais il nous faut encore travailler pour obtenir un programme qui atteigne un niveau proche de 10ème kyu, comparable à celui du meilleur programme, Goliath.

Expliciter des connaissances non conscientes

Pour expliciter des connaissances non conscientes du joueur de Go humain, nous avons utilisé **une méthode, basée sur l'implémentation du modèle cognitif, dans laquelle la machine joue le rôle de révélateur**.

Nous avons établi une correspondance entre les concepts du modèle computationnel et les connaissances d'un joueur humain suivant le degré de conscience des connaissances humaines. Pour classer les connaissances suivant trois classes (**conscient**, **conscientisable**, **intuitif**) nous nous sommes largement basé sur la présence ou non-présence de termes correspondant à ces connaissances dans les **verbalisations** de joueurs humains.

Les concepts conscients identifiés sont :
le **jeu**,
la **stratégie**.

Les concepts conscientisables sont :
certains **jeux élémentaires cachés**,
la **connexion**,
la **fraction**,
l'interaction.

Les concepts intuitifs sont :
le **multi-échelle**,
l'**incrémentalité**,
l'**intérieur-extérieur**,
des concepts **topologiques**
ou **morphologiques**.

La large proportion de concepts intuitifs, ou au moins non conscients, dans notre liste montre que l'étude du fonctionnement du système cognitif de l'être humain, basée sur le jeu de Go et la machine, doit être poursuivie. Notre étude est un pas vers la compréhension du fonctionnement du système cognitif humain.

ANNEXES

Les annexes comprennent :

Annexe A : La règle du jeu de Go

Annexe B : Parties jouées par INDIGO

Annexe C : L'interface homme-machine

Annexe D : Glossaire

Annexe E : Quelques fichiers C++

ANNEXE A : LA RÈGLE DU JEU DE GO

Ce chapitre décrit plusieurs aspects du jeu de Go et de la programmation du jeu de Go liés à la **règle du jeu**.

Premièrement, nous donnons une règle du jeu de Go destinée aux lecteurs ne connaissant pas le jeu de Go.

Deuxièmement, nous montrons que la règle du jeu présentée au lecteur humain ne suffit pas pour faire jouer simplement un programme de Go. Nous montrons qu'une règle du jeu plus simple, plus fondamentale et sans implicite est nécessaire.

Aperçu des règles du jeu de go au travers d'une partie

Dans ce paragraphe, nous présentons le jeu de Go au travers d'un exemple de partie. Ensuite nous donnons les règles du jeu sous forme plus précise.

Une partie de Go

Une partie se joue à deux joueurs, **Noir** et **Blanc**. Une partie se joue sur un damier appelé **goban** constitué de 19 lignes et 19 colonnes qui se coupent suivant 361 **intersections**. Pour simplifier, nous donnons un exemple de partie jouée sur 9-9 avec 81 intersections seulement. En général, un débutant apprend à jouer sur un 9-9, puis sur un 13-13 et enfin sur un 19-19. Au début de la partie le goban est **vide** comme sur la figure *RDJ-0* :

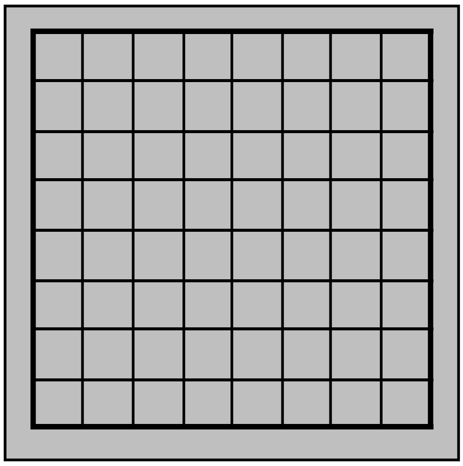


figure RDJ-0

Noir commence en posant une **pierre** noire sur une intersection, par exemple avec ❶ comme sur la figure *RDJ-1*:

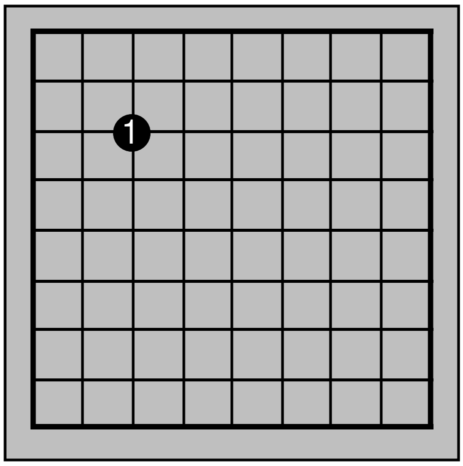


figure RDJ-1

La pierre posée sur le goban est voisine de 4 intersections vides: les **libertés** qui lui permettent de respirer. On ne peut pas jouer de coup sur une intersection où l'on n'a pas de liberté. Une pierre posée sur une intersection du bord (respectivement coin) du goban possède 3 (resp. 2) libertés.

Noir et Blanc posent à tour de rôle des pierres sur le goban. A partir de la position de la figure *RDJ-1*, la partie peut par exemple se poursuivre comme sur la figure *RDJ-2*:

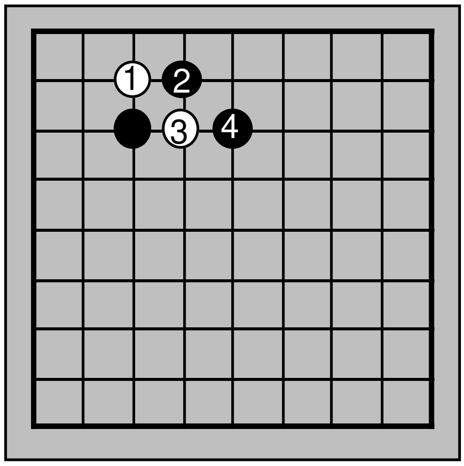


figure RDJ-2

Ici, ❹ a fait **atari** à ③ Cela signifie que ③ ne possède plus qu'une seule liberté. A partir de cette position, deux cas sont possibles.

Blanc peut défendre sa pierre avec ① :

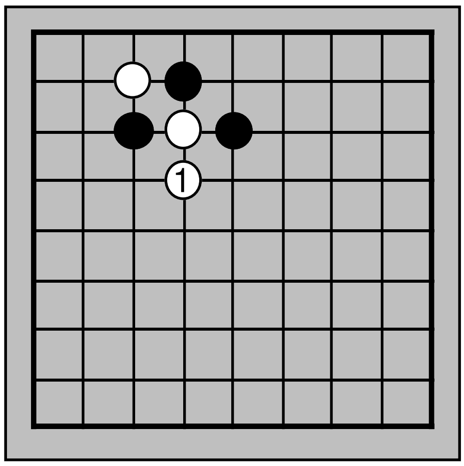


figure RDJ-2a

Sur la figure *RDJ-2a*, Blanc a constitué une **chaîne** de 2 pierres. Le sort des 2 pierres est lié à celui de la chaîne, elle mettent en commun leur libertés. Sur la figure *RDJ-2a*, la chaîne blanche a 3 libertés.

On peut attaquer les pierres adverses en les capturant, c’est-à-dire en supprimant la dernière liberté d’une chaîne de pierres. Par exemple, si Blanc joue ailleurs, Noir peut **capturer** la pierre blanche comme sur la figure *RDJ-2b* :

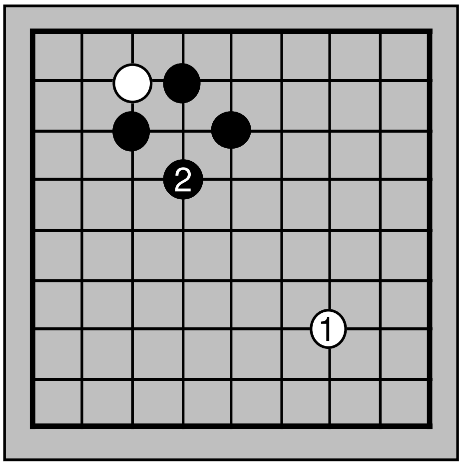


figure RDJ-2b

Les pierres capturées sont enlevées du goban.

La partie se poursuit, le goban se remplit de pierres comme par exemple sur la figure *RDJ-3* :

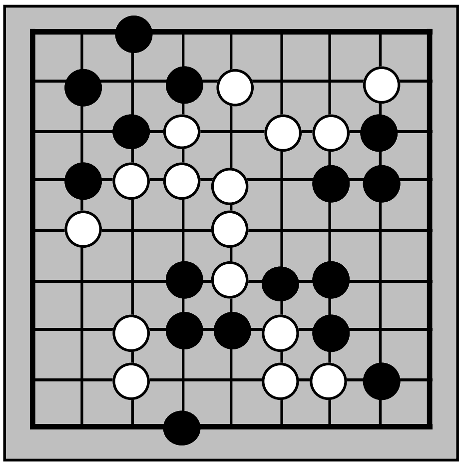


figure RDJ-3

La partie se termine quand Noir et Blanc **passent**. A ce moment là, on compte le **score**. On attribue un **point** par intersection **contrôlée** au joueur qui la contrôle. Une intersection est contrôlée par une couleur si elle est occupée par une pierre de la couleur ou si, étant vide, lorsqu'une pierre de l'autre couleur y sera posée, elle sera capturée. Le joueur qui a le plus de points **gagne** la partie.

Par exemple, sur la figure *RDJ-4*. toutes les intersections appartiennent enfin soit à Noir, soit à Blanc :

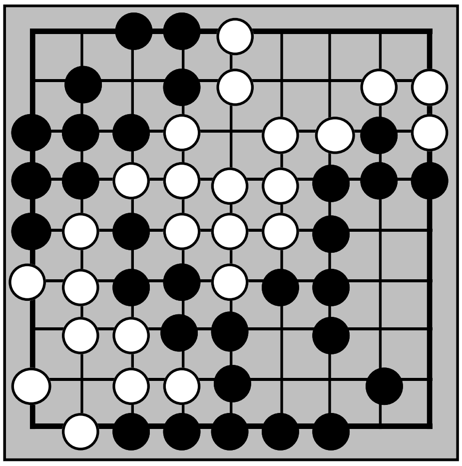


figure RDJ-4

Les deux joueurs passent et comptent les points. La figure *RDJ-5* montre quel joueur contrôle quelles intersections. Les ● et les ■ désignent les intersections contrôlées par Noir et les ○ et les □ désignent des intersections contrôlées par Blanc.

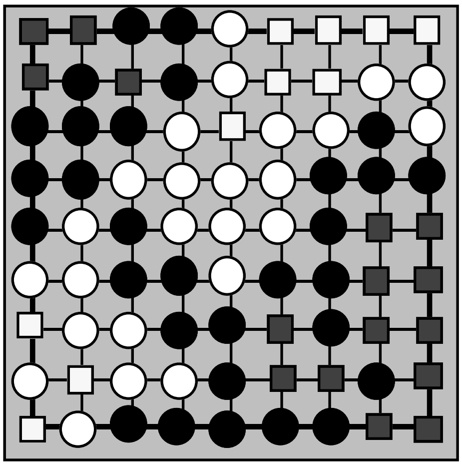


figure RDJ-5

Ici, Noir contrôle 46 intersections et Blanc contrôle 35 intersections. Noir gagne de 11 points.

Au moment où les deux joueurs passent, ils peuvent ne pas être d'accord sur l'appartenance des intersections à l'un et à l'autre. Si c'est le cas, ils continuent de jouer jusqu'à être **d'accord**. Par exemple, si Blanc conteste l'appartenance d'intersections à Noir, il joue dedans comme dans la figure *RDJ-6* :

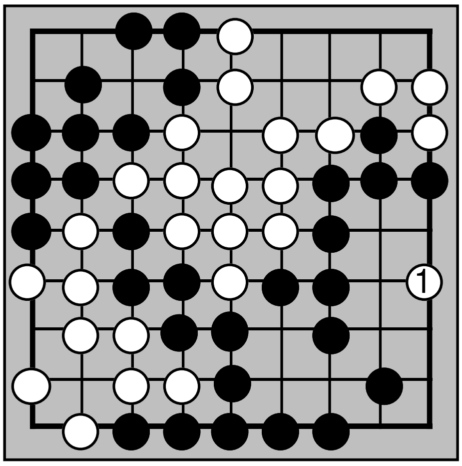


figure RDJ-6

Noir peut capturer la pierre blanche, même si Blanc se défend. La figure *RDJ-6a* montre un exemple de séquence qui capture la pierre blanche.

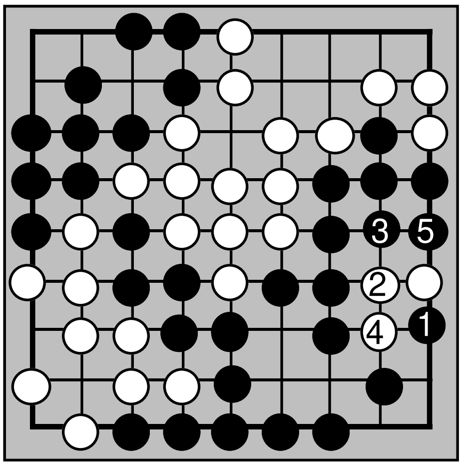


figure RDJ-6a

Ainsi, Noir prouve explicitement qu'il contrôle cette intersection.

Mais Blanc peut contester à nouveau comme sur la figure *RDJ-6b* :

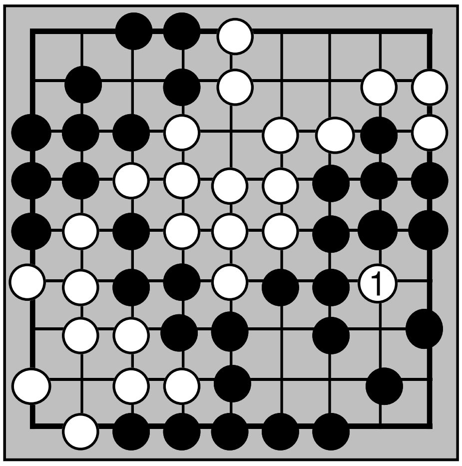


figure RDJ-6b

Noir peut prouver que cette intersection lui appartient. La figure *RDJ-6c* montre un exemple de séquence qui capture la pierre blanche :

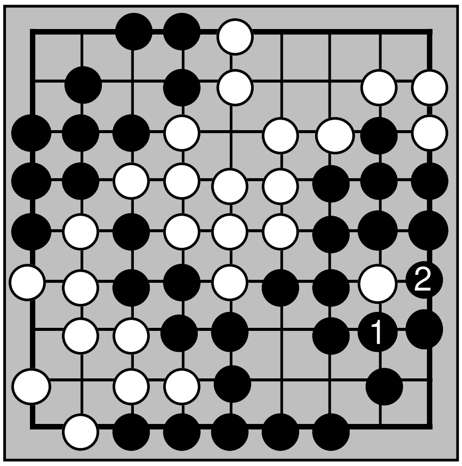


figure RDJ-6c

Comme Blanc, Noir peut contester l'appartenance des intersections de Blanc. La figure *RDJ-7* montre une séquence de contestation vaine des intersections blanches par Noir.

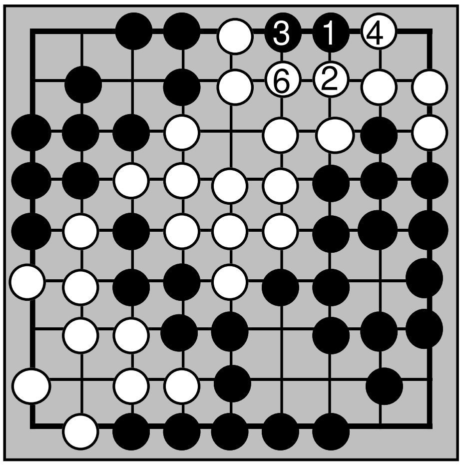


figure RDJ-7

Toutes les intersections peuvent être contestées tant que le contrôle n'est pas suffisamment explicite. La figure *RDJ-8*. montre une position où le contrôle est explicite.

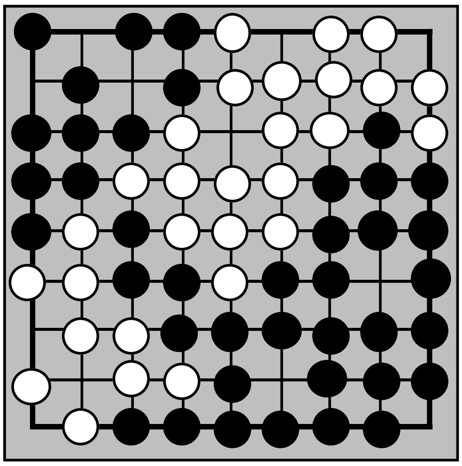


figure RDJ-8

La figure *RDJ-9* montre l'appartenance des intersections à Blanc et à Noir avec les mêmes notations que celles de la figure *RDJ-5*.

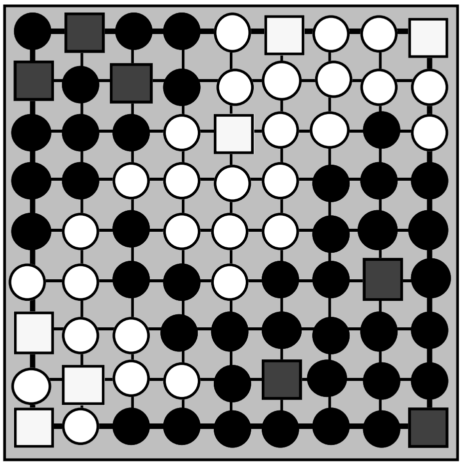


figure RDJ-9

Sur cette figure, aucune ambiguïté n'est possible. Le contrôle des intersections est explicite. Noir possède les intersections ● et ■ et Blanc possède les intersections ○ et □. Le lecteur vérifiera que Noir gagne toujours de 11 points comme sur la figure *RDJ-5*. Le déroulement de la partie entre les figures *RDJ-5* et *RDJ-9* n'a évidemment pas lieu si les deux joueurs sont d'accord dès la figure *RDJ-4*.

Des précisions

Le lecteur a senti que les règles du jeu de Go présentées au travers de la partie sur 9-9 sont simples.

Nous pouvons les résumer ainsi :

(Rdj1) On joue à tour de rôle sur les intersections,
(Rdj2) On peut capturer l'adversaire en supprimant la dernière liberté d'une chaîne,
(Rdj3) On peut défendre ses pierres en les connectant sous forme de chaîne,
(Rdj4) On gagne en contrôlant plus d'espace que l'adversaire.

Nous avons omis volontairement la règle des situations répétitives:

(Rdj5) On ne peut pas jouer un coup qui donne une position déjà rencontrée dans la partie.

En pratique, la situation du "ko" est la situation répétitive la plus fréquente.

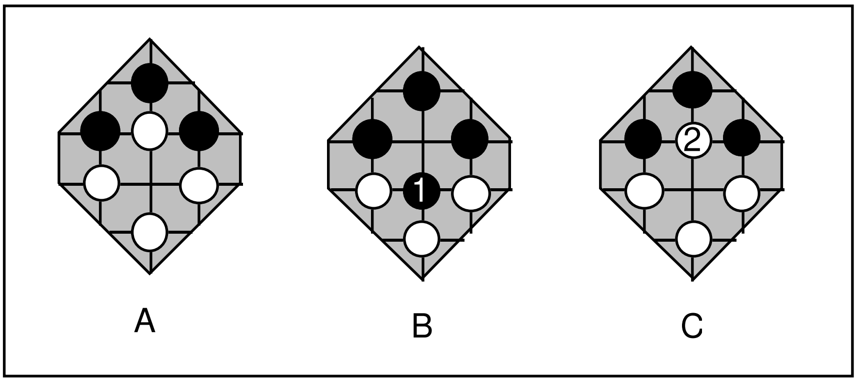


figure RDJ-ko

Dans la position A de la figure *RDJ-ko*, si Noir prend la pierre blanche comme sur la position B, Blanc n'a pas le droit de recapturer au coup suivant : la position C serait une répétition de la position A. Pour jouer en 2 et reprendre la pierre noire 1, Blanc doit d'abord jouer ailleurs, ce qui fait changer l'état du reste du goban. Si Noir joue aussi ailleurs, l'ensemble du goban aura changé et Blanc aura le droit de jouer en 2.

La règle de fin de partie:

(Rdj6) La partie s'arrête quand les deux joueurs passent et sont d'accord sur le contrôle de l'espace. Si les joueurs ne sont pas d'accord, ils continuent de jouer jusqu'à être d'accord.

Les règles du jeu que nous avons présentées sont le produit direct de notre travail sur la modélisation du jeu de Go.

Une règle du jeu pour ordinateur

Introduction

Au début de notre travail, nous n'aurions pas présenté les règles du jeu à un non joueur de Go comme nous venons de le faire au paragraphe précédent. Étant joueur de Go, les règles du jeu de Go que nous utilisons nous paraissaient évidentes et ne pas être la cause de problèmes. En fait, les règles du jeu que nous utilisons contiennent beaucoup d'**implicites**. Ces implicites nous ont posé problème. Voulant obtenir des résultats pratiques, nous avons commencé notre travail en écrivant un programme "n'ayant aucune connaissance sur la façon de jouer au Go, mais ayant la connaissance des règles du jeu". Pour la suite nous appelons ce programme le **zéro-programme**.

Nous pensions que cela serait facile d'écrire le zéro-programme puisque le jeu de Go est connu pour la "simplicité de ses règles". Le zéro-programme peut jouer sur toutes les intersections validées par la règle du jeu ou passer. Nous nous sommes aperçu alors que les règles du jeu Rdj1, Rdj2, Rdj3, Rdj5 étaient effectivement simples et claires. On peut les spécifier, puis les implémenter. Par contre, les règles Rdj4 et Rdj6 nous ont posé problèmes. Rdj4 pour le **contrôle de l'espace** et Rdj6 pour **l'accord de fin de partie**.

Qu'est-ce que le contrôle de l'espace ?

Le contrôle de l'espace par "territoire"

Pourquoi un joueur de Go dit-il que la partie est terminée sur la figure *RDJ-4* ? Comment spécifier le contrôle de l'espace au zéro-programme pour qu'il dise que tout l'espace est contrôlé sur la position de la figure *RDJ-4* ?

Nous avons vu que le joueur démontre par la pratique (cf. figures *RDJ-6* à *RDJ-8*) qu'il contrôle un espace. Il appelle d'ailleurs ce type d'espace un **territoire**. Un territoire est un espace contrôlé par un joueur au sens où si l'adversaire essaie de s'installer dedans, il sera capturé. Il n'est pas naturel de spécifier au zéro-programme ce qu'est un territoire. Si on le fait, il faut lui dire comment capturer un "groupe" envahisseur, ce qu'est un "groupe", comment il peut se défendre pour ne pas être capturé, comment un "groupe" peut être "mort" ou "vivant", qu'est-ce qu'un "œil". Cela devient très vite très compliqué. Si la notion de contrôle de l'espace comprend celle de territoire alors il est faux de dire que les règles du jeu de Go sont simples ! La notion de territoire est trop complexe pour être spécifiée au zéro-programme. La règle du contrôle de l'espace est une règle implicite qui dépend de l'expérience de chaque joueur. Les joueurs de Go, nous y compris, oublient cela lorsqu'ils enseignent le Go à un débutant.

Le contrôle physique de l'espace

Délaissant la confusion qui règne autour de la notion de territoire, nous sommes parti en sens inverse. Nous avons cherché une notion de bas niveau de contrôle de l'espace pour le zéro-programme. Au plus bas niveau, nous pouvons spécifier le contrôle de l'espace en disant qu'une intersection est contrôlée par une couleur si **une pierre de cette couleur est posée sur l'intersection** comme le montre la figure *RDJ-contrôle-espace-1*.

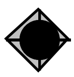


figure RDJ-contrôle-espace-1

Avec cette connaissance, le zéro-programme est capable de jouer une partie contre lui-même. Lorsque deux passes consécutifs sont enregistrés, la partie s'arrête et on compte les points. On donne donc un point à une couleur par intersection occupée par une pierre de cette couleur. Le programme est d'accord avec lui-même. C'est simple et clair.

En pratique, la probabilité de passer étant égale à 1/N où N est le nombre de coups permis, une partie s'arrête bien avant que le goban soit totalement contrôlé. En théorie, avec cette notion bas niveau du contrôle de l'espace, il est impossible que le goban soit totalement contrôlé. Aucune pierre n'aurait de liberté. **Ce zéro-programme est-il vraiment un programme de Go ?** Quel est son niveau ? Combien de pierres de handicap doit-il prendre pour gagner une partie contre un joueur classé ? Ces questions sont amusantes. Nous laissons le lecteur intéressé y répondre. Nous pensons qu'il est plus intéressant de trouver un moyen plus évolué pour définir le contrôle de l'espace au zéro-programme pour que celui-ci "joue au Go".

Le contrôle de l'espace par voisinage direct

Le contrôle de l'espace par voisinage direct que nous pouvons spécifier au zéro-programme est le suivant :

Pour Noir, ne pas jouer au centre de la forme de la figure *RDJ-contrôle-espace-2*: car c'est une intersection contrôlée par Noir. C'est un **point** Noir.

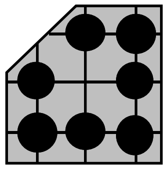


figure RDJ-contrôle-espace-2

Avec la connaissance de la figure *RDJ-contrôle-espace-2* les choses se passent beaucoup mieux. Le zéro-programme, ne passe pas tant que le goban n'est pas complètement contrôlé. Il ne se bouche pas ses points et la partie s'arrête naturellement sur une position du type *RDJ-8*. **La fin de partie correspond au contrôle total du goban**. Avec cette notion de contrôle de l'espace nous avons un vrai zéro-programme : il joue à un jeu qui suit les règles du jeu de Go sans avoir de connaissances sur la manière de jouer.

C'est le fait d'écrire un programme de Go qui nous a rendu conscient de la nature floue et vague du contrôle de l'espace que nous utilisons lorsque nous jouons au Go et qui nous a fait expliciter un contrôle de l'espace clair et net de bas niveau. **Avec ce contrôle de l'espace de bas niveau, nous enseignons le Go aux débutants beaucoup plus facilement**. Le débutant n'a plus de problème d'amorçage du type : "Qu'est qu'un territoire ? Qu'est-ce qui se passe si on vient chez moi ? Qu'est-ce qu'un groupe ? Pourquoi c'est capturé ? Pourquoi c'est mort ? Pourquoi c'est un œil ? Pourquoi c'est encerclé ici et pas là ?". Il part directement de très bas et reconstruit lui-même ce qu'il veut bien reconstruire. En particulier, il comprend sans difficulté pourquoi la position de la figure *RDJ-8* est totalement contrôlée. En quelques parties, il a construit suffisamment de repères pour savoir que la position de la figure *RDJ-4* est elle aussi totalement contrôlée.

Que signifie être d'accord ?

A la fin de la partie, le zéro-programme qui joue contre lui-même est d'accord avec lui-même sur le contrôle du goban ! C'est une bonne chose, mais que se passe-t-il si deux programmes différents jouent l'un contre l'autre ?

Pour clarifier la discussion, supposons que **alpha** soit le zéro-programme qui joue avec le contrôle de l'espace par voisinage direct et que **bêta** soit le zéro-programme qui joue avec le contrôle physique de l'espace. Il faut noter que alpha est très évolué par rapport à bêta car il ne passe que lorsque le contrôle de l'espace coïncide avec tout le goban. En pratique, alpha gagne "toutes" les parties contre bêta en contrôlant "tout" le goban. Lorsque les deux programmes passent, alpha pense avoir toutes les intersections vides ou occupées physiquement par lui et bêta pense que alpha ne contrôle que les intersections occupées physiquement. Ils ne sont pas d'accord. Comment faire pour les départager ? Bêta n'a plus de coups permis possibles sauf passer et alpha passe sous peine de se boucher des intersections vides. Le consensus est impossible. Plus généralement, **les programmes ne sont pas assez évolués pour se mettre d'accord[1], il faut une règle pour les départager**. Actuellement, la règle Ing pour programmes [Ing] fait foi : un point par intersection occupée, sinon au prorata des intersections voisines occupées. Dans ce cas, il ne fait pas bon avoir construit un grand territoire ou capturé un gros groupe mort encore physiquement présent sur le goban ! D'autres règles simples existent [Michel 1983] [Yoshikawa 1989] et sont utilisables pour définir un zéro-programme. Un historique et un état de l'art sur les différentes versions des règles du jeu de Go existantes à travers le monde a également été fait dans les annexes de [Berlekamp & Wolfe 1994]. Ce qui montre les difficultés posées par l'axiomatisation des règles du jeu de Go.

Conclusion

Nous avons montré comment la définition d'une règle du jeu de Go pour un zéro-programme était plus complexe que prévu. Nous avons fait une marche arrière : nous sommes parti de la règle du jeu de Go, évoluée et floue, que tout le monde connaît et utilise comme une évidence. Nous avons creusé vers le bas pour découvrir une règle du jeu claire et de bas niveau. Nous utilisons cette règle du jeu dans INDIGO et nous l'enseignons désormais avec succès aux débutants. La marche arrière effectuée sur le cas de la règle du jeu est significative du reste de notre recherche sur le jeu de Go. Dans cette recherche, nous sommes parti de nos connaissances conscientes, floues et évoluées pour écrire un programme. Nous avons découvert des concepts plus précis et de bas niveau dont nous n'avions pas conscience au départ. On dit que les règles du jeu de Go sont simples. C'est vrai, à condition de prendre les précautions énoncées ci-dessus.

Bibliographie

[Berlekamp & Wolfe 1994] - E.R. Berlekamp, D. Wolfe - Rules of Go : a Top-down Overview, Appendix A & Foundations of the Rules of Go, Appendix B - Mathematical Go Endgames - Nightmares for the Professional Go Player - Ishi Press International - San Jose, London, Tokyo - 1994

[Ing] - La règle Ing.

[Michel 1983] - J. Michel - Les règles du jeu de Go - Revue française de Go n°20 - 1983

[Yoshikawa 1989]- T. Yoshikawa - The most primitive go rule - Computer Go 13, winter 89-90

[1] Après deux passes consécutifs, tous les programmes que nous connaissons s'arrêtent et supposent que la partie est finie. Ils n'imaginent pas que l'adversaire ne soit pas d'accord avec eux sur l'interprétation du goban.

ANNEXE B : DES PARTIES JOUÉES PAR INDIGO

Dans cette annexe, nous présentons des parties jouées par INDIGO :

INDIGO - Many Faces of Go
INDIGO - Poka
INDIGO - Gogol (I)
INDIGO - Gogol (II)
INDIGO - Hugh

Dans ces parties, nous fournissons trois types de commentaires sur les coups joués :

en geneva, un commentaire d'INDIGO[1] ,

`en courier, le commentaire du concepteur d'INDIGO`;

en italique, un commentaire de joueur de Go.

Par exemple, le premier coup de la première partie contre Many Faces of Go pourrait donner les trois commentaires suivants:

Le coup 1 est joué pour occuper un espace vide.

`De manière générale, INDIGO stabilise ses groupes et occupe les espaces vides. Au premier coup, un seul espace vide existe, le goban entier, il est occupé par le jeu du remplissage du vide qui conseille de jouer dans un coin.`

Le coup 1 est très bon, d'ailleurs beaucoup de professionnels le jouent.

Dans les premières parties nous privilégions les commentaires d'INDIGO pour fixer les idées du lecteur. Ensuite, nous enlevons les commentaires (assez simplistes somme toute) d'INDIGO pour les remplacer peu à peu par une discussion entre un *joueur de Go fort* et `nous-même`.

[1]Notre programme ne dispose pas de module de langage naturel. Ces commentaires sont une traduction manuelle, faite par nous-même, de la trace sous une forme compréhensible du lecteur.

INDIGO - Many Faces of Go

Partie jouée le 8 Février 1994 sur IGS.
Noir : INDIGO
Blanc : Many Faces of Go

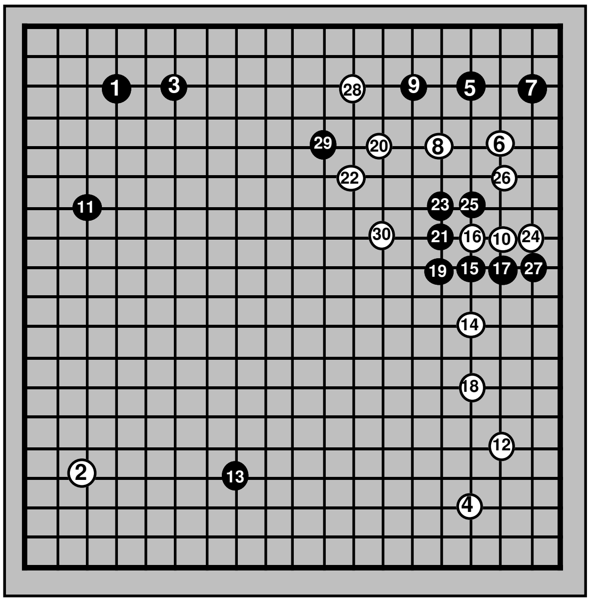


figure INDIGO-MFG-1-30

Le coup 1 est joué pour occuper un espace vide.

De manière générale, INDIGO stabilise ses groupes puis occupe les espaces vides. Au premier coup, aucun groupe n'existe et il y a un seul espace vide, le goban entier; il est occupé par le jeu du remplissage du vide. Celui-ci conseille de jouer sur la troisième ou quatrième ligne dans un coin ou sur les bords. Quand INDIGO dispose de plusieurs coups équivalents, il joue le coup le plus en haut à gauche. C'est pourquoi il joue dans le coin supérieur gauche.

Le coup 1 est très bon, d'ailleurs beaucoup de professionnels le jouent.

Le coup 3 est joué pour stabiliser le groupe noir 1 avec le jeu de la dilatation.

Le coup 3 est assez curieux.

Le coup 5 est joué pour occuper un espace vide.

La forme 1-3-5 est typique d'INDIGO dans les fusekis parallèles.

Les coups 7 et 9 sont joués pour stabiliser (resp. déstabiliser) le groupe noir 5 (resp. blanc 6) avec le jeu de l'opposition.

Il faut attaquer la pierre 2.

Les coups 11 et 13 sont joués pour occuper un espace vide.

Les coups 15, 17 et 19 sont joués pour déconnecter le groupe blanc 6-8-10 du groupe blanc 14.

Les coups 21, 23, 25 et 27 sont joués pour stabiliser (resp. déstabiliser) le groupe noir 15-17-19 (resp. blanc 6-8-10-20) avec le jeu de l'opposition.

Les coups d'INDIGO sont un peu lents et lourds par rapport à de ceux de Many Faces.

Le coup 29 est joué pour déconnecter le groupe blanc 6-8-10-20 du groupe blanc 28.

`Il s'agit d'une bogue dans INDIGO : une règle acquise à la main contenait une inversion du coup blanc et du coup noir !`

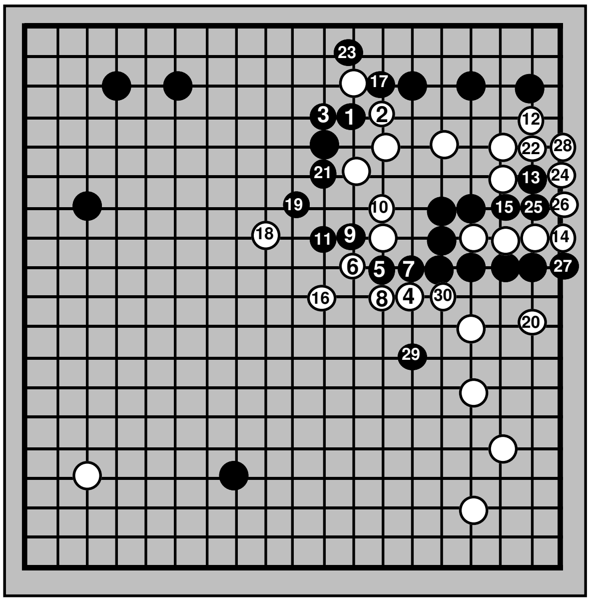


figure INDIGO-MFG-31-60

Le coup 1 est joué pour la même raison que le précédent.

1 est mauvais : il ne marche pas.

`INDIGO ne s'en aperçoit que lorsque 2 est joué.`

Le coup 3 est joué pour sauver la pierre 1 capturable par le jeu de la chaîne.

Le coup 5 est joué pour déconnecter le groupe blanc en haut à droite du groupe blanc 4.

Le coup 7 est joué pour connecter le groupe noir du milieu du bord de droite avec la pierre 5.

Le coup 9 est joué pour déconnecter les groupes blancs.

Le coup 11 est joué pour sauver la pierre 9 capturable par le jeu de la chaîne.

Le coup 13 est joué pour que le jeu de la séparation du territoire en 2 appliqué au petit territoire blanc en haut à droite soit négatif.

Le coup 15 est joué pour déconnecter les groupes blancs.

Le coup 17 est joué pour déconnecter la pierre blanche du bord haut du groupe blanc.

Le coup 19 est joué pour déconnecter la pierre 18 du groupe blanc.

Le coup 21 est joué pour connecter le groupe noir 9-11-19 avec le groupe noir 1-3.

Le coup 23 est joué pour capturer la pierre blanche avec le jeu de la chaîne et connecter le groupe noir 17 avec le groupe noir 1-3-9-11-19-21.

Pourquoi ne joue-t-il pas 23 en 25 ?

Le coup 23 implique le groupe 1-3-9-11-19-21, la pierre blanche, le groupe 17 et le groupe blanc 2-10-12-22, c'est-à-dire 23 pierres instables, il aussi implique le territoire controlé par ces groupes, c'est-à-dire 6 intersections supplémentaires; au total, le coup 23 implique 29 intersections instables pour INDIGO. Le coup 25 implique le groupe 5-7-15, la pierre noire 13, la pierre blanche 24, et le groupe blanc 2-10-12-22, c'est-à-dire 26 pierres instables, il n'implique pas de territoire; au total, le coup 25 implique 26 intersections instables pour INDIGO. C'est pourquoi INDIGO préfère 23 à 25.

24 est un très mauvais coup de Many Faces. Il aurait dû jouer en 25.

Le coup 25 est joué pour connecter le groupe noir 5-7-15 avec le groupe noir 13.

Le coup 27 est joué pour déconnecter le groupe blanc 2-10-12-14-22-24-26 du groupe blanc 20.

Heureusement, 27 est un atari et garde l'initiative.

Le coup 29 est joué pour déconnecter le groupe blanc 6-8-10-20 du groupe blanc 28.

L'échange 29 contre 30 est très mauvais. Il fait perdre une liberté au groupe noir.

`29 est encore un effet de la bogue précitée sur la figure précédente (au coup 29 aussi !)`

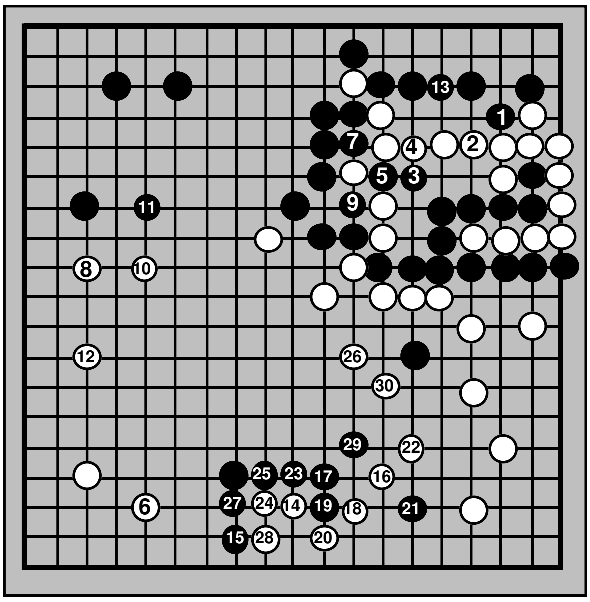


figure INDIGO-MFG-61-90

Le coup 1 est joué pour supprimer une liberté du groupe blanc, car un combat a été reconnu.

`INDIGO nomme combat un encerclement mutuel.`

Le coup 3 est joué pour déstabiliser le groupe blanc par le jeu de la dilatation.

3 est très bon. Bravo.

`INDIGO a de la chance que ce coup soit aussi un coup de déconnexion !`

Le coup 5 est joué pour déconnecter blanc.

Le coup 7 est joué pour déconnecter blanc.

7 est inutile.

Le coup 9 est joué pour capturer une pierre au jeu de la chaîne et connecter le groupe noir 3-5 avec le groupe noir 1.

9 est inutile.

```
INDIGO possède un horizon à profondeur 0 aux niveaux itératif et
global. Many Faces voit plus tôt que son groupe est mort et joue
deux coups gratuits ailleurs.
```

Le coup 11 est joué pour stabiliser (resp. déstabiliser) le groupe noir en haut à gauche (resp. blanc 8-10) avec le jeu de l'opposition.

13 est inutile, il enlève de l'aji, c'est tout.

```
La raison du coup 13 est inconnue, pas de chance.
```

Le coup 15 est joué pour déconnecter le groupe blanc 6 du groupe blanc 14.

Les coups 17, 19, 23, 25, 27 et 29 sont joués pour stabiliser (resp. déstabiliser) le groupe noir en bas (resp. blanc en bas) avec le jeu de l'opposition.

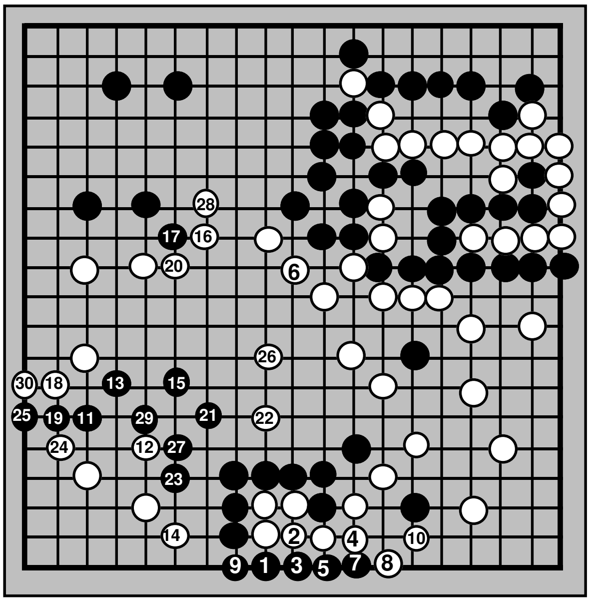


figure INDIGO-MFG-91-120

Les coups 1, 3, 5 et 7 sont joués pour stabiliser (resp. déstabiliser) le groupe noir en bas (resp. blanc en bas) avec le jeu de l'opposition.

5 est inutile.

```
1 est un atari, 3 est une menace de capture de la chaîne blanche
donc INDIGO garde l'initiative par chance. 5 est inutile. Parmi
les coups d'opposition, INDIGO ne distingue pas les coups urgents
des coups inutiles comme 5.
```

Les coups 11, 13, 15, 17, 19, 21 et 25 sont joués pour déconnecter les groupes blancs.

Le coup 23 est joué pour connecter le groupe noir 1-3-5-7-9 avec le groupe noir 11-13-15-19-21.

Les coups 27 et 29 sont joués pour stabiliser le groupe noir avec les jeux de la dilatation et de l'œil.

Noir a stabilisé son groupe du bas et annule le potentiel blanc. La partie est équilibrée.

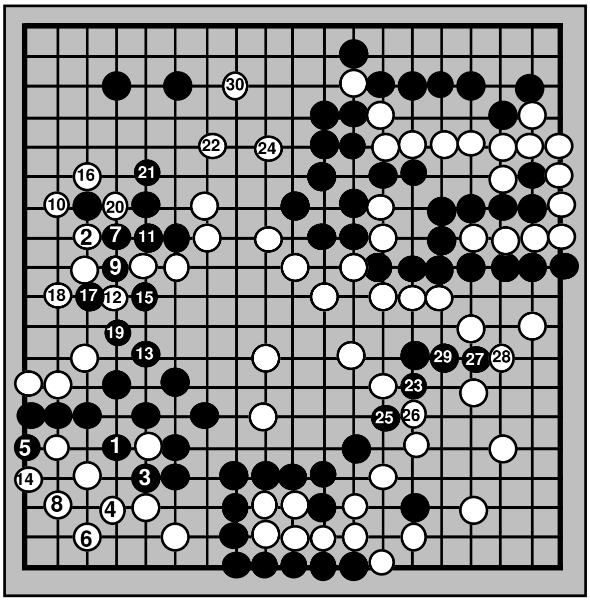


figure INDIGO-MFG-121-150

Le coup 1 est joué pour stabiliser le groupe noir avec les jeux de la dilatation et de l'œil.

Le coup 3 est joué pour stabiliser le groupe noir en capturant une pierre blanche avec le jeu de la chaîne.

Le coup 5 est joué pour stabiliser (resp. déstabiliser) le groupe noir 1-3 (resp. blanc 4) avec le jeu de l'opposition.

Les coups 7, 9 et 11 sont joués pour déstabiliser le groupe blanc 2 avec le jeu de l'opposition.

Le coup 13 est joué pour stabiliser le groupe noir avec les jeux de la dilatation et de l'œil.

Le coup 15 est joué pour déconnecter le groupe blanc 2-10-12 du groupe blanc central.

Le coup 17 est joué pour déconnecter le groupe blanc 2-10 du groupe blanc 12.

Le coup 21 est joué pour déconnecter le groupe blanc 2-10-16-18-20 du groupe blanc central.

Les coups 23, 25, 27 et 29 sont joués pour stabiliser la pierre noire .

Ce qui est résussi (!) : le groupe noir 23-27-29 est stable (!) mais mort (!) !!!

Quand INDIGO est victime de son horizon spatial limité, il défend des groupes indéfendables et perd des points. Il joue en sens inverse tant qu'il ne voit pas que le groupe est mort, ce qui est le cas après le coup 29.

Que s'est-il passé après ?

La partie a été interrompue par une coupure d'IGS quelques coups plus tard.

INDIGO a tué un gros groupe blanc qui équivaut au gros territoire blanc mais Many Faces of Go possède une vingtaine de points ailleurs que n'a pas INDIGO. Many faces of Go est donc en avance.

INDIGO - Poka

Partie jouée le 10 Février 1994 sur IGS.
Noir : INDIGO
Blanc : Poka

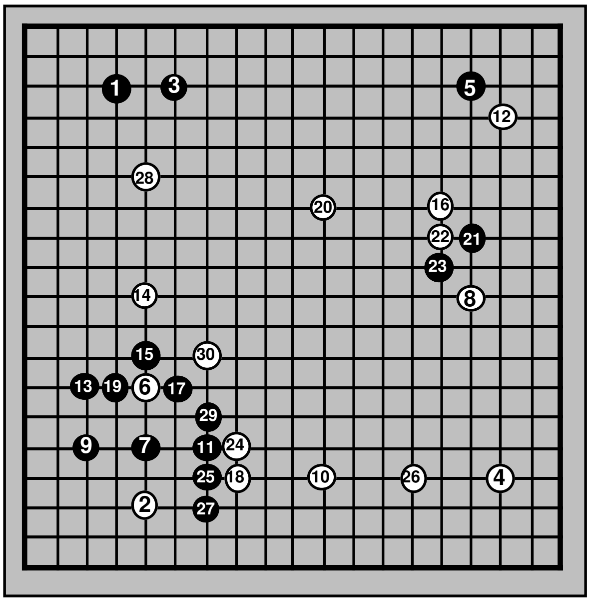


figure INDIGO-Poka-1-30

A nouveau, on retrouve la forme 1-3-5, typique d'INDIGO dans les fusekis parallèles. INDIGO ne dispose pas de tirage aléatoire entre des coups équivalents. Il peut rejouer la même partie si son adversaire fait de même.

Les coups 7, 9, et 11 sont joués pour déconnecter les groupes blancs 2 et 6.

Cela fait trois coups pour séparer deux pierres!

Heureusement, 7, 9 et 11 ont une efficacité globale car ils vont tout droit.

Le coup 13 est joué pour stabiliser (resp. déstabiliser) le groupe noir 7-9-11 (resp. blanc 6) .

Le coup 15 est joué pour déconnecter les groupes blancs 14 et 6.

Le coup 17 est joué pour stabiliser (resp. déstabiliser) le groupe noir 7-9-11-13-15 (resp. blanc 6).

Le coup 19 est joué pour capturer la pierre blanche 6 au jeu de la chaîne.

Cela fait quatre coups pour capturer une seule pierre !

INDIGO ne voit pas la rentabilité des coups. Il se jette sur ce qu'il trouve à manger. Globalement, ce n'est pas catastrophique car la force construite peut servir à attaquer le bord gauche où blanc est présent mais faible.

21 est un joli coup d'érosion du moyo blanc.

Le coup 21 est joué pour déconnecter le groupe blanc 16 du groupe blanc 8.

INDIGO ne connaît pas la notion d'érosion de moyo. Pour INDIGO, l'origine des coups est toujours ce qu'il connaît : la connexion de groupes (amitié), la dilatation de groupe (vacuité), l'opposition de deux groupes (inimitié) ou la base de vie d'un groupe ou le remplissage d'un espace vide. Toute ressemblance avec un objectif d'un fort joueur de Go existant est involontaire (malheureusement).

23 est absurde.

Le coup 23 est joué pour déconnecter le groupe blanc 16-22 du groupe blanc 8.

21 et 23 sont cohérents si l'on sait ce que cherche INDIGO : déconnecter Poka. Ils sont incohérents si l'on tente de les associer à un objectif évolué qui ne correspond pas aux objectifs d'INDIGO.

Les coups qui suivent 23 devraient tous être joués en 7 sur la figure INDIGO-Poka-31-60.

Pour INDIGO, le coup de connexion de la pierre 21 à la pierre 23 implique 2 intersections instables seulement, c'est pourquoi il n'y joue pas. C'est une grave erreur.

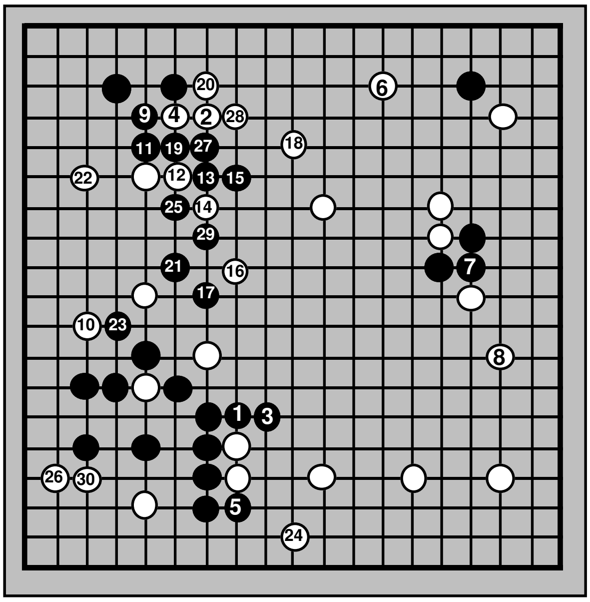


figure INDIGO-Poka-31-60

Le coup 7 est joué pour connecter les deux pierres noires.

Enfin !

Ouf !

Les coups 9 et 11 sont joués pour stabiliser (resp. déstabiliser) le groupe noir en haut à gauche (resp. blanc 2-4) avec le jeu de l'opposition.

Le coup 13 est joué pour déconnecter le groupe blanc 2-4 du groupe blanc 12.

Ah bon ? L'objectif n'est pas atteint.

C'est la bogue citée dans la partie précedente, inversion du coup noir et du coup blanc, mais sur une autre règle. Le problème est que nos règles topologiques ont été construites à la main sans outil de vérification automatique. Ce type de bogue pousserait à reconstruire automatiquement la base de règles automatiquement.

Ensuite, INDIGO déconnecte tout ce qu'il peut déconnecter sur le bord gauche. Il y arrive au prix de nombreux coups.

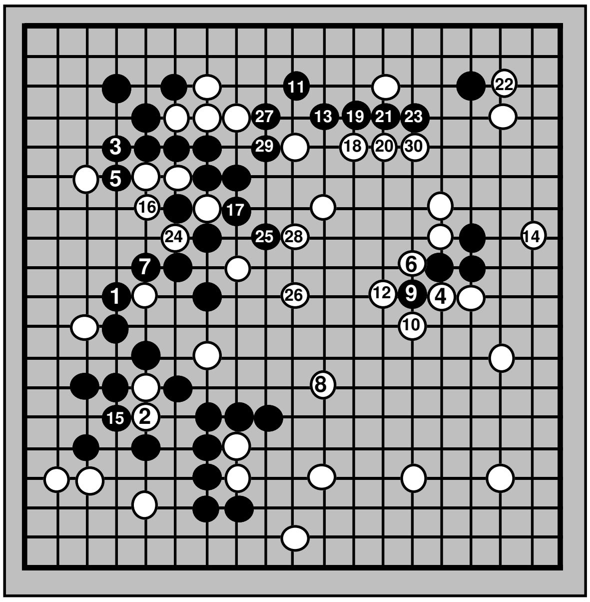


figure INDIGO-Poka-61-90

Pendant ce temps, Poka encercle les 3 pierres noires du bord droit.

Le coup 11 est joué pour déstabiliser les groupes blancs du bord haut avec le jeu de la dilatation.

INDIGO laisse un ponnuki à Poka !!!

Pour qu'INDIGO empêche le ponnuki en utilisant des heuristiques statiques, il faudrait améliorer le niveau global en différenciant les gains certains des gains potentiels. Le coup 11 d'INDIGO est une déstabilisation potentielle du bord haut (20 intersections instables selon INDIGO). Le coup 12 de Poka est une stabilisation certaine de quatre petits groupes : 4, 6 et 10 mangent le groupe 9 (quatre intersections instables selon INDIGO).

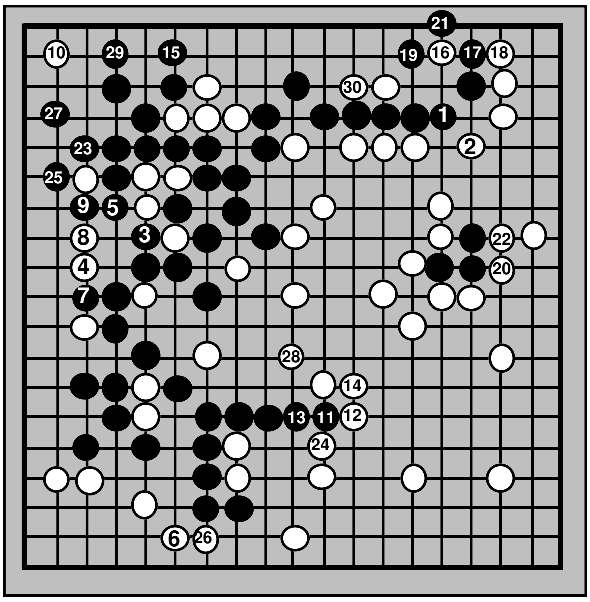


figure INDIGO-Poka-91-120

Pour s'approprier le bord gauche, INDIGO joue neuf coups contre trois à Poka. Par différence, on peut estimer qu'INDIGO dépense six coups pour prendre le bord gauche. Ceci est un des problèmes des programmes de Go actuels : évaluer la rentabilité des coups a priori.

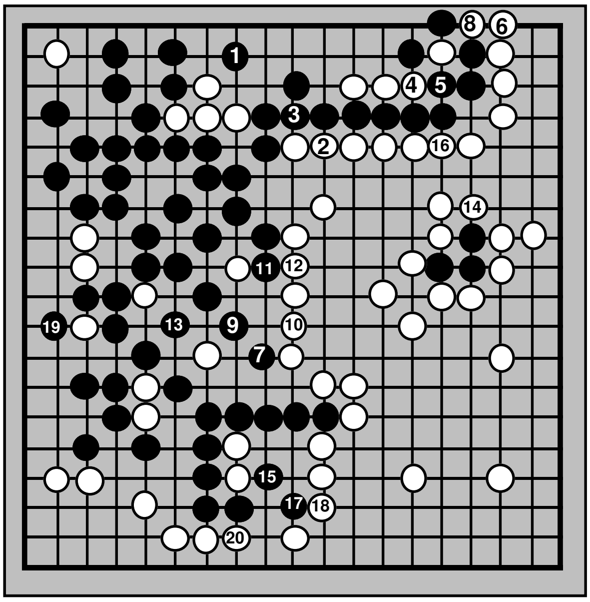


figure INDIGO-Poka-121-140

INDIGO dépense encore deux coups pour capturer définitivement. Pendant ce temps Poka aggrandit et ferme ses territoires.

La partie jouée manuellement sur IGS a été arrêtée quelques coups plus tard. Le goban est clairement partagé en diagonale en deux part persque égales. Poka est en avance d'une vingtaine de points car il a trois coins contre un à INDIGO. D'abord, Poka construit des territoires, ce que ne fait pas INDIGO. Ensuite, Poka donne des pierres à INDIGO qui dépense beaucoup trop de coups pour les capturer. INDIGO est plus combatif que Poka. Howard Landman, l'auteur de Poka, et nous-même avons été très interéssés par cette partie riche et équilibrée opposant le style combatif d'INDIGO au style territorial de Poka.

INDIGO - Gogol (I)

Partie jouée le 6 Juillet 1994.
Noir : INDIGO
Blanc : Gogol

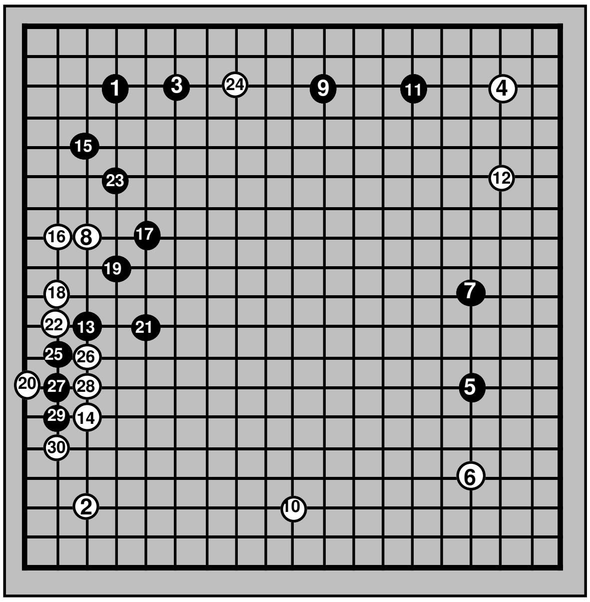


figure INDIGO-Gogol-(I)-1-30

Le coup 5 est joué pour remplir un espace vide.

La formation 1-3-5 est typique d'INDIGO dans les fusekis croisés. 5 est curieux.

Les coups 7, 11, 13 et 15 sont joués pour stabiliser ou déstabiliser des groupes.

Les extensions[1] ou contre-extension (dans le jargon du Go) 7, 11, 12, 13, 14, 15 sont typiques des fusekis INDIGO-Gogol.

Ce sont de bons coups.

Les coups 19, 21, 23 sont joués pour connecter les groupes noirs 13, 15 et 17.

Le coup 25 est joué pour déstabiliser le groupe blanc 8-16-18-22.

INDIGO joue avec un seuil de stabilité du jeu de la chaîne égal à 3 libertés et ne voit pas la capture de la chaîne avant le coup 29.

[1] Dilatations dans le jargon d'INDIGO.

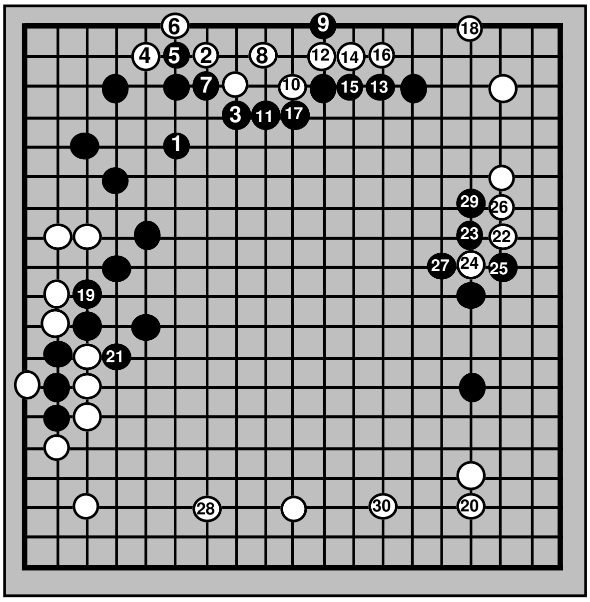


figure INDIGO-Gogol-(I)-31-60

La séquence de 1 à 18 est simplement ahurissante.

À chaque coup, INDIGO et Gogol ont une bonne raison de jouer leur coup. Personne ne profite des erreurs de l'autre. Finalement, le résultat est équilibré : vie sur le bord pour Gogol et influence pour INDIGO.

Le moyo central de noir est impressionnant.

De règles locales simples, émerge un effet global imprévisible... Les choses vont parfois dans le bon sens.

A voir leur synchronisation pour passer d'une séquence à l'autre, Gogol et INDIGO semblent avoir des objectifs similaires.

INDIGO et Gogol ont beucoup joué ensemble et finissent par avoir des points communs. Entre autre, la combativité et la priorité mise sur la (dé-)connexion des groupes.

Par contre, ils diffèrent sur certains points, comme le montre les coups 19-21-27 pour INDIGO, contre 20-28-30 pour Gogol. Ici, INDIGO privilégie la solidité et Gogol le territoire.

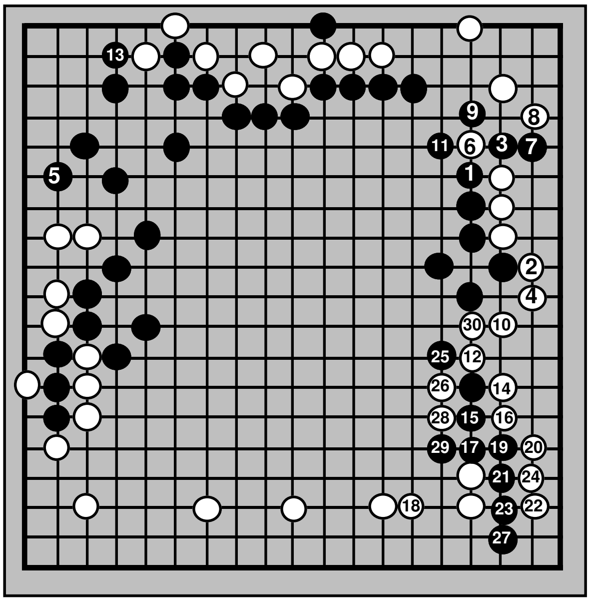


figure INDIGO-Gogol-(I)-61-90

Suite des hostilités.

On peut taper chef ?

Vas-y, tape mais fait attention, lui aussi, Gogol, il tape.

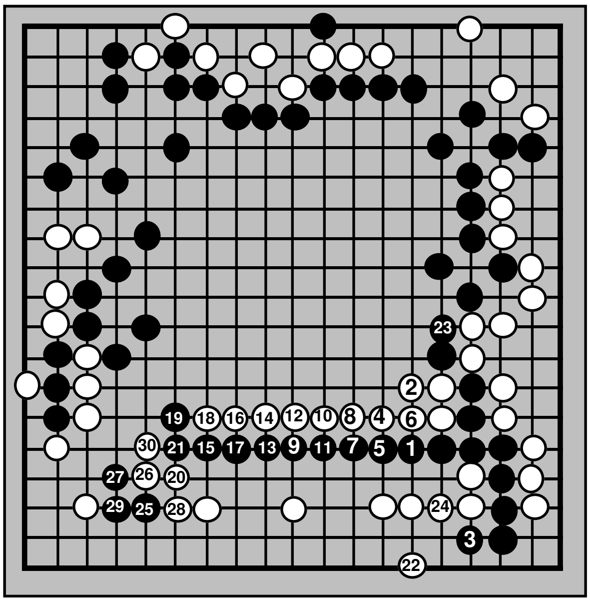


figure INDIGO-Gogol-(I)-91-120

Le p'tit train.

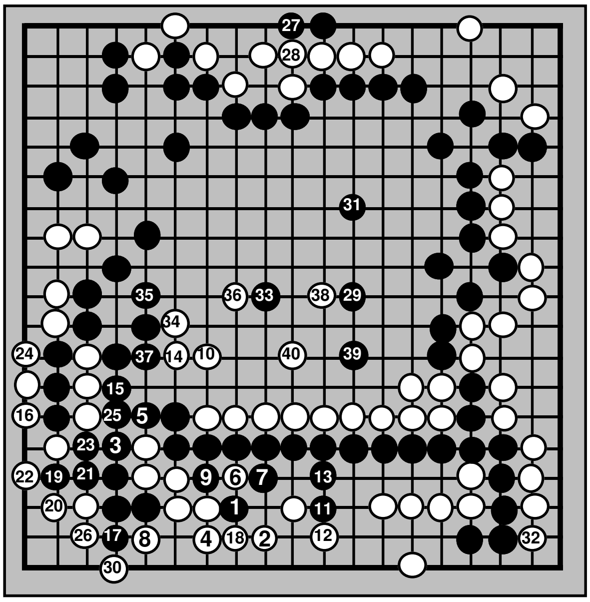


figure INDIGO-Gogol-(I)-121-160

Le statut du groupe blanc central à l'air de ne pas préoccuper les protagonistes.

Gogol et moi, on a des comptes à règler sur le bord, le groupe central, on verra après.

`Abandon de poste caractérisé.`

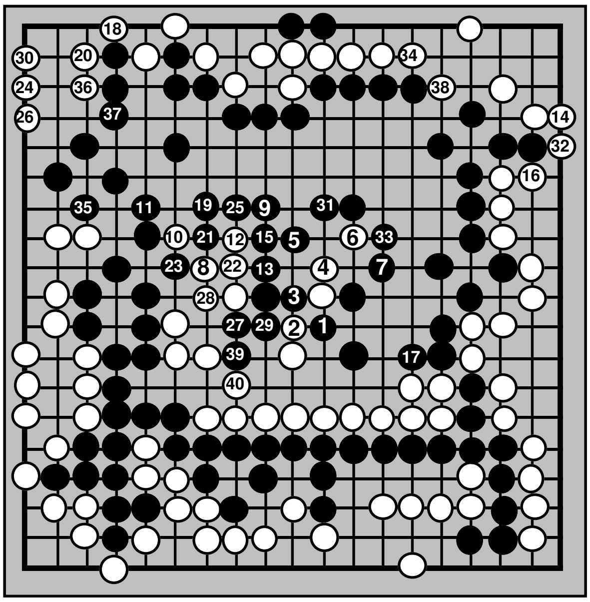


figure INDIGO-Gogol-(I)-161-200

Ca y est, ils s'intéressent enfin au groupe central.

`Finalement INDIGO tue le groupe blanc et gagne la partie de 40 points.`

INDIGO - Gogol (II)

Partie jouée le 13 Juillet 1994.
Noir : INDIGO
Blanc : Gogol

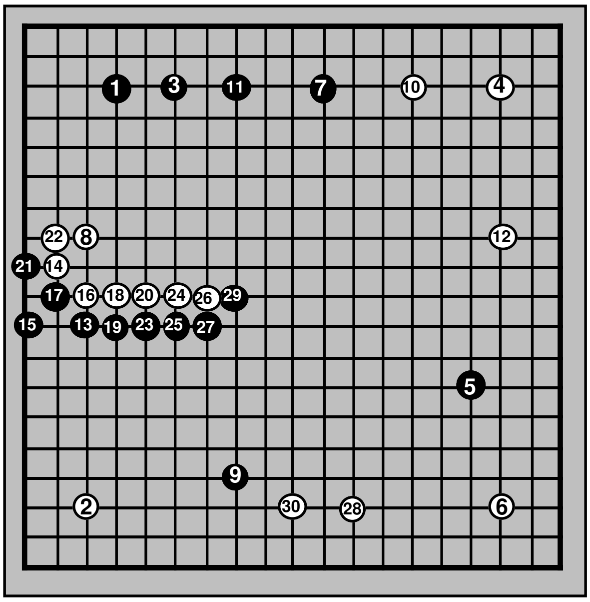


figure INDIGO-Gogol-(II)-1-30

Le coup 5 est joué pour remplir un espace vide.

Le coup 11 est joué pour stabiliser les groupes 1-3 et 7.

Le coup 13 est joué pour déstabiliser le groupe 8.

Le coup 15 est joué pour encercler le groupe 8-14.

Les coups 17, 19, 21, 23, 25, 27 et 29 sont joués pour déstabiliser le groupe blanc avec le jeu de l'opposition.

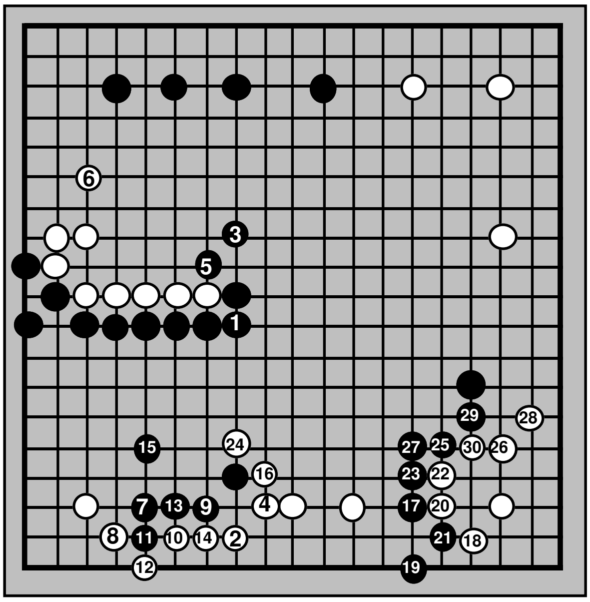


figure INDIGO-Gogol-(II)-31-60

Le coup 1 est joué pour connecter les groupes noirs.

INDIGO semble vouloir construire une puissance centrale

Non, INDIGO construit la puissance centrale sans le vouloir.

Gogol et INDIGO laissent tomber 24 !

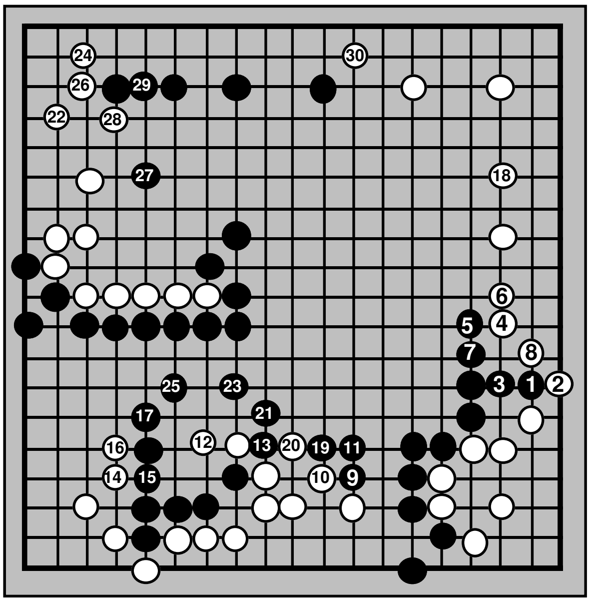


figure INDIGO-Gogol-(II)-61-90

Gogol connecte toujours ses groupes pres du bord avant qu'INDIGO ne le fasse car il manque des règles de (dé-)connexion près du bord à INDIGO, 8 est un exemple...

C'est curieux que 13 soit joué si tardivement !

22, 24, 26 de Gogol sont très gros. INDIGO ne fait pas grossir ses territoires. C'est une perspective pour améliorer INDIGO.

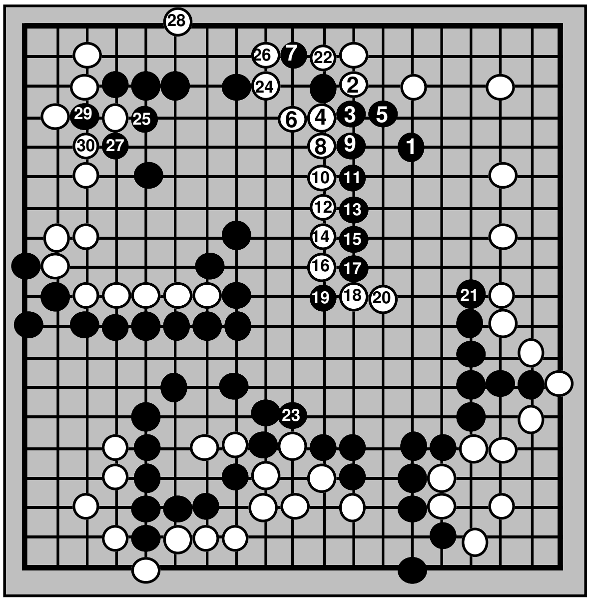


figure INDIGO-Gogol-91-(II)-120

7 est une grosse erreur, INDIGO pense que la chaîne est sauvée car le seuil de stabilité est de 3 libertés dans cette version d'INDIGO.

Après 21, 22 est une grosse erreur (atari dans le mauvais sens), 23 est gros mais pourquoi n'a-t-il pas répondu à l'atari ?

A cause de la valeur des coups au niveau global, sans doute.

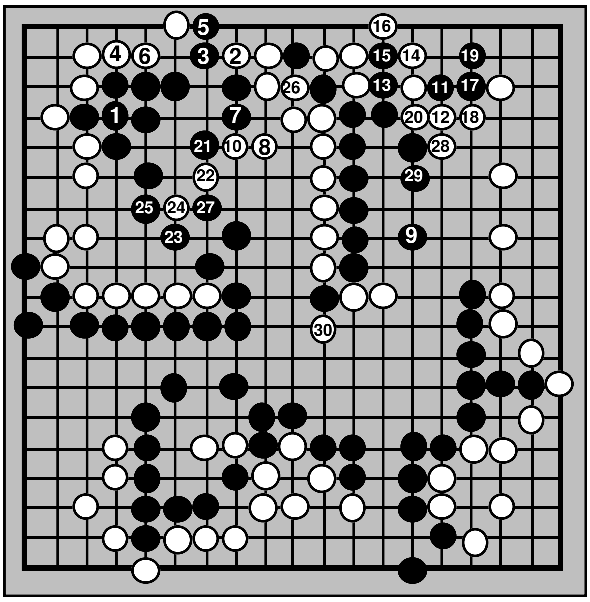


figure INDIGO-Gogol-(II)-121-150

Jouer le premier vers 9 est très important. Gogol et INDIGO n'ont pas du tout l'air de le considérer

INDIGO va-t-il répondre à 30 ?

Dans ce type de situation, on n'est jamais sûr de rien...

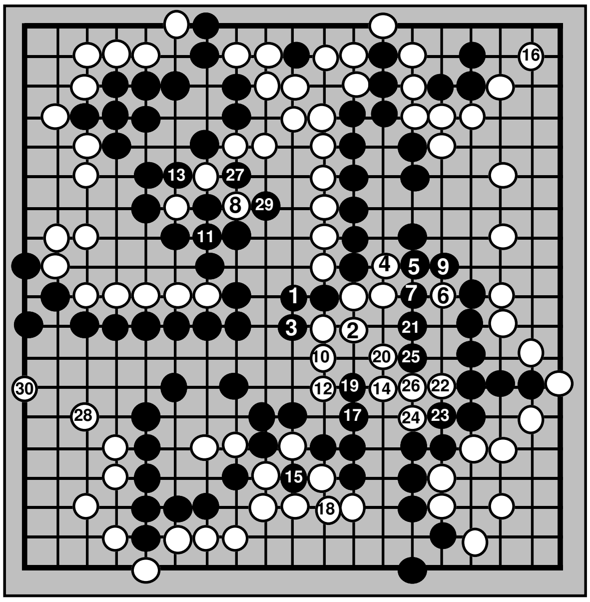


figure INDIGO-Gogol-(II)-151-180

1, ouf...

INDIGO tue à peu près correctement le groupe blanc qui tente de vivre.

Bravo INDIGO !

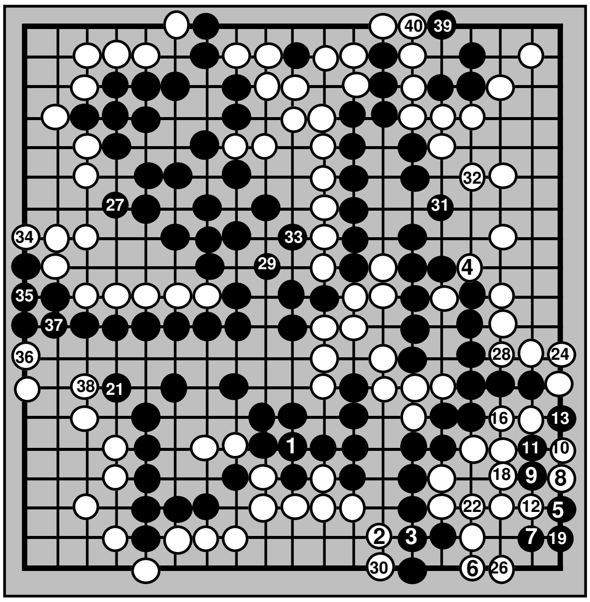


figure INDIGO-Gogol-(II)-181-220

A la fin de la partie, Gogol gagne de 29 points. Bravo Gogol !

INDIGO - Hugh

Partie jouée le 26 Mars 1994.
Noir : INDIGO
Blanc : Hugh

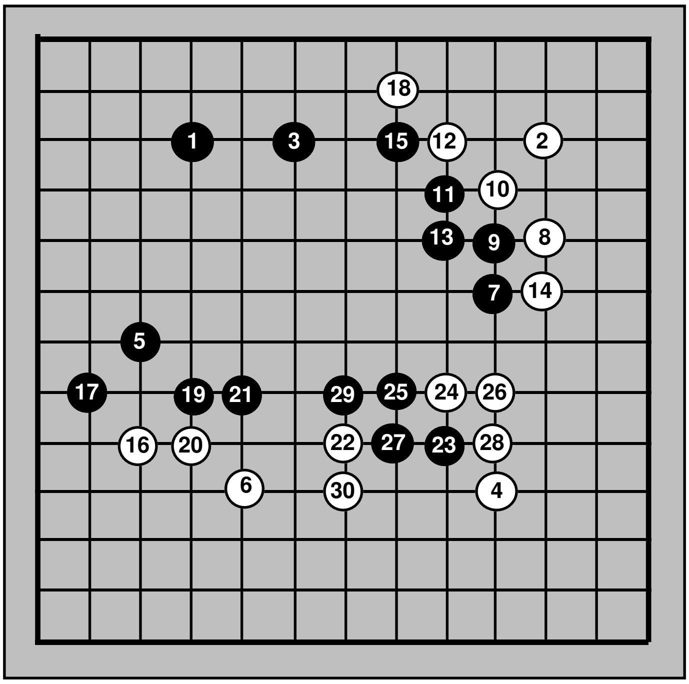


figure INDIGO-Hugh-1-30

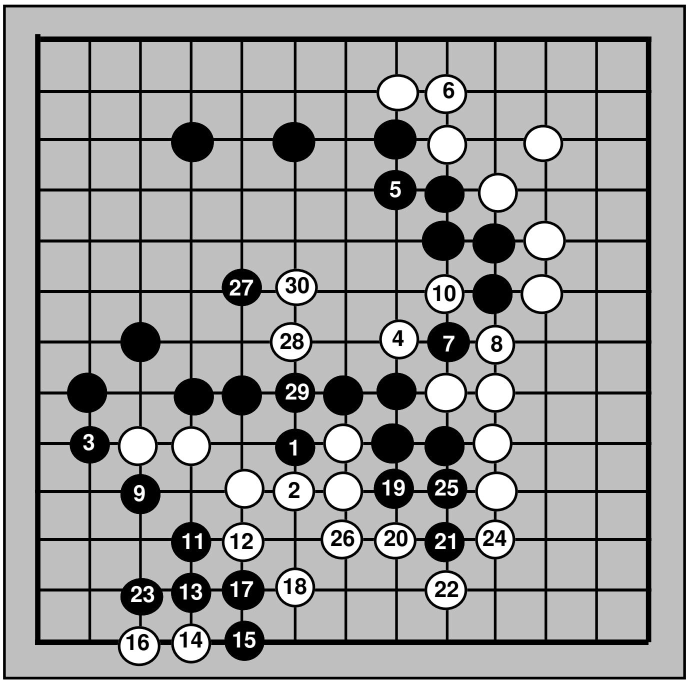


figure INDIGO-Hugh-31-60

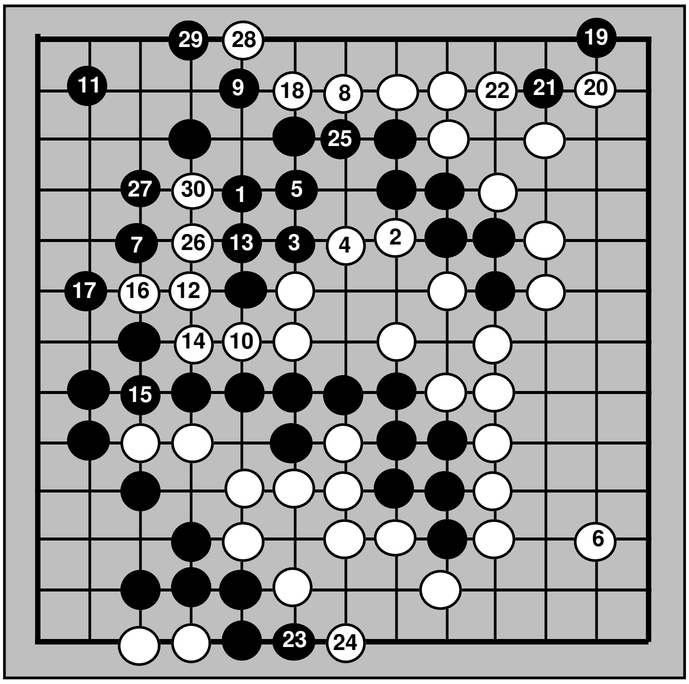


figure INDIGO-Hugh-61-90

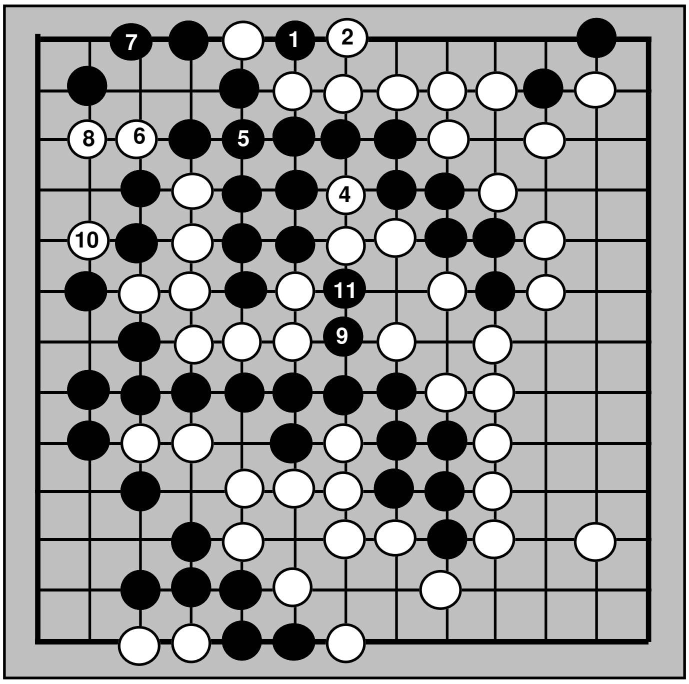
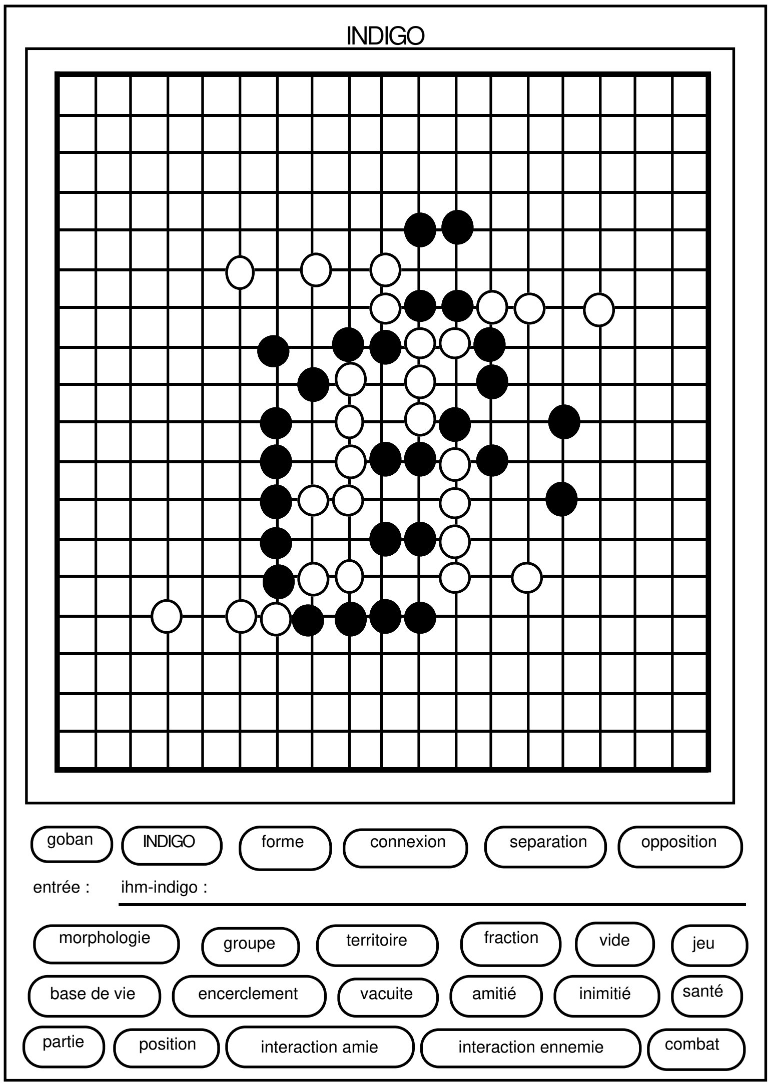


figure INDIGO-Hugh-91-101

Hugh est un être humain (!). Il a appris à jouer en jouant contre INDIGO. Cette partie date du moment où les parties sont devenues équilibrées entre Hugh et INDIGO. Hugh fait une erreur avec 8 et abandonne (enragé!).

ANNEXE C : L'INTERFACE HOMME MACHINE

L'aspect de l'interface homme-machine d'INDIGO est le suivant :



Cette interface homme machine nous a permis de développer et mettre au point INDIGO. Elle possède un goban surlequel il est possible de jouer ou demander des informations locales. Elle possède aussi un ensemble de boutons permettant d'effectuer des opérations globales sur le goban. Les résultats sont affichés dans la fenêtre Unix d'où a été lancé le programme. Une perspective intéressante pour améliorer cette interface, serait d'afficher les résultats visualisables sous forme visuelle sur le goban. Nous avons développé cette interface avec xview.

Le goban

En cliquant sur une intersection du goban, l'utilisateur peut jouer un coup, poser ou enlever une pierre, ou bien cliquer sur des menus déroulants, qui permettent de demander quels objets d'une classe donnée existent sur l'intersection. Si des objets n'ont pas été reconnus ou calculés sur l'intersection cliquée, l'utilisateur peut demander la reconnaissance ou les calculs. L'information associée à cette partie de l'interface reste locale.

Les boutons inférieurs

En utilisant les boutons inférieurs, l'utilisateur peut demander des informations globales sur l'état du système ou sur une classe d'objets particulières. Les informations globales d'une classe sont entre autres, la liste et le nombre d'objets d'une classe. L'utilisateur peut aussi demander des opérations globales sur une classe. En particulier, l'utilisateur peut lancer une interprétation du goban; charger ou sauver une position; charger, sauver ou visualiser pas à pas une partie; appliquer des filtres morphologiques sur une position, lancer des recherches arborescentes sur une classe de jeu.

aji

Terme japonais qui signifie *arrière-goût*. Au Go, se dit d'une situation théoriquement acquise à une couleur pour laquelle l'autre couleur garde beaucoup de possibilités.

alpha-bêta

Technique de recherche arborescente. cf. Théorie de jeux, partie 2.

atari

Coup qui ne laisse qu'une seule liberté à une chaîne.

attribut

Terme spécifique de la programmation orientée objet qui désigne une propriété d'un objet.

bogue

Terme informatique qui signifie erreur. Vient du mot anglais *bug.*

chaîne

Ensemble maximal de pierres voisines au sens de la règle du jeu (cf. Règle du jeu en Annexe A)

chaîne-poison

Type de règle. cf. Jeu de la séparation du territoire en 2, niveau élémentaire, partie 3.

chunk

Morceau de mémoire (cf. partie 1)

classe

Terme spécifique de la programmation orientée objet qui représente un ensemble d'objets ayant les mêmes propriétés.

cogniticien

Individu qui recueille une expertise dans un domaine auprès d'un expert du domaine.

computationnel(le)

Se dit d'une théorie adaptée à une machine.

connaissance

1 - Pour un être humain, activité intellectuelle visant à avoir la compétence de quelquechose.
2 - En Intelligence Artificielle, par antropomorphisme, se dit d'une règle dans un programme informatique sur une machine.

conscientisable

Se dit d'une connaissance non consciente que l'on peut rendre consciente par un travail de conscientisation.

conscientisation

Le travail nécessaire pour transformer une connaissance conscientisable en connaissance consciente.

C++

Langage de développement de notre programme. Surlangage de C avec des "fonctionnalités" objet.

dame

Intersection neutre en fin de partie au Go.

damezumari

Terme de Go, (mauvais) coup qui supprime une liberté amie d'une chaîne et la rend vulnérable.

dan

Niveau au Go. cf partie 1. Un joueur confirmé est 1er dan. Un joueur fort est 5ème dan.

déclaratif

Par opposition à procédural. Propriété d'une connaissance dont la déclaration est séparée de son mode d'emploi. Une connaissance déclarative est souvent facilement modifiable.

est-un

Terme spécifique de la programmation orientée objet qui désigne la relation d'héritage. Exemple : une chien est un animal.

fraction

Concept de notre modèle décrit dans la partie 3

fuseki

Début de partie au Go

goban

Damier sur lequel on joue au Go. Sa taille usuelle est 9-9, 13-13 ou 19-19.

géta

Terme japonais qui signifie *sabot*. Au go, il signifie coup d'encerclement.

hane

Terme japonais. cf. partie 2, verbalisations, une histoire comme ça.

IA

Intelligence Artificielle

IAD

Intelligence Artificielle Distribuée

IGS

International Go Server. Serveur de Go où des joueurs du monde entier jouent au Go.

ikken-tobi

Terme de Go qui signifie saut d'une intersection.

ima

Terme japonais qui signifie *maintenant*.

implémentation

Activité qui consiste à traduire un modèle sous forme de programme pour que celui-ci tourne sur une machine.

incrémental

En programmation du jeu de Go, se dit d'un algorithme qui ne recalcule que les changements induits par la pose ou le retrait d'une pierre.

INDIGO

Le nom de notre programme. Signifie Is Now Designed In Good Objects.

instance

Spécifique de la programmation orientée objet, anglais francisé qui signifie exemple.

intérieur-extérieur

Dualité entre l'intérieur et l'extérieur. L'être humain reconnaît facilement l'intérieur de l'extérieur des objets.

interpréteur

Mécanisme qui évalue les symboles d'un langage.

intersection

Terme de Go. Lieu où les pierres de Go sont posées. Il y a 361 intersections sur un goban 19-19.

intuitif

Se dit d’un concept (ou d’un processus associé à un concept) que l’on ne peut pas conscientiser ou expliciter. On peut seulement supposer son existence et lui faire correspondre un équivalent.

joseki

Terme de Go. Séquence de coups équilibrée pour les deux joueurs jouée en début de partie.

kiri

Terme de Go. Représente une séparation mutuelle. cf partie 2, verbalisations, histoire comme ça.

ko

Terme japonais qui signifie *infini*. Concept de la règle du jeu qui interdit une répétition d'ordre 2. (cf. Annexe A règle du jeu).

koko

Terme japonais qui signifie *ici*.

kosumi

Terme de Go. Représente une connexion implicite. cf partie 2, verbalisations, histoire comme ça.

kyu

Niveau au Go. cf partie 1. Un joueur débutant est 30ème kyu. Un joueur faible est 20ème kyu. Un joueur moyen est 10ème kyu. Un joueur presque confirmé est 1er kyu.

LAFORIA

LAboratoire de reconnaissance de FORmes et d'Intelligence Artificielle où nous avons effectué le travail présenté dans ce document.

liberté

Terme de Go. Se dit d'une intersection vide voisine d'une chaîne.

MCT

Mémoire à Court Terme.

MLT

Mémoire à Long Terme.

méta

Préfixe qui s'attribue à quelquechose. Un métaquelquechose est un quelquechose qui s'applique à un quelquechose. Exemple: une métarègle est une règle qui agit sur une règle.

miai

Terme de Go. Deux objets sont miai s'ils sont équivalents. Si un joueur, prend un objet miai, l'autre joueur prend l'autre.

minimax

Technique de recherche arborescente utilisée en théorie des jeux. cf. Théorie de jeux, Partie 2.

MOOD INDIGO

Le nom composé de notre programme. Signifie My Object Oriented Design Is Now Designed In Good Objects.

moyo

Terme de Go. Ebauche de territoire ou territoire à grande échelle.

n-connexe

Se dit d'un réseau de points où chaque nœud est voisin de n nœuds.

nobi

Terme de Go qui désigne 2 pierres voisines de la même couleur.

objet

Terme en informatique qui regroupe et structure des données pour les utiliser.

objet-posé

Terme de notre modèle qui représente un objet qui repose sur le goban. Par exemple, un groupe, une chaîne, un territoire. Contre-exemples : une partie, une liste.

œil

Liberté particulière d'un groupe qui est telle que si un groupe a deux vrais yeux, il est vivant indépendamment du voisinage.

pattern

Dans ce document, le terme *pattern* signifie un petit dessin qui contient la disposition des pierres sur une partie du goban.

pattern-matching

Terme d'informatique. Processus d'appariement entre une base de formes (ou pattern) et une position.

pierre

Un pion au Go s'appelle une pierre.

point

Intersection contrôlée par un joueur.

ponnuki

Terme japonais. Forme engendrée par la prise d'une chaîne constituée d'une seule pierre.

procédural

Par opposition à déclaratif. Une connaissance est procédurale si sa déclaration et son mode d'emploi sont dépendants.

quiescence

Technique de recherche arborescente qui engendre seulement les coups forcés jusqu'à trouver des positions stables sans coup forcé (cf. Théorie de jeux, Partie 2).

seki

Position de deux groupes voisins qui ne peuvent s'attaquer mutuellement. Selon la théorie de Conway, correspond au jeu 0 : le premier qui joue perd (cf. Théorie de jeux, Partie 2).

sente

Terme de Go qui signifie initiative.

shicho

Terme japonais qui signifie *escalier*. Séquence de coups au Go qui part en diagonale avec une alternance d'ataris et de sorties à deux libertés. cf. Partie 3, niveau zéro ou Partie 2, domaines voisins, IAD.

Sun

Machine sur lesquelles nous avons implémenté INDIGO.

this

Terme de C++ qui désigne l'objet courant.

tsumego

Terme de Go qui désigne un problème de vie et de mort.

Unix

Système d'exploitation sur lequel nous avons développé notre programme.

yose

Terme de Go qui désigne la fin de partie au Go. Le petit yose désigne la toute fin de partie quand les joueurs jouent des coups qui ne valent plus que 1 ou 2 points.

zone-poison

Type de règle de notre modèle utilisé dans le jeu de la séparation du territoire en 2. cf. niveau élémentaire, partie 3.

zéro-programme

Programme minimal qui ne connaît que les règles du jeu. cf. Annexe A, Règle du jeu.

=+

ANNEXE E

=+

INTRODUCTION

Cette annexe est constituee de QUELQUES fichiers C++ de INDIGO.

Quelques points de vue, relativement auto-suffisants, sont illustres par un exemple representatif.

Les points de vue choisis sont:

1 - Representation des objets generaux
2 - Jeu illustre par le jeu de la chaine
3 - Interpretation du goban et decision globale
4 - La classe Groupe et un attribut, la classe Amitie
5 - "Optimisation" de la morphologie mathematique

=+

1 - Representations des objets generaux:

goban_taille_max.h
entier.h
objet.h
go_objet.h
objet_pose.h
ensemble_intersection.h

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
//  goban_taille_max.h

#ifndef GOBAN_T_MAX_H
#define GOBAN_T_MAX_H

#define T_MAX      21
#define SURFACE_MAX     400
#define N_LOCAL_MAX     20

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// entier.h

#ifndef ENTIER_H
#define ENTIER_H

#define Entier_16   int
#define Entier_32   int
#define Entier_64   int

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// objet.h

#ifndef OBJET_H
#define OBJET_H

#include "objet_macro.h"
#include <stdio.h>

class Objet: public Ressource
{
public:
	Objet();
	~Objet();

	static int nombre;
	static int nombre_precedent;

	static void imprimer_nombre (int);

	virtual void detruire();
	virtual void afficher();
	virtual void fimprimer(FILE *, int, int);
	virtual void fafficher(FILE *, int);
	virtual void marquer_ihm(char);
};

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
//  go_objet.h

#ifndef GO_OBJET_H
#define GO_OBJET_H

#include "objet.h"

class Liste;
class Go_objet: public Objet
{
public:
	int numero;
	Go_objet();
	~Go_objet();

	static void detruire_les_instances(Liste *);
	virtual void detruire();
	virtual void fimprimer(FILE *, int, int);
};

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
//  objet_pose.h

#ifndef OBJET_POSE_H
#define OBJET_POSE_H

#include "objet.h"
#include "go_objet.h"
#include "goban_taille_max.h"

#include "objet_pose_macro.h"

#include <stdio.h>

class Liste;
class Intersection;
class Ensemble_intersection;
class Objet_pose: public Go_objet
{
public:
	static int nombre_op;

	Ensemble_intersection * intersections;
	Ensemble_intersection * intersections_limite;
	Ensemble_intersection * empreinte;
	int couleur;
	int naissance;
	int mort;

	Objet_pose(Ensemble_intersection *, int);
	~Objet_pose();

	static Objet_pose * de_liste_a_adresse(
		Liste *, Ensemble_intersection *);

	static void focaliser(Liste *, int, Objet_pose** );

	void fafficher(FILE *, int);

	static void fafficher_intersections(
		FILE *, int, Ensemble_intersection *);
	static void ordonner_par_taille(Liste *, Liste **);
	static int trouver_taille_max(Liste *);
	static void trouver_objets_de_taille(
			Liste *, int, Liste **);
	static void trouver_objets_autre_couleur(
			Liste *, int, Liste **);
	virtual void detruire();
	virtual void fimprimer(FILE *, int, int);

	int valeur();
	int vide();
	int monocolore();
	int bicolore();
};

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// ensemble_intersection.h

#ifndef ENSEMBLE_INTERSECTION_H
#define ENSEMBLE_INTERSECTION_H

#include "ensemble_intersection_macro.h"
#include "objet.h"
#include "go_objet.h"
#include "objet_echelonne.h"
#include "goban_taille_max.h"

#include <stdio.h>

class Intersection;
class Liste;
class Ensemble_intersection : public Objet_echelonne
{
public:
	static int nombre_ei;
	static int debug;
	static Ensemble_intersection * A;
	static Ensemble_intersection * B;
	static Ensemble_intersection * C;

	Ensemble_intersection();
	~Ensemble_intersection();

	void detruire();
	void fimprimer(FILE *, int, int);

	static void ou_logique(
```

```
                        Ensemble_intersection **, Intersection *);
        static void et_logique(
                        Ensemble_intersection **, Intersection *);
        static void enlever(Ensemble_intersection **, Intersection *);
        static void ajouter(Ensemble_intersection **, Intersection *);
        static void enlever_tout(Ensemble_intersection **);

        void ou_logique_liste(Ensemble_intersection **);
        void ajouter_liste(Ensemble_intersection **);
        void et_logique_liste(Ensemble_intersection **);
        void enlever_liste(Ensemble_intersection **);

        void ensemble_dilate(Ensemble_intersection **);
        void ensemble_dilate_classique(Ensemble_intersection **);
        void ensemble_dilate_optimise(Ensemble_intersection **);

        void ensemble_dilate_nd(Ensemble_intersection **, int);
        void ensemble_dilate_nd_classique(
                        Ensemble_intersection **, int);
        void ensemble_dilate_nd_optimise(
                        Ensemble_intersection **, int);
        void ensemble_dilate_obstacle_nd(
                        Ensemble_intersection **, int);
        void ensemble_dilate_obstacle_nd_classique(
                        Ensemble_intersection **, int);
        void ensemble_dilate_obstacle_nd_optimise(
                        Ensemble_intersection **, int);

        void ensemble_dilate_obstacle(Ensemble_intersection **);
        void ensemble_dilate_obstacle_classique(
                        Ensemble_intersection **);
        void ensemble_dilate_obstacle_optimise(
                        Ensemble_intersection ;

        void ensemble_erode(Ensemble_intersection **);
        void ensemble_erode_classique(Ensemble_intersection **);
        void ensemble_erode_optimise(Ensemble_intersection **);

        void ensemble_erode_ne(Ensemble_intersection **, int);
        void ensemble_erode_ne_classique(
                        Ensemble_intersection **, int);
        void ensemble_erode_ne_optimise(
                        Ensemble_intersection **, int);

        void frontiere_externe(Ensemble_intersection **);
        void frontiere_interne(Ensemble_intersection **);

        void ensemble_dilate_diagonal(Ensemble_intersection **);

        int rechercher(Intersection *);


};

#endif
```

=+

2 - Jeu illustre par le jeu de la chaine

jeu_define.h
jeu.h
jeu.c
jeu_simple.h
jeu_simple.c
jeu_chaine.h
jeu_chaine.c

=+

```
// jeu_define.h

#ifndef JEU_DEFINE_H
#define JEU_DEFINE_H

#define GAME_EVERY        64

#define GAME_STAR    16
#define GAME_POSITIVE_OR_ZERO        9
#define GAME_MORE   8
#define GAME_KO         3
#define GAME_POSITIVE_BUT_DOUBLE_IS_NEGATIVE 2
#define GAME_POSITIVE     1
#define GAME_FUZZY        0
#define GAME_NEGATIVE   -1
#define GAME_LESS     -8
#define GAME_ZERO_OR_NEGATIVE     -9
#define GAME_ZERO    -16

#define GAME_UNCALCULATED      -32
#define GAME_ININTERESTING        32

#define JEU_DEBUT     0
#define JEU_FIN           1

#define JEU_ZERO        16
#define JEU_GAGNE     8
#define JEU_SATURE   -2
#define JEU_DEFAUT   -1
#define JEU_INSTABLE       0
#define JEU_PERDU_OU_INCONNU        1
#define JEU_INSTABLE_MAIS_DOUBLE_EST_PERDU 2
#define JEU_INFINI       3
#define JEU_PERDU      -8
#define JEU_INCALCULE    -16


#endif
```

=+

```
// jeu.h

#ifndef JEU_H
#define JEU_H

#include  "objet.h"
#include  "go_objet.h"
#include  "objet_pose.h"
#include  <stdio.h>


class Liste;
class Ensemble_intersection;
class Intersection;
class Coup;
class Position_calculee;
class Jeu : public Objet_pose
{
public:
        static Jeu * courant;
        static int calcul_en_cours;
        static int nombre_j;
        static int nombre_saturations;

        int resultat;
        int etat_couleur;
        int etat_autre_couleur;

        Intersection * attaque;
        Intersection * defense;

        Liste * coups;

        int couleur_premier_coup;

        int taille_max;
        int taille_etudiee;
        int profondeur_etudiee;

        Liste * positions_calculees;

        Jeu(Ensemble_intersection *, int);
        ~Jeu();

        static void imprimer_tout(FILE *, int, int);
        virtual void relier_empreinte(int);
        virtual void detruire();
        virtual void formes_activees(int);
        virtual void initialiser_calculs();
        int determiner_etat2(int);
        void analyse_dynamique(int, int *, Intersection **, int );
        virtual void analyse_statique(int, int *, Liste **);
        virtual Coup * jouer_coup2(Intersection *, int);
        virtual void reprendre_coup2(Coup *);

        void sauver_position_calculee(
              int, int, Intersection *, int, int);
        Position_calculee * rechercher_position_calculee(
              int, int*, Intersection **);

        virtual void valoriser_mauvais();
        virtual void identifier();

};

#endif
```

=+

```
// jeu.c

#include  "jeu.h"
#include  "jeu_define.h"
#include  "jeu_ihm.h"

#include  "liste.h"
#include  "coup.h"
#include  "forme_reconnue.h"
#include  "goban.h"
#include  "intersection.h"
#include  "pierre.h"
#include  "chaine.h"
#include  "regle_du_jeu.h"
#include  "utile.h"
#include  "couleur.h"
#include  "goban_affiche.h"
#include  "lib_jeu.h"
#include  "lib_env.h"
#include  "partie.h"
#include  "version.h"


#include  <stdio.h>
```

```
#include <stream.h>
#include <stddef.h>

#define DEBUT_DE_JEU      -2
#define FIN_DE_JEU     -3

int Jeu::nombre_j = 0;
int Jeu::nombre_saturations = 0;
Jeu * Jeu::courant;
int Jeu::calcul_en_cours;

//////////////////////////////////////////////////////////////////////
Jeu::Jeu(Ensemble_intersection * l, int c)
	: Objet_pose(l, c)
{
	Coup * cp;
	Partie * pa;

	nombre_j++;
	courant = this;
	taille_etudiee = 0;
	profondeur_etudiee = 0;
	attaque = NULL;
	defense = NULL;
	resultat = 0;
	etat_couleur = 0;
	etat_autre_couleur = 0;
	coups = new Liste();
	pa = Partie::courante;
	if (ptrok(pa)) cp = pa->coup_precedent();
	else cp = NULL;
	if (ptrok(cp)) Liste::ou_logique(&coups, cp);

}

//////////////////////////////////////////////////////////////////////
Jeu::~Jeu()
{
	if (ptrok(coups))  coups->detruire();
	nombre_j--;
}

definir_destruction(Jeu)

//////////////////////////////////////////////////////////////////////
void Jeu::analyse_dynamique(
int trait, int * res, Intersection ** i, int beta)
{
	int etat_dyn;
	int etat_stat;
	Liste * li;	// liste des intersections envisagees
	Intersection * j;
	Intersection * k;
	int largeur_etudiee;
	int passe;
	int alpha;
	Coup * cp;

	*i = NULL;
	passe = FAUX;
	if (couleur==trait) {
		alpha = JEU_PERDU;
		*res = JEU_PERDU;
	}
	else {
		alpha = JEU_GAGNE;
		*res = JEU_GAGNE;
	}

	li = new Liste();
	analyse_statique(trait, &etat_stat, &li);

	if (debug_ad==VRAI) {
		printf("%s: ad, etat_stat = <%d>\n", l_identite, etat_stat);
	}

	switch(etat_stat) {
	case JEU_GAGNE:
	case JEU_PERDU:
	case JEU_INFINI:
		*res = etat_stat;
		if (li->longueur==1) *i = (Intersection *) li->premier();
		li->detruire();
		return;
	default:
		if (li->longueur==0) passe = VRAI;
		break;
	}

	largeur_etudiee = 0;
	for(;;) {
		j = (Intersection *) li->premier();

		if ( (j==NULL) && (passe == FAUX) ) {
			li->detruire();
			return;
		}
		else { passe = FAUX; }
```

```
		if (taille_etudiee==taille_max) {
			nombre_saturations++;
			li->detruire();
			*res = JEU_SATURE;
			return;
		}

		cp = jouer_coup2(j, trait);
		largeur_etudiee++;
		taille_etudiee++;
		analyse_dynamique(
			autre_couleur(trait), &etat_dyn, &k, alpha);
		reprendre_coup2(cp);

		Liste::enlever(&li, j);

		if (etat_dyn==JEU_SATURE) {
			*res = JEU_SATURE;
			li->detruire();
			return;
		}

		if (*res==JEU_INFINI) {
			li->detruire();
			*i = j;
			return;
		}

		if (couleur==trait) {
			if ( (*res<etat_dyn) || (*res==etat_dyn) ) {
				alpha = etat_dyn;
				*res = etat_dyn;
				*i = j;
			}
			if (*res==JEU_GAGNE) {
				li->detruire();
				return;
			}
			if (couleur_premier_coup == couleur) {
				if (alpha > beta) {
					li->detruire();
					return;
				}
			}
		}
		else {
			if ( (*res>etat_dyn) || (*res==etat_dyn) ) {
				alpha = etat_dyn;
				*res = etat_dyn;
				*i = j;
			}
			if (*res==JEU_PERDU) {
				li->detruire();
				return;
			}
			if (couleur_premier_coup != couleur) {
				if (alpha < beta) {
					li->detruire();
					return;
				}
			}
		}
	}
}

//////////////////////////////////////////////////////////////////////
int Jeu::determiner_etat2(int t)
{
	int nombre_debut;
	int nombre_milieu;
	int nombre_fin;

	resultat = GAME_UNCALCULATED;
	taille_max = t;
	nombre_debut = Chaine::nombre_ch;
	formes_activees(JEU_DEBUT);
	Forme_reconnue::reconnaitre_formes_l(intersections);
	goban_affiche_pierre = FAUX;
	calcul_en_cours = VRAI;
	taille_etudiee = 0;
	couleur_premier_coup = couleur;
	initialiser_calculs();
	analyse_dynamique(
			couleur, &etat_couleur, &defense, JEU_GAGNE);
	nombre_milieu = Chaine::nombre_ch;
	taille_etudiee = 0;
	couleur_premier_coup = autre_couleur(couleur);
	initialiser_calculs();
	analyse_dynamique( autre_couleur(couleur),
			&etat_autre_couleur,
			&attaque, JEU_PERDU);
	nombre_fin = Chaine::nombre_ch;
	if ( (etat_couleur==JEU_INFINI) ||
			(etat_autre_couleur==JEU_INFINI) ) {
		resultat = GAME_KO;
	}
	if ( (etat_couleur==JEU_SATURE) ||
			(etat_autre_couleur==JEU_SATURE) ) {
		resultat = GAME_FUZZY;
```

```
	}
	if ( (resultat!=GAME_KO) && (resultat!=GAME_FUZZY) ) {
		resultat = etat_couleur - etat_autre_couleur;
		switch (resultat) {
		case GAME_STAR:
		case GAME_MORE:
			break;
		case 0: switch(etat_couleur) {
			case JEU_PERDU:
				resultat = GAME_NEGATIVE;
				attaque = defense = NULL;
				break;
			case JEU_INSTABLE:
				resultat = GAME_FUZZY;
				attaque = defense = NULL;
				break;
			case JEU_GAGNE:
				resultat = GAME_POSITIVE;
				valoriser_mauvais();
				attaque = defense = NULL;
				break;
			}
			break;
		case GAME_LESS:
		case GAME_ZERO:
			valoriser_mauvais();
			attaque = defense = NULL;
			break;
		default:
			fprintf(stdout, "jeu, determiner_etat2, erreur...\n");
			exit(-1);
		}
	}

	goban_affiche_pierre = VRAI;
	calcul_en_cours = FAUX;
	formes_activees(JEU_FIN);
	initialiser_calculs();
	relier_empreinte(VRAI);

	return (resultat);
}


////////////////////////////////////////////////////////////////////
Coup * Jeu::jouer_coup2(Intersection * i, int c)
{
	Coup * cp;

	EI::enlever_tout(&Regle_du_jeu::intersections_changees);
	cp = Coup::jouer2(i, c, &coups, NULL);
	Forme::reconnaitre_sur_EI(
		Regle_du_jeu::intersections_changees);
	identifier();
	profondeur_etudiee++;
	return cp;
}

////////////////////////////////////////////////////////////////////
void Jeu::reprendre_coup2(Coup * cp)
{
	EI::enlever_tout(&Regle_du_jeu::intersections_changees);

	cp->reprendre2(&coups);

	Forme::reconnaitre_sur_EI(
		Regle_du_jeu::intersections_changees);

	 cp->detruire();
	identifier();
	profondeur_etudiee--;
}
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// jeu_simple.h

#ifndef JEU_SIMPLE_H
#define JEU_SIMPLE_H

#include "objet.h"
#include "go_objet.h"
#include "objet_pose.h"
#include "jeu.h"
#include "goban_taille_max.h"
#include <stdio.h>

class Intersection;
class Liste;
class Ensemble_intersection;
class Groupe;
class Chaine;
class Jeu_simple : public Jeu
{
public:
	Jeu_simple(Ensemble_intersection *, int);
	~Jeu_simple();

	void generer_coups_defense(Liste **);
	void generer_coups_attaque(Liste **);
	void formes_activees(int);
	void ordonner_ataris_sur_ennemies(Liste *, Liste **);
	virtual void relier_empreinte(int);
	virtual void detruire();
	virtual void initialiser_calculs();
	virtual void analyse_statique(int, int *, Liste **);
	virtual void valoriser_mauvais();
	virtual void identifier();
	virtual Coup * jouer_coup2(Intersection *, int);
	virtual void reprendre_coup2(Coup *);

};

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// jeu_simple.c

#include "jeu_simple.h"
#include "jeu_define.h"

#include "chaine.h"
#include "pierre.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "ensemble_intersection_macro.h"
#include "objet_echelonne.h"
#include "intersection.h"
#include "regle_du_jeu.h"
#include "oeil.h"
#include "goban.h"
#include "goban_affiche.h"
#include "couleur.h"
#include "utile.h"
#include "print.h"
#include "liberte.h"

#include "autre_fr.h"
#include "dilate_fr.h"
#include "ko_fr.h"
#include "chaine_fr.h"
#include "oeil_fr.h"
#include "topo_fr.h"
#include "contact_fr.h"
#include "vide_fr.h"
#include "zone_fr.h"
#include "zizi_fr.h"
#include "fuseki_fr.h"

#include <stdio.h>
#include <stddef.h>
#include <sysent.h>


////////////////////////////////////////////////////////////////////
Jeu_simple::Jeu_simple(Ensemble_intersection * l, int c)
	: Jeu(l, c)
{
}


////////////////////////////////////////////////////////////////////
Jeu_simple::~Jeu_simple()
{
}

definir_destruction(Jeu_simple)

////////////////////////////////////////////////////////////////////
void Jeu_simple::generer_coups_attaque(Liste ** l)
{
	Ensemble_intersection * lr;
	Liste * lcapture;
	Liste * lv;
	Ensemble_intersection * lw_r;
	Ensemble_intersection * latari;
	Liste * liste_atari;
	Ensemble_intersection * lforme2;
	Intersection * i;
	Chaine * ch;
	Chaine * c;
	Chaine * cv;

	i = (Intersection *) intersections->premier();
	ch = Chaine::de_i_a_adresse(i, INDIFFERENT);

	switch(ch->libertes->taille()) {
	case 1:
		lr = new Ensemble_intersection();
		Regle_du_jeu::accessibles_non_oeil(
			ch->libertes, &lr, autre_couleur(ch->couleur), coups);
		lr->ou_logique_liste_avec_this(l);
		lr->detruire();
		return;
	case 2:
		lcapture = new Liste();
		ch->liste_chaines_ennemies_libertes(&lcapture, 1);
		while (lcapture->longueur>0) {
			c = (Chaine * ) lcapture->premier();

			lv = new Liste();
```

```
				c->liste_chaines_ennemies_libertes(&lv, 1);
				while (lv->longueur>0) {
					cv = (Chaine * ) lv->premier();

					lw_r = new Ensemble_intersection();
					Regle_du_jeu::accessibles_non_oeil(
						cv->libertes, &lw_r,
						autre_couleur(ch->couleur), coups);
					i = (Intersection *) lw_r->premier();
					if (i!=NULL) Liste::ou_logique(l, i);
					 lw_r->detruire();
					Liste::enlever(&lv, cv);
				}
				lv->detruire();

				lw_r = new Ensemble_intersection();
				Regle_du_jeu::accessibles_non_oeil(
					c->libertes, &lw_r,
					autre_couleur(ch->couleur), coups);
				i = (Intersection *) lw_r->premier();
				if (i!=NULL) {
					if (c->intersections->taille()<2) {
						Liste::ou_logique(l, i);
					}
					if ( Liberte::nombre_fictif(i,
						autre_couleur(ch->couleur)) > 1 ) {
						Liste::ou_logique(l, i);
					}
				}
				lw_r->detruire();
				Liste::enlever(&lcapture, c);
			}
			lcapture->detruire();

			latari = new Ensemble_intersection();
			liste_atari = new Liste();
			ch->libertes->ou_logique_liste(&latari);
			lr = new Ensemble_intersection();
			Regle_du_jeu::accessibles_non_oeil(
				latari, &lr, autre_couleur(ch->couleur), coups);
			lr->ou_logique_liste_avec_this(&liste_atari);
			ordonner_ataris(liste_atari, l);
			lr->detruire();
			latari->detruire();
			liste_atari->detruire();

			lforme2 = new Ensemble_intersection();
			ch->coups_de_forme(&lforme2, autre_couleur(ch->couleur));

			lr = new Ensemble_intersection();
			Regle_du_jeu::accessibles_non_oeil(
				lforme2, &lr, autre_couleur(ch->couleur), coups);
			lr->ou_logique_liste_avec_this(l);
			lr->detruire();
			Liste::inverser(l);
			lforme2->detruire();
			return;

		}
}

/////////////////////////////////////////////////////////////////////////
void Jeu_simple::generer_coups_defense(Liste ** l)
{
	int n;
	Ensemble_intersection * l2lib;
	Liste * lc2lib;
	Ensemble_intersection * l3lib;
	Liste * lc3lib;
	Liste * lcapture;
	Ensemble_intersection * lsortie;
	Ensemble_intersection * latari;
	Ensemble_intersection * lforme;
	Ensemble_intersection * lr;
	Ensemble_intersection * li_capture_3_pierres;
	Ensemble_intersection * li_capture_1_ou_2_pierres;
	Ensemble_intersection * li_sortie_3_libertes;
	Ensemble_intersection * li_sortie_1_ou_2_libertes;
	Ensemble_intersection * lw_r;
	Chaine * ch;
	Chaine * c;
	Intersection * i;


	i = (Intersection *) intersections->premier();
	ch = Chaine::de_i_a_adresse(i, INDIFFERENT);

	lcapture = new Liste();
	ch->liste_chaines_ennemies_libertes(&lcapture, 1);

	lforme = new Ensemble_intersection();
	li_capture_3_pierres = new Ensemble_intersection();
	li_capture_1_ou_2_pierres = new Ensemble_intersection();
	li_sortie_3_libertes = new Ensemble_intersection();
	li_sortie_1_ou_2_libertes = new Ensemble_intersection();

	while (lcapture->longueur>0) {

		c = (Chaine * ) lcapture->premier();
		lw_r = new Ensemble_intersection();
		Regle_du_jeu::accessibles_non_oeil(
			c->libertes, &lw_r, ch->couleur, coups);
		i = (Intersection *) lw_r->premier();
		if (i!=NULL) {
			if (c->intersections->taille()>2) {
				EI::ou_logique(&li_capture_3_pierres, i);
			}
			else {
				EI::ou_logique(&li_capture_1_ou_2_pierres, i);
			}
		}
		 lw_r->detruire();
		Liste::enlever(&lcapture, c);
	}
	lcapture->detruire();

	lsortie = new Ensemble_intersection();
	ch->libertes_fictives_maximales(
		ch->couleur, &n, &lsortie);
	if (n>2) {
		Regle_du_jeu::accessibles_non_oeil(
			lsortie, &li_sortie_3_libertes, ch->couleur, coups );
	}
	if (n<3) {
		Regle_du_jeu::accessibles_non_oeil(
			lsortie, &li_sortie_1_ou_2_libertes,
			ch->couleur, coups );
	}
	lsortie->detruire();

	switch(ch->libertes->taille()) {
	case 2:
		lr = new Ensemble_intersection();
		ch->coups_de_forme(&lr, ch->couleur);
		Regle_du_jeu::accessibles_non_oeil(
			lr, &lforme, ch->couleur, coups);
		lr->detruire();
		break;
	}

	li_sortie_1_ou_2_libertes->ou_logique_liste_avec_this(l);
	li_capture_1_ou_2_pierres->ou_logique_liste_avec_this(l);
	li_capture_3_pierres->ou_logique_liste_avec_this(l);
	li_sortie_3_libertes->ou_logique_liste_avec_this(l);
	lforme->ou_logique_liste_avec_this(l);

	if (li_sortie_3_libertes->taille()>0) {
		lforme->detruire();
		li_sortie_1_ou_2_libertes->detruire();
		li_sortie_3_libertes->detruire();
		li_capture_1_ou_2_pierres->detruire();
		li_capture_3_pierres->detruire();
		return;
	}

	if (li_capture_3_pierres->taille()>0) {
		lforme->detruire();
		li_sortie_1_ou_2_libertes->detruire();
		li_sortie_3_libertes->detruire();
		li_capture_1_ou_2_pierres->detruire();
		li_capture_3_pierres->detruire();
		return;
	}

	if (li_capture_1_ou_2_pierres->taille()>0) {
		lforme->detruire();
		li_sortie_1_ou_2_libertes->detruire();
		li_sortie_3_libertes->detruire();
		li_capture_1_ou_2_pierres->detruire();
		li_capture_3_pierres->detruire();
		return;
	}

	lforme->detruire();
	li_sortie_1_ou_2_libertes->detruire();
	li_sortie_3_libertes->detruire();
	li_capture_1_ou_2_pierres->detruire();
	li_capture_3_pierres->detruire();

	switch(ch->libertes->taille()) {
	case 1:
		if ((n==1) || (n==0)) Liste::enlever_tout(l);
		return;

	case 2:
		if (n==1) {
			Liste::enlever_tout(l);
		}
		l2lib = new Ensemble_intersection();
		latari = new Ensemble_intersection();
		lc2lib = new Liste();
		ch->liste_chaines_ennemies_libertes(&lc2lib, 2);
		if (lc2lib->longueur>0) {
			Chaine::de_liste_chaines_a_liste_libertes(
				lc2lib, &l2lib);
			Regle_du_jeu::accessibles_non_oeil(
				l2lib, &latari, ch->couleur, coups );
		}
```

```
		else {
			l3lib = new Ensemble_intersection();
			lc3lib = new Liste();
			ch->liste_chaines_ennemies_libertes(&lc3lib, 3);
			if (lc3lib->longueur>0) {
				Chaine::de_liste_chaines_a_liste_libertes(
					lc3lib, &l3lib);
				ch->libertes->enlever_liste(&l3lib);
				Regle_du_jeu::accessibles_non_oeil(
					l3lib, &latari, ch->couleur, coups );
			}
			l3lib->detruire();
			lc3lib->detruire();
		}

		latari->ou_logique_liste_avec_this(l);
		Liste::inverser(l);
		l2lib->detruire();
		lc2lib->detruire();
		latari->detruire();
		return;
	}
}

/////////////////////////////////////////////////////////////////////
void Jeu_simple::ordonner_ataris(Liste * liste, Liste ** l)
{
	Intersection * a;
	Intersection * b;
	Intersection * first;
	Intersection * second;
	int r, s;

	switch(liste->longueur) {
	case 0:
		return;
	case 1:
		liste->ou_logique_liste(l);
		return;
	case 2:
		break;
	default:
		fprintf(stdout, "Jeu_simple::ordonner_ataris:\n");
		fprintf(stdout, "\t Erreur, plus de 2 ataris envisages...\n");
		return;
	}

	a = (Intersection *) liste->premier();
	b = (Intersection *) liste->prochain();

	r = a->premiere_ligne();
	s = b->premiere_ligne();


	if ((r==0) && (s!=0)) {
		first = b;
		second = a;
	}
	if ((r!=0) && (s==0)) {
		first = a;
		second = b;
	}
	if ((r==0) && (s==0)) {
		first = a;
		second = b;
	}
	if ((r!=0) && (s!=0)) {
		first = a;
		second = b;
	}

	Liste::ou_logique(l, first);
	Liste::ou_logique(l, second);

}

/////////////////////////////////////////////////////////////////////
void Jeu_simple::formes_activees(int df)
{
	switch (df) {
	case JEU_DEBUT:
		Autre_fr::marche = VRAI;
		Dilate_fr::marche = FAUX;
		Ko_fr::marche = VRAI;
		Chaine_fr::marche = FAUX;
		Zone_fr::marche = FAUX;
		Zizi_fr::marche = FAUX;
		Contact_fr::marche = FAUX;
		Topo_fr::marche = FAUX;
		Oeil_fr::marche = VRAI;
		Vide_fr::marche = FAUX;
		Fuseki_fr::marche = FAUX;
		break;
	case JEU_FIN:
		Autre_fr::marche = FAUX;
		Dilate_fr::marche = VRAI;
		Ko_fr::marche = VRAI;
		Chaine_fr::marche = FAUX;
		Zone_fr::marche = FAUX;
		Zizi_fr::marche = FAUX;
		Contact_fr::marche = VRAI;
		Topo_fr::marche = VRAI;
		Oeil_fr::marche = VRAI;
		Vide_fr::marche = VRAI;
		Fuseki_fr::marche = VRAI;
		break;
	}
}
```

=+

```
// jeu_chaine.h

#ifndef JEU_CHAINE_H
#define JEU_CHAINE_H

#include "objet.h"
#include "go_objet.h"
#include "objet_pose.h"
#include "jeu.h"
#include "jeu_simple.h"
#include "goban_taille_max.h"
#include <stdio.h>

class Intersection;
class Liste;
class Ensemble_intersection;
class Groupe;
class Chaine;
class Jeu_chaine : public Jeu_simple
{
public:
	static Liste * liste_jeux_chaine;
	static Jeu_chaine * occupant[T_MAX][T_MAX];
	static Liste * empreintes[T_MAX][T_MAX];
	static int nombre_jc;

	Ensemble_intersection * mauvais;

	Jeu_chaine(Chaine *);
	~Jeu_chaine();

	static void relier(Ensemble_intersection *, Jeu_chaine *);
	void detruire();
	static void calculer_jeux(Ensemble_intersection *, Liste **);
	static Jeu_chaine * de_i_a_adresse(Intersection *, int);
	static Jeu_chaine * de_liste_a_adresse(
				Ensemble_intersection *, int);
	static void de_liste_a_liste(
		Ensemble_intersection *, Liste **, int, int);
	static int stable_G(int, Groupe *, Jeu_chaine **);

	void analyse_statique(int, int *, Liste **);
	void fimprimer(FILE *, int, int);

	void valoriser_empreinte();
	void valoriser_empreinte_interne();
	static void detruire_empreinteurs(Intersection *);
	void relier_empreinte(int);

	void valoriser_mauvais();
	void valoriser_double_atari(Intersection *);
};

#endif
```

=+

```
// jeu_chaine.c

#include "jeu_chaine.h"
#include "jeu_chaine_init.h"
#include "jeu_chaine_ihm.h"
#include "jeu_define.h"

#include "groupe.h"
#include "chaine.h"
#include "pierre.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "ensemble_intersection_macro.h"
#include "objet_echelonne.h"
#include "intersection.h"
#include "coup.h"
#include "regle_du_jeu.h"
#include "oeil_basique.h"
#include "goban.h"
#include "goban_affiche.h"
#include "couleur.h"
#include "utile.h"
#include "print.h"
#include "etage.h"
#include "liberte.h"

#include <stdio.h>
#include <stddef.h>
#include <sysent.h>

Liste * Jeu_chaine::liste_jeux_chaine;
Jeu_chaine * Jeu_chaine::occupant[T_MAX][T_MAX];
Liste * Jeu_chaine::empreintes[T_MAX][T_MAX];
```

```
int Jeu_chaine::nombre_jc = 0;

////////////////////////////////////////////////////////////////////////////
void jeu_chaine_init()
{
	if (ptrok(Jeu_chaine::liste_jeux_chaine)) {
		Go_objet::detruire_les_instances(
			Jeu_chaine::liste_jeux_chaine);
		Jeu_chaine::liste_jeux_chaine->detruire();
		Jeu_chaine::liste_jeux_chaine = NULL;
	}
	if (Jeu_chaine::liste_jeux_chaine==NULL)
		Jeu_chaine::liste_jeux_chaine = new Liste();

	Jeu_simple::debug_as		= FAUX;
	Jeu_simple::debug_as_gc	= FAUX;
	Jeu_chaine::debug_as		= FAUX;
	Jeu_chaine::debug_as_gc	= FAUX;
}

////////////////////////////////////////////////////////////////////////////
void jeu_chaine_init_tab(int v, int h)
{
	Jeu_chaine::occupant[v][h] = NULL;
	Jeu_chaine::empreintes[v][h] = new Liste();
}

////////////////////////////////////////////////////////////////////////////
void jeu_chaine_termin_tab(int v, int h)
{
	if (Jeu_chaine::empreintes[v][h]!=NULL) {
		 Jeu_chaine::empreintes[v][h]->detruire();
	}
}

////////////////////////////////////////////////////////////////////////////
Jeu_chaine::Jeu_chaine(Chaine * ch)
	: Jeu_simple(ch->intersections, ch->couleur)
{
	nombre_jc++;
	mauvais = new Ensemble_intersection();
	Liste::ajouter(&liste_jeux_chaine, this);
	relier(ch->intersections, this);
	intersections->frontiere_externe(&intersections_limite);
}

////////////////////////////////////////////////////////////////////////////
Jeu_chaine::~Jeu_chaine()
{

	relier_empreinte(FAUX);
	relier(intersections, NULL);
	Liste::enlever(&liste_jeux_chaine, this);
	if (mauvais!=NULL) mauvais->detruire();
	nombre_jc--;
}

definir_destruction(Jeu_chaine)

////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::relier(
Ensemble_intersection * l, Jeu_chaine * pointeur)
{
	DOLISTINT(l,i,
		occupant[i->numero_vertical][i->numero_horizontal] =
		pointeur;
	)
}

definir_liaison_empreinte(Jeu_chaine)

////////////////////////////////////////////////////////////////////////////
Jeu_chaine * Jeu_chaine::de_i_a_adresse(Intersection * i, int c)
{
	int v, h; if (c);

	if (i==NULL) return NULL;
	v = i->numero_vertical;
	h = i->numero_horizontal;
	switch (c) {
	case INDIFFERENT:return occupant[v][h];
	case BLANC:
	case NOIR:
	if ((occupant[v][h]!=NULL) && (occupant[v][h]->couleur==c))
		return occupant[v][h];
	else return NULL;
	}
}

////////////////////////////////////////////////////////////////////////////
Jeu_chaine  * Jeu_chaine::de_liste_a_adresse(
Ensemble_intersection * li, int c)
{
	Jeu_chaine * jc;
	Intersection * i;

	i = (Intersection*) li->premier();
	if (i==NULL) return NULL;
	jc = de_i_a_adresse( i, c);
	if (jc!=NULL) {
		if (jc->intersections->identique_a(li)==0) return jc;
	}
	return NULL;
}

////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::de_liste_a_liste(
Ensemble_intersection * li , Liste ** lo, int c,
int r)
{
	Jeu_chaine * jc;
	DOLISTINT(li,i,
		jc = de_i_a_adresse(i, c);
		if (jc!=NULL) {
			if (GAME_EVERY==r) {
				Liste::ou_logique(lo, jc);
			}
			else {
				if (jc->resultat==r) Liste::ou_logique(lo, jc);
			}
			jc->intersections->enlever_liste(&L);
		}
	)

}


////////////////////////////////////////////////////////////////////////////
int Jeu_chaine::stable_G(int a, Groupe * g, Jeu_chaine ** jjcc)
{
	Jeu_chaine * jc;

	not_used(a);

	if (g==NULL) {
		*jjcc = NULL;
		return GAME_UNCALCULATED;
	}
	jc = de_liste_a_adresse(g->intersections, g->couleur);
	if (jc!=NULL) {
		*jjcc = jc;
		return jc->resultat;
	}
	else  {
		*jjcc = NULL;
		return GAME_UNCALCULATED;
	}
}

////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::valoriser_mauvais()
{
	Chaine * ch;
	Intersection * i;
	Ensemble_intersection * li;
	int rc, rac;

	switch (resultat) {
	case GAME_POSITIVE:
		i = (Intersection *) intersections->premier();
		ch = (Chaine*) Chaine::de_i_a_adresse(i, couleur);
		if (ch!=NULL) {
			li = new Ensemble_intersection();
			ch->libertes->ajouter_liste(&li);
			for (;;) {
				i = (Intersection *) li->premier();
				if (i==NULL) break;
				rc = Liberte::nombre_fictif(
					i, couleur);
				rac = Liberte::nombre_fictif(
					i, autre_couleur(couleur));
				switch (rc) {
				case 0: // cas impossible theoriquement.
					printf("jeu chaine: cas impossible\n");
					EI::ou_logique(&mauvais, i);
					break;
				case 1:
					if (rac>0) {
						//printf("jeu chaine: tutu\n");
						EI::ou_logique(&mauvais, i);
					}
					break;
				}
				EI::enlever(&li, i);
			}
			li->detruire();
		}
		attaque = defense = NULL;
		break;
	case GAME_NEGATIVE: // il faut mettre qqchose ?
		break;
	case GAME_ZERO:
		i = (Intersection *) intersections->premier();
		ch = (Chaine*) Chaine::de_i_a_adresse(i, couleur);
		if (ch!=NULL) {
			ch->libertes->ou_logique_liste(&mauvais);
		}
		attaque = defense = NULL;
```

```
		break;
	default:
		break;
	}
}

//////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::valoriser_double_atari(Intersection * i)
{
	switch (resultat) {
	case GAME_POSITIVE:
		attaque = defense = i;
		resultat =
			GAME_POSITIVE_BUT_DOUBLE_IS_NEGATIVE;
		break;
	default:
		break;
	}
}

//////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::analyse_statique(
int trait, int * res, Liste ** l)
{
	int n;
	Chaine * ch;
	Intersection * i;
	EI * lr;

	i = (Intersection *) intersections->premier();
	ch = Chaine::de_i_a_adresse(i, couleur);


	if (ch == NULL) {
		*res = JEU_PERDU;
		return;
	}

	valoriser_empreinte();

	switch(ch->libertes->taille()) {
	case 2:
		n = Oeil_basique::yeux_vrais_chaine(i);
		if (n>1) {
			*res = JEU_GAGNE;
			return;
		}
		break;
	case 1:
		if (trait==autre_couleur(couleur)) {
			lr = new Ensemble_intersection();
			Regle_du_jeu::accessibles_non_oeil(
				ch->libertes, &lr, trait, coups);

			if (lr->taille()==0) {
				*res = JEU_INFINI;

			}

			else {
				lr->ou_logique_liste_avec_this(l);


				*res = JEU_PERDU;
			}
			lr->detruire();
			return;
		}
		break;
	default:
		*res = JEU_GAGNE;
		return;
	}

	if (trait==couleur) {
		generer_coups_defense(l);
		if ((*l)->longueur==0) {
			*res = JEU_PERDU;
			return;
		}
	}
	else {
		generer_coups_attaque(l);
		if ((*l)->longueur==0) {
			*res = JEU_GAGNE;
			return;
		}
	}

	*res = JEU_INSTABLE;
	return;
}

//////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::calculer_jeux(
Ensemble_intersection * li, Liste ** ljc)
{
	Jeu_chaine * jc;
	Chaine * ch;

	DOLISTINT(li,i,
		jc = Jeu_chaine::de_i_a_adresse(i, INDIFFERENT);
		ch = Chaine::de_i_a_adresse(i, INDIFFERENT);
		if (jc==NULL) {
			if (ch!=NULL) {
				jc = new Jeu_chaine(ch);
				jc->determiner_etat2(200);
			}
		}
		else {
			if (ch==NULL) {
				fprintf(stdout, "calculer_jeux_chaine: detruire
jc.\n");
				jc->detruire();
				jc = NULL;
			}
		}
		if ((ljc!=NULL) && ((*ljc)!=NULL) && (jc!=NULL))
			Liste::ou_logique(ljc, jc);
		if (jc!=NULL) jc->intersections->enlever_liste(&L);
	)

}



//////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::valoriser_empreinte()
{
	EI * ei;
	ei = new EI();
	intersections->ensemble_dilate_nd(&ei, 2);
	ei->ou_logique_liste(&empreinte);
	ei->detruire();
	valoriser_empreinte_interne();
}


//////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::valoriser_empreinte_interne()
{
	Chaine * ch;
	EI * ei;

	DOLISTINT(intersections_limite,j,
		ch = Chaine::de_i_a_adresse(j, INDIFFERENT);
		if (ch!=NULL) {
			ei = new EI();
			ch->intersections->ensemble_dilate_nd(&ei, 2);
			ei->ou_logique_liste(&empreinte);
			ei->detruire();
			ch->intersections->enlever_liste(&L);
		}
	)
}

//////////////////////////////////////////////////////////////////////////////
void Jeu_chaine::detruire_empreinteurs(Intersection * i)
{
	int v, h;
	v = i->numero_vertical;
	h = i->numero_horizontal;
	Go_objet::detruire_les_instances(Jeu_chaine::empreintes[v][h]);
}

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

3 - Interpretation du goban et decision globale

	interpretation.h
	interpretation.c
	global.h
	global.c

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// global.h

#ifndef GLOBAL_H
#define GLOBAL_H

#include "ressource.h"
#include "objet.h"
#include "go_objet.h"
#include "objet_pose.h"
#include "goban_taille_max.h"
#include <stdio.h>

class Intersection;
class Ensemble_intersection;
class Liste;
class Groupe;
class Global : public Objet_pose
{
public:
	 static Liste * liste_globals;
	static Global * occupante[T_MAX][T_MAX];

	 int valeur_regle_du_jeu;
```

```
        int valeur_intersection;
        int valeur_groupe_instable;
        int valeur_totale;
        Liste * instabilites;
        Liste * erreurs;
        Intersection * intersection;

        Global(Intersection *);
        ~Global();

        static Global * de_i_a_adresse(Intersection *);
        static void creer_les_instances();

        void detruire();

        static Intersection * decider(int);
        static Intersection * trouver(int);
        static Intersection * conclure(int);
        static int totaliser();

        static void valoriser_avec_regle_du_jeu (int);
        static void valoriser_avec_regle_du_jeu_interne (
                        Liste *, int);
        static void valoriser_avec_groupes_instables (int);
        static void valoriser_avec_groupes_instables_1 (int);
        static void valoriser_avec_groupes_instables_2 ();
        static void valoriser_avec_groupe_instable_1 (
                        Groupe *, int);
        static void valoriser_avec_groupe_instable_1_avec_liste (
                Ensemble_intersection *, Groupe *, int);
        void valoriser_groupes_instables_2 ();
        void valoriser_groupes_instables_2_liste (Liste *, int);

};

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// global.c

#include "global.h"
#include "global_init.h"
#include "global_ihm.h"

#include "objet_pose.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "objet_echelonne.h"
#include "intersection.h"
#include "jeu_intersection.h"
#include "groupe.h"
#include "territoire.h"
#include "sante.h"
#include "regle_du_jeu.h"
#include "chaine.h"
#include "forme_reconnue.h"
#include "forme_connue.h"
#include "goban.h"
#include "liberte.h"
#include "jeu_define.h"
#include "couleur.h"
#include "utile.h"
#include "lib_global.h"
#include "oeil_basique.h"
#include "pont_define.h"
#include "potentiel.h"
#include "partie.h"
#include "etage.h"
#include "vide.h"
#include "version.h"

#include <stdio.h>
#include <stddef.h>

#define GLOBAL_C_REGLE_DU_JEU     -9999

#define GLOBAL_C_GROUPE_INSTABLE      200

Liste * Global::liste_globals;
Global * Global::occupante[T_MAX][T_MAX];

//////////////////////////////////////////////////////////////////////////////////
void global_init()
{
        int i, j;

        if (Global::liste_globals==NULL) {
                Global::liste_globals = new Liste();
        }

        if (Regle_du_jeu::liste_coups==NULL)
                Regle_du_jeu::liste_coups = new Liste();

        if (Groupe::liste_coups==NULL)
                Groupe::liste_coups = new Liste();

        for (i=0; i<T_MAX; i++) {
                for (j=0; j<T_MAX; j++) {
                        Global::occupante[i][j] = NULL;
                }
```

```
        }
}

//////////////////////////////////////////////////////////////////////////////////
void global_init_bis()
{
        int i, j;

        for (i=0; i<T_MAX; i++) {
                for (j=0; j<T_MAX; j++) {
                        global_termin_tab(i,j);
                        Global::occupante[i][j] = NULL;
                }
        }
}

//////////////////////////////////////////////////////////////////////////////////
void global_init_tab(int v, int h)
{
        Intersection * i;
        global_termin_tab(v,h);
        i = Goban::courant->tableau[v][h];
        Global::occupante[v][h] = new Global(i);
}

//////////////////////////////////////////////////////////////////////////////////
void global_termin_tab(int v, int h)
{
        if (Global::occupante[v][h] != NULL)
                Global::occupante[v][h]->detruire();
}

//////////////////////////////////////////////////////////////////////////////////
Global::Global(Intersection * i)
        : Objet_pose(NULL, INDIFFERENT)
{
        if (liste_globals!=NULL) Liste::ajouter(&liste_globals, this);

        intersection = i;

        instabilites = new Liste();
        erreurs = new Liste();

        valeur_groupe_instable = 0;
        valeur_intersection = 0;
        valeur_regle_du_jeu = 0;
        valeur_totale = 0;

        occupante[i->numero_vertical][i->numero_horizontal] = this;
}

//////////////////////////////////////////////////////////////////////////////////
Global::~Global()
{

        if (ptrok(instabilites))  instabilites->detruire();
        if (ptrok(erreurs))  erreurs->detruire();

        if (liste_globals!=NULL)
                Liste::enlever(&liste_globals, this);

        if (Groupe::liste_coups!=NULL)
                Liste::enlever(&Groupe::liste_coups, this);

        if (Regle_du_jeu::liste_coups!=NULL)
                Liste::enlever(&Regle_du_jeu::liste_coups, this);

        occupante
                [intersection->numero_vertical]
                [intersection->numero_horizontal] = NULL;

}

definir_destruction(Global)

//////////////////////////////////////////////////////////////////////////////////
void Global::creer_les_instances()
{
        int v,h;
        DOLISTINT(Intersection::ensemble_intersections,i,
                v = i->numero_vertical;
                h = i->numero_horizontal;
                if (occupante[v][h] != NULL)  occupante[v][h]->detruire();
                occupante[v][h] = new Global(i);
        )

        Liste::enlever_tout(&Regle_du_jeu::liste_coups);
        Liste::enlever_tout(&Groupe::liste_coups);
}

//////////////////////////////////////////////////////////////////////////////////
Global * Global::de_i_a_adresse(Intersection * i)
{
        return occupante[i->numero_vertical][i->numero_horizontal];
}

//////////////////////////////////////////////////////////////////////////////////
Intersection * Global::decider(int c)
{
```

```
	Intersection * i;
	Global * d;
	creer_les_instances();
	valoriser_avec_regle_du_jeu(c);
	valoriser_avec_groupes_instables(c);
	if (Groupe::liste_coups->longueur==0) {
		i = Vide::decider(c);
		if (i==NULL) {
			fprintf(stdout, "Partie terminee.\n");
			fprintf(stdout, "Je pense passer.\n");
		}
		return i;
	}
	Forme_reconnue::valoriser_avec_formes(
		Intersection::ensemble_intersections, c);

	i = conclure(c);
	return i;

}

////////////////////////////////////////////////////////////////////////////////
Intersection * Global::conclure(int trait)
{
	Intersection * i;
	int m;

	not_used(trait);
	m = totaliser();
	if (m<=0) {
		fprintf(stdout, "conclure: Je pense passer.\n");
		i = NULL;
	}
	else {
		i = trouver(m);
	}
	return i;
}


////////////////////////////////////////////////////////////////////////////////
int Global::totaliser()
{
	Global * d;
	int v, h;
	int m = GLOBAL_COEF_REGLE_DU_JEU;
	DOLISTINT(Intersection::ensemble_intersections,i,
		d = (Global *) Global::de_i_a_adresse(i);
		if (d->valeur_regle_du_jeu ==
				GLOBAL_C_REGLE_DU_JEU) {
			d->valeur_totale = GLOBAL_C_REGLE_DU_JEU;
		}
		else {
			v = i->numero_vertical;
			h = i->numero_horizontal;
			d->valeur_totale =
				d->valeur_groupe_instable
				+ Forme_reconnue::somme[v][h]
				+ d->valeur_intersection;
		}
		if (d->valeur_totale>m) m = d->valeur_totale;
	)
	return m;
}

////////////////////////////////////////////////////////////////////////////////
Intersection * Global::trouver(int m)
{
	Intersection * i;
	Liste * l;

	i = NULL;
	l = new Liste();
	DOLIST(liste_globals,d,Global,
		if (d->valeur_totale == m) {
			i = d->intersection;
			Liste::ajouter(&l, i);
			break;
		}
	)
	if (l->longueur!=0) i = (Intersection *) l->element_hasard();
	l->detruire();
	if (i!=NULL) i->marquer_ihm('*');
	return i;
}

////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_regle_du_jeu(int c)
{
	Partie * pa;
	Liste * lc;

	lc = new Liste();
	pa = Partie::courante;
	if (pa!=NULL) {
		pa->coups->ajouter_liste(&lc);
		Liste::inverser(&lc);
	}
	valoriser_avec_regle_du_jeu_interne(lc, c);
	lc->detruire();
}

////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_regle_du_jeu_interne(
Liste * lc, int c)
{
	int r1, r2;

	DOLIST(liste_globals,d,Global,

		r1 = Regle_du_jeu::accessible(d->intersection, c, lc);
		r2 = Oeil_basique::oeil_vrai(d->intersection, c);
		if ( (r1==0) && (r2==-1) ) {
			Liste::ajouter(&Regle_du_jeu::liste_coups, d);
			d->valeur_regle_du_jeu = 0;
		}
		else {
			d->valeur_regle_du_jeu =
GLOBAL_C_REGLE_DU_JEU;
		}
	)

}

////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_groupes_instables(int c)
{
	valoriser_avec_groupes_instables_1(c);
	valoriser_avec_groupes_instables_2();
}

////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_groupes_instables_1(int c)
{
	DOLIST(Groupe::liste_groupes,g,Groupe,
		valoriser_avec_groupe_instable_1(g,c);
	)
}

////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_groupes_instables_2()
{
	DOLIST(Groupe::liste_coups,d,Global,
		d->valoriser_groupes_instables_2();
	)
}

////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_groupes_instables_2()
{
	Liste * l;
	int f;

	if (erreurs->longueur!=0) {
		l = erreurs;
		f = -1;
	}
	else {
		l = instabilites;
		f = +1;
	}

	valoriser_groupes_instables_2_liste(l, f);
}


////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_groupes_instables_2_liste(
Liste * l, int f)
{
	Objet_pose * op;
	op = new Objet_pose(NULL, INDIFFERENT);
	DOLIST(l,g,Groupe,
		g->intersections->ou_logique_liste(&(op->intersections));
	)
	valeur_groupe_instable =
		f*(op->valeur())*GLOBAL_COEF_GROUPE_INSTABLE;
	op->detruire();
}


////////////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_groupe_instable_1(
Groupe * g, int c)
{
	Ensemble_intersection * l;
	int f;

	if (g==NULL) return;
	if (g->sante==NULL) return;

	switch (g->etage) {
	case ETAGE_0:
		f = 1;
		l = NULL;
		switch(g->sante->etat) {
		case JEU_PERDU_OU_INCONNU:
		case JEU_INFINI:
```

```
                if (c==g->couleur) l = g->sante->defenses;
                else l = g->sante->attaques;
                valoriser_avec_groupe_instable_1_avec_liste(l,g,f);

                // une chaine en atari rend inutiles les coupes...
                if (c==g->couleur) l = g->sante->mauvaises_defenses;
                else l = g->sante->mauvaises_attaques;
                valoriser_avec_groupe_instable_1_avec_liste(l,g,-10);

                break;

            case JEU_INSTABLE_MAIS_DOUBLE_EST_PERDU:

                if (c==g->couleur) l = g->sante->defenses;
                else l = g->sante->attaques;
                valoriser_avec_groupe_instable_1_avec_liste(l,g,f);

                if (c==g->couleur) l = g->sante->mauvaises_defenses;
                else l = g->sante->mauvaises_attaques;
                valoriser_avec_groupe_instable_1_avec_liste(l,g,-1);

                break;

            case JEU_INSTABLE:

                if (c==g->couleur) l = g->sante->mauvaises_defenses;
                else l = g->sante->mauvaises_attaques;
                valoriser_avec_groupe_instable_1_avec_liste(l,g,-1);

                break;
            default:
                break;
            }
            break;
        case ETAGE_2:
        case ETAGE_3:

            l = NULL;
            f = 1;
            switch(g->sante->etat) {
            case JEU_ZERO:
                valoriser_avec_groupe_instable_1_avec_liste(
                    g->sante->interdites, g, -100);
                break;
            case JEU_PERDU: // pour les shiboris...
                if (c==g->couleur) f = 1;
                else f = -1;
                valoriser_avec_groupe_instable_1_avec_liste(
                    g->sante->defenses,g,f);
                break;
            case JEU_INSTABLE:
            case JEU_PERDU_OU_INCONNU:
            case JEU_INFINI:
            case JEU_INSTABLE_MAIS_DOUBLE_EST_PERDU:
                if (c==g->couleur) l = g->sante->defenses;
                else l = g->sante->attaques;
                valoriser_avec_groupe_instable_1_avec_liste(l,g,f);
                break;
            default:
                break;
            }
            break;
    }
}

//////////////////////////////////////////////////////////////////////////
void Global::valoriser_avec_groupe_instable_1_avec_liste(
Ensemble_intersection * l, Groupe * g, int f)
{
    Global * d;
    DOLISTINT(l,i,
        d = (Global *) de_i_a_adresse(i);
        if (f>0) {
            Liste::ou_logique(&(d->instabilites), g);
            Liste::ou_logique(&Groupe::liste_coups, d);
        }
        if (f<0) {
            Liste::ou_logique(&(d->erreurs), g);
        }
    )
}

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// interpretation.h

#ifndef INTERPRETATION_H
#define INTERPRETATION_H

#include "ressource.h"
#include "objet.h"
#include "go_objet.h"
#include "goban_taille_max.h"
#include "etage.h"
#include <stdio.h>

class Liste;
class Intersection;
class Ensemble_intersection;
class Joueur;
class Interpretation : public Go_objet
{
public:
        double score_noir;
        double score_blanc;
        double score_noir_moins_blanc;
        int occupant[N_ETAGE][T_MAX][T_MAX];
        Joueur * joueur;

        Interpretation(Joueur *);
        ~Interpretation();

        void detruire();

        static void interpreter(Ensemble_intersection *, int);
        static void interpreter_etage(Ensemble_intersection *, int);

        static int bouclage_tant_que_catastrophes(
                                        Ensemble_intersection *, int);
        static int bouclage_jusqu_a_catastrophe_ou_fin(
                Ensemble_intersection *, int);
        static int catastrophe_ou_fin(Ensemble_intersection *, int);
        static int bouclage_tant_que_attributs_non_values(
                Ensemble_intersection *, int);
        static int valoriser_un_attribut(Ensemble_intersection *, int);

        static int bouclage_tant_que_fusions(
                                        Ensemble_intersection *, int);
        static int
            bouclage_jusqu_a_fusion_ou_interactions_amies_calculees(
                Ensemble_intersection *, int);
        static int fusion_ou_interactions_amies_calculees(
                Ensemble_intersection *, int);
        static int occupant_C(Intersection *, int);
};

#endif

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// interpretation.c

#include "interpretation.h"

#include "couleur.h"
#include "etage.h"
#include "goban.h"
#include "jeu.h"
#include "intersection.h"
#include "pierre.h"
#include "chaine.h"
#include "oeil.h"
#include "groupe.h"
#include "zone.h"
#include "territoire.h"
#include "interaction_amie.h"
#include "interaction_ennemie.h"
#include "viditie.h"
#include "amitie.h"
#include "inimitie.h"
#include "combat.h"
#include "dilate.h"
#include "contact.h"
#include "pont.h"
#include "pont_define.h"
#include "barriere.h"
#include "barriere_define.h"
#include "sante.h"
#include "base_de_vie.h"
#include "encerclement.h"
#include "jeu.h"
#include "jeu_define.h"
#include "jeu_chaine.h"
#include "jeu_intersection.h"
#include "atari.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "objet_echelonne.h"
#include "objet_incremental.h"
#include "vide.h"
#include "version.h"

#include "regle_du_jeu.h"
#include "joueur.h"
#include "objet_incremental_define.h"

#include "utile.h"

#include <stdio.h>
#include <stddef.h>
#include <sysent.h>

#define N_CATASTROPHES_MAX              50
#define N_SANTES_MAX                    200
#define N_FUSIONS_MAX                   50
#define N_INTERACTIONS_AMIES_MAX        100
#define N_ATTRIBUTS_MAX                 600

//////////////////////////////////////////////////////////////////////////
```

```
Interpretation::Interpretation(Joueur * j)
{
	joueur = j;
	score_noir = 0.;
	score_blanc = 0.;
	score_noir_moins_blanc = 0.;
	if (joueur!=NULL) joueur->interpretation = this;
}

//////////////////////////////////////////////////////////////////////////////
Interpretation::~Interpretation()
{
	joueur->interpretation = NULL;
}

definir_destruction(Interpretation)

//////////////////////////////////////////////////////////////////////////////
void Interpretation::interpreter(
Ensemble_intersection * liste, int t)
{
	switch (t) {
	case ABSOLU:
		Objet_incremental::detruire_niveaux_inferieurs();
		break;
	case INCREMENTAL:
		break;
	}
	interpreter_etage(liste, ETAGE_0);
	interpreter_etage(liste, ETAGE_1);
	interpreter_etage(liste, ETAGE_2);
	Vide::determiner_espaces_vides();
}


//////////////////////////////////////////////////////////////////////////////
void Interpretation::interpreter_etage(
Ensemble_intersection * liste, int e)
{
	switch(e) {
	case ETAGE_0:
		Jeu_chaine::calculer_jeux(
			Intersection::ensemble_intersections, NULL);
		Atari::calculer_doubles(
			Intersection::ensemble_intersections);
		Groupe::creer_groupes_avec_liste_etage_inferieur(
			Jeu_chaine::liste_jeux_chaine, e);
		break;
	case ETAGE_1:
		Groupe::creer_groupes_avec_liste_etage_inferieur(
			Groupe::liste_groupes_non_caches_etage[e-1], e);
		if (bouclage_tant_que_catastrophes(liste, e)==-1) exit(-1);

		Dilate::creer_dilates(liste);
		Contact::creer_contacts(liste);
		Pont::creer_ponts(liste);
		Barriere::creer_barrieres(liste);
		break;
	case ETAGE_2:
		Groupe::creer_groupes_avec_liste_etage_inferieur(
			Groupe::liste_groupes_non_caches_etage[e-1], e);
		if (bouclage_tant_que_catastrophes(liste, e)==-1) exit(-1);

		break;

	default:
		fprintf(stdout, "Etage %d inconnu du concierge...\n" , e);
	}
}

//////////////////////////////////////////////////////////////////////////////
int Interpretation::bouclage_tant_que_catastrophes(
Ensemble_intersection * liste, int e)
{
	int r, n;

	for(n=1;n<=N_CATASTROPHES_MAX;n++) {

		if (e==ETAGE_2) {
			Territoire::etudier_territoires(e);
		}
		r = bouclage_jusqu_a_catastrophe_ou_fin(liste, e);
		switch (r) {
		case 0: // c'est fini
			return 0;
		case 1: // catastrophe, on continue
			break;
		case -1: // erreur
			return -1;
		}
	}

	if (n>N_CATASTROPHES_MAX) {
	fprintf(stdout, "bouclage_tant_que_catastrophes:\n") ;
	fprintf(stdout, "FIN ANORMALE.\n") ;
	fflush(stdout);
		return -1;
	}
	else {
		return 0;
	}
}

//////////////////////////////////////////////////////////////////////////////
int Interpretation::bouclage_jusqu_a_catastrophe_ou_fin(
Ensemble_intersection * liste, int e)
{
	int r, n;

	for(n=1;n<=N_SANTES_MAX;n++) {

		r = catastrophe_ou_fin(liste, e);

		switch (r) {
		case 0: // c'est fini
			return 0;
		case 1: // il y a eu une catastrophe
			return 1;
		case -1: break;
		case -2: // anomalie
			return -1;
		}
	}

	fprintf(stdout, "bouclage_jusqu_a_catastrophe_ou_fin:\n") ;
	fprintf(stdout, "FIN ANORMALE.\n") ;
	fflush(stdout);
	return -1;
}


//////////////////////////////////////////////////////////////////////////////
int Interpretation::catastrophe_ou_fin(
Ensemble_intersection * liste, int e)
{
	int a, b, r;

	not_used(liste);

	if (e==ETAGE_2) {
		a = bouclage_tant_que_fusions(liste, e);

		switch (a) {
		case 0: break;
		case -1:
			return -2; // anomalie
		}

		b = bouclage_tant_que_attributs_non_values(liste, e);

		switch (b) {
		case 0: break;
		case -1:
			return -2; // anomalie
		}
	}

	r = Sante::etudier_bonnes_santes(e);
	switch (r) {
	case 0:
		return -1;
	case 1:
		return 1;
	case -1:
		return 0;
	}

}


//////////////////////////////////////////////////////////////////////////////
int Interpretation::bouclage_tant_que_fusions(
Ensemble_intersection * liste, int e)
{
	int r, n;

	for(n=1;n<=N_FUSIONS_MAX;n++) {

		r =
		bouclage_jusqu_a_fusion_ou_interactions_amies_calculees(
				liste, e);

		switch (r) {
		case 0: // c'est fini
			return 0;
		case 1: // fusion, on continue
			break;
		case -1: // erreur
			return -1;
		}
	}

	if (n>N_FUSIONS_MAX) {
	fprintf(stdout, "bouclage_tant_que_fusions:\n") ;
	fprintf(stdout, "FIN ANORMALE.\n") ;
	fflush(stdout);
		return -1;
	}
```

```
	else {
		return 0;
	}
}

//////////////////////////////////////////////////////////////////////////
int  Interpretation::bouclage_tant_que_attributs_non_values(
Ensemble_intersection * liste, int e)
{
	int r, n;

	for( n=1; n<=N_ATTRIBUTS_MAX; n++ ) {
		r = valoriser_un_attribut(liste, e);
		if (r==0) break;
	}

	if (n>N_ATTRIBUTS_MAX) {
	fprintf(stdout,  "bouclage_tant_que_attributs_non_values:\n")  ;
	fprintf(stdout, "FIN ANORMALE.\n") ;
	fflush(stdout);
		return -1;
	}
	else {
		return 0;
	}
}

//////////////////////////////////////////////////////////////////////////
int Interpretation::
bouclage_jusqu_a_fusion_ou_interactions_amies_calculees(
Ensemble_intersection * liste, int e)
{
	int r, n;

	for(n=1;n<=N_INTERACTIONS_AMIES_MAX;n++) {

		r = fusion_ou_interactions_amies_calculees(liste, e);

		switch (r) {
		case 0: // c'est fini
			return 0;
		case 1: // il y a eu une fusion
			return 1;
		case -1: break; // on continue
			}
	}

	fprintf(stdout,
"bouclage_jusqu_a_fusion_ou_interactions_amies_calculees:\n")  ;
	fprintf(stdout, "FIN ANORMALE.\n") ;
	fflush(stdout);
	return -1;
}

//////////////////////////////////////////////////////////////////////////
int Interpretation::fusion_ou_interactions_amies_calculees(
Ensemble_intersection * liste, int e)
{
	int r;

	not_used(liste);

	r = Interaction_amie::etudier_interactions_amies(
			e, GAME_POSITIVE);

	switch (r) {
	case 0:
		return -1;
	case 1:
		return 1;
	case -1:
		return 0;
	}

}

//////////////////////////////////////////////////////////////////////////
int Interpretation::valoriser_un_attribut(
Ensemble_intersection * liste, int e)
{
	not_used(liste);

	if (Base_de_vie::etudier_bases_de_vie(e)==0)  return -1;
	if (Encerclement::etudier_encerclements(e)==0) return -1;
	if (Viditie::etudier_vidities(e)==0) return -1;
	if (Interaction_amie::etudier_interactions_amies(e, GAME_STAR)

							==0)
		return -1;
	if (Amitie::etudier_amities(e)==0) return -1;
	if (Interaction_ennemie::etudier_interactions_ennemies(e)==0)
		return -1;
	if (Inimitie::etudier_interdictions(e)==0) return -1;
	if (Inimitie::etudier_inimities_urgentes(e)==0) return -1;
	if (Inimitie::etudier_inimities_non_urgentes(e)==0) return -1;
	return 0;
}

//////////////////////////////////////////////////////////////////////////
int Interpretation::occupant_C(Intersection * in, int e)
{
	int c;

	c = Groupe::occupant_C(in, e);

	if (e!=ETAGE_2) return c;

	switch (c) {
	case BLANC:
	case NOIR:
		break;
	case INDIFFERENT:
		c = Territoire::occupant_C(in, e);
	}
	return c;
}
```

=+

4 - La classe groupe et un attribut: l'amitie:

groupe.h
groupe.c
amitie.h
amitie.c
interaction_amie.h
interaction_amie.c

=+

```
//  groupe.h

#ifndef GROUPE_H
#define GROUPE_H

#include  "objet.h"
#include  "go_objet.h"
#include  "objet_pose.h"
#include  "goban_taille_max.h"
#include  "etage.h"
#include  <stdio.h>

class Liste;
class Ensemble_intersection;
class Intersection;
class  Base_de_vie;
class Encerclement;
class Amitie;
class Inimitie;
class Viditie;
class Sante;
class Groupe : public Objet_pose
{
public:
	static  int  debug_as;
	static int compteur;
	static int nombre_gr[N_ETAGE];
	static Liste * liste_groupes;
	static Liste * liste_groupes_caches;
	static  Liste  *  liste_groupes_non_caches;
	static Liste * liste_groupes_etage[N_ETAGE];
	static Liste * liste_groupes_non_caches_etage[N_ETAGE];
	static Liste * liste_groupes_caches_etage[N_ETAGE];

	static Liste * blancs[N_ETAGE][T_MAX][T_MAX];
	static Liste * noirs[N_ETAGE][T_MAX][T_MAX];
	static Liste * empreintes[N_ETAGE][T_MAX][T_MAX];
	static Groupe * focus;
	static Liste * liste_coups;

	int cache;
	int etage;
	Ensemble_intersection * intersections_adherentes;
	Base_de_vie  *  base_de_vie;
	Encerclement * encerclement;
	int  fusions_a_jour;
	int interactions_amies_a_jour;
	Liste * interactions_amies;
	int  interactions_ennemies_a_jour;
	Liste * interactions_ennemies;
	Amitie * amitie;
	Inimitie * inimitie;
	Viditie * viditie;
	Sante * sante;

	Groupe(Intersection *, int, int);
	~Groupe();

	void detruire();
	static Groupe * chercher_groupe(Liste *, int);
	void  construire_is_adherentes();
	void construire_is_adherentes_avec_liste_territoires(Liste *);

	static  void  de_liste_a_liste(
		Ensemble_intersection *, Liste **, int, int, int);
	static  void  de_liste_a_liste_incluant(
		Ensemble_intersection *, Liste **, int, int, int);
```

```
	static void de_liste_a_liste_inclus(
		Ensemble_intersection *, Liste **, int, int, int);
	static void de_lg_a_lg_incluant(
		Ensemble_intersection *, Liste *, Liste **);
	static void de_lg_a_lg_inclus(
		Ensemble_intersection *, Liste *, Liste **);

	static void de_i_a_liste(Intersection *, int, int, int, Liste **);
	static void de_lg_a_lg_cache(Liste *, int, Liste **);
	static Groupe * de_i_a_adresse(Intersection *, int, int, int);
	static Groupe * diaa(Intersection *, int, int);

	void ajouter_intersection(Intersection *);
	void enlever_intersection(Intersection *);
	void ajouter_ensemble_intersections(Ensemble_intersection *);
	void enlever_ensemble_intersections(Ensemble_intersection *);

	void ajouter_is_g(Liste *);
	static int occupant_C(Intersection *, int);
	static void creer_groupes_avec_liste_etage_inferieur(Liste *, int);

	static void mettre_a_jour_caches();
	static void mettre_a_jour_caches_etage(int);
	static void cacher_les_instances(Liste *);
	static void detruire_les_attributs_interactifs_des_instances(
				Liste *);
	static void determiner_groupes_voisins_de_liste_groupes(
			Liste *, Liste **);

	void determiner_groupes_voisins(Liste **);
	void cacher(int);
	void cacher_instance(int);
	void cacher_attributs_locaux(int);
	void cacher_interactions(int);
	void cacher_interactions_amies(int);
	void cacher_interactions_ennemies(int);
	void cacher_attributs_interactifs(int);
	void detruire_attributs_interactifs();
	int decachable();
	int abstrait();

	void valoriser_empreinte(Liste *);
	void valoriser_empreinte_basique(int);
	void valoriser_empreinte_avec_potentiel();
	void valoriser_empreinte_avec_fenetre();
	void valoriser_empreinte_avec_ponts(Groupe *);
	void valoriser_empreinte_avec_ponts_interne(Liste *);
	static void detruire_empreinteurs(Intersection *, int);
	void relier_empreinte(int);

};

#endif

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// groupe.c

#include "groupe.h"
#include "groupe_init.h"
#include "groupe_ihm.h"
#include "lib_groupe.h"
#include "attribut_groupe.h"

#include "goban.h"
#include "intersection.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "objet_echelonne.h"
#include "pierre.h"
#include "chaine.h"
#include "chaine_fr.h"
#include "jeu_chaine.h"
#include "pont.h"
#include "pont_define.h"
#include "jeu_define.h"
#include "print_define.h"
#include "territoire.h"
#include "adherence.h"
#include "zone.h"
#include "etage.h"
#include "objet_incremental.h"

#include "base_de_vie.h"
#include "base_de_vie_init.h"
#include "encerclement.h"
#include "encerclement_init.h"
#include "amitie.h"
#include "amitie_init.h"
#include "interaction_amie.h"
#include "interaction_amie_init.h"
#include "inimitie.h"
#include "inimitie_init.h"
#include "interaction_ennemie.h"
#include "interaction_ennemie_init.h"
#include "viditie.h"
#include "viditie_init.h"
#include "sante.h"
#include "sante_init.h"
#include "controle.h"
#include "couleur.h"
#include "utile.h"
#include "fenetre_texte.h"

#include <stdio.h>
#include <stddef.h>

#define DEBUG_GROUPE_NUMERO

int Groupe::debug_as;
int Groupe::compteur;
int Groupe::nombre_gr[N_ETAGE];
Liste * Groupe::liste_groupes;
Liste * Groupe::liste_groupes_caches;
Liste * Groupe::liste_groupes_non_caches;
Liste * Groupe::liste_groupes_etage[N_ETAGE];
Liste * Groupe::liste_groupes_non_caches_etage[N_ETAGE];
Liste * Groupe::liste_groupes_caches_etage[N_ETAGE];
Liste * Groupe::blancs[N_ETAGE][T_MAX][T_MAX];
Liste * Groupe::noirs[N_ETAGE][T_MAX][T_MAX];
Liste * Groupe::empreintes[N_ETAGE][T_MAX][T_MAX];
Groupe * Groupe::focus;

Liste * Groupe::liste_coups;


////////////////////////////////////////////////////////////////////////////////
void groupe_init()
{
	groupe_init_global();

	groupe_init_etage(ETAGE_0);
	groupe_init_etage(ETAGE_1);
	groupe_init_etage(ETAGE_2);

	sante_init();
	base_de_vie_init();
	encerclement_init();
	inimitie_init();
	amitie_init();
	interaction_ennemie_init();
	interaction_amie_init();
	viditie_init();
}

////////////////////////////////////////////////////////////////////////////////
void groupe_init_global()
{
	if (ptrok(Groupe::liste_groupes)) {
		Go_objet::detruire_les_instances(Groupe::liste_groupes);
		Groupe::liste_groupes->detruire();
		Groupe::liste_groupes = NULL;
	}

	if (Groupe::liste_groupes==NULL)
		Groupe::liste_groupes = new Liste();

	if (ptrok(Groupe::liste_groupes_caches)) {
		Groupe::liste_groupes_caches->detruire();
		Groupe::liste_groupes_caches = NULL;
	}

	if (Groupe::liste_groupes_caches==NULL)
		Groupe::liste_groupes_caches = new Liste();

	if (ptrok(Groupe::liste_groupes_non_caches)) {
		Groupe::liste_groupes_non_caches->detruire();
		Groupe::liste_groupes_non_caches = NULL;
	}

	if (Groupe::liste_groupes_non_caches==NULL)
		Groupe::liste_groupes_non_caches = new Liste();

	Groupe::compteur = 0;
	Groupe::focus = NULL;
	Groupe::debug_as = FAUX;
}

////////////////////////////////////////////////////////////////////////////////
void groupe_init_etage(int e)
{
	if (ptrok(Groupe::liste_groupes_etage[e])) {
		Go_objet::detruire_les_instances(
				Groupe::liste_groupes_etage[e]);
		Groupe::liste_groupes_etage[e]->detruire();
		Groupe::liste_groupes_etage[e] = NULL;
	}
	if (Groupe::liste_groupes_etage[e]==NULL)
		Groupe::liste_groupes_etage[e] = new Liste();


	if (ptrok(Groupe::liste_groupes_caches_etage[e])) {
		Groupe::liste_groupes_caches_etage[e]->detruire();
		Groupe::liste_groupes_caches_etage[e] = NULL;
	}
	if (Groupe::liste_groupes_caches_etage[e]==NULL)
		Groupe::liste_groupes_caches_etage[e] = new Liste();
```

```
	if (ptrok(Groupe::liste_groupes_non_caches_etage[e])) {
		Groupe::liste_groupes_non_caches_etage[e]->detruire();
		Groupe::liste_groupes_non_caches_etage[e] = NULL;
	}
	if (Groupe::liste_groupes_non_caches_etage[e]==NULL)
		Groupe::liste_groupes_non_caches_etage[e] = new Liste();


	Groupe::nombre_gr[e] = 0;
}

/////////////////////////////////////////////////////////////////////////////
void groupe_init_tab(int v, int h)
{
	Groupe::noirs[ETAGE_0][v][h] = new Liste();
	Groupe::noirs[ETAGE_1][v][h] = new Liste();
	Groupe::noirs[ETAGE_2][v][h] = new Liste();

	Groupe::blancs[ETAGE_0][v][h] = new Liste();
	Groupe::blancs[ETAGE_1][v][h] = new Liste();
	Groupe::blancs[ETAGE_2][v][h] = new Liste();

	Groupe::empreintes[ETAGE_0][v][h] = new Liste();
	Groupe::empreintes[ETAGE_1][v][h] = new Liste();
	Groupe::empreintes[ETAGE_2][v][h] = new Liste();
}

/////////////////////////////////////////////////////////////////////////////
void groupe_termin_tab(int v, int h)
{
	if (Groupe::noirs[ETAGE_0][v][h]!=NULL)
		Groupe::noirs[ETAGE_0][v][h]->detruire();
	if (Groupe::noirs[ETAGE_1][v][h]!=NULL)
		Groupe::noirs[ETAGE_1][v][h]->detruire();
	if (Groupe::noirs[ETAGE_2][v][h]!=NULL)
		Groupe::noirs[ETAGE_2][v][h]->detruire();

	if (Groupe::blancs[ETAGE_0][v][h]!=NULL)
		Groupe::blancs[ETAGE_0][v][h]->detruire();
	if (Groupe::blancs[ETAGE_1][v][h]!=NULL)
		Groupe::blancs[ETAGE_1][v][h]->detruire();
	if (Groupe::blancs[ETAGE_2][v][h]!=NULL)
		Groupe::blancs[ETAGE_2][v][h]->detruire();

	if (Groupe::empreintes[ETAGE_0][v][h]!=NULL)
		Groupe::empreintes[ETAGE_0][v][h]->detruire();
	if (Groupe::empreintes[ETAGE_1][v][h]!=NULL)
		Groupe::empreintes[ETAGE_1][v][h]->detruire();
	if (Groupe::empreintes[ETAGE_2][v][h]!=NULL)
		Groupe::empreintes[ETAGE_2][v][h]->detruire();
}

/////////////////////////////////////////////////////////////////////////////
Groupe::Groupe(Intersection * i, int c, int e)
	: Objet_pose(NULL, c)
{
	etage = e;
	cache = FAUX;

	Liste::ou_logique(&liste_groupes, this);
	Liste::ou_logique(&liste_groupes_non_caches, this);
	Liste::ou_logique(&liste_groupes_etage[e], this);
	Liste::ou_logique(&liste_groupes_non_caches_etage[e], this);
	compteur++;
	nombre_gr[e]++;
	numero = compteur;
	intersections_adherentes = new Ensemble_intersection();
	base_de_vie = NULL;
	encerclement = NULL;
	interactions_amies = new Liste();
	interactions_amies_a_jour = FAUX;
	fusions_a_jour = FAUX;
	interactions_ennemies = new Liste();
	interactions_ennemies_a_jour = FAUX;
	amitie = NULL;
	inimitie = NULL;
	viditie = NULL;
	sante = NULL;
	ajouter_intersection(i);
}

/////////////////////////////////////////////////////////////////////////////
Groupe::~Groupe()
{
	Intersection * i;

	relier_empreinte(FAUX);
	if (ptrok(sante)) {
		sante->detruire();
		sante = NULL;
	}
	if (ptrok(inimitie)) {
		inimitie->detruire();
		inimitie = NULL;
	}
	if (ptrok(viditie)) {
		viditie->detruire();
		viditie = NULL;
	}
	if (ptrok(amitie)) {
		amitie->detruire();
		amitie = NULL;
	}
	if (ptrok(base_de_vie)) {
		base_de_vie->detruire();
		base_de_vie = NULL;
	}
	if (ptrok(encerclement)) {
		encerclement->detruire();
		encerclement = NULL;
	}

	if (intersections_adherentes!=NULL) {
		intersections_adherentes->detruire();
	}

	if (intersections!=NULL) {
		for (;;) {
			i = (Intersection *) intersections->premier();
			if (i==NULL) break;
			enlever_intersection(i);
		}
	}

	if (ptrok(interactions_amies)) {
		Go_objet::detruire_les_instances(interactions_amies);
		interactions_amies->detruire();
		interactions_amies = NULL;
	}
	if (ptrok(interactions_ennemies)) {
		Go_objet::detruire_les_instances(interactions_ennemies);
		interactions_ennemies->detruire();
		interactions_ennemies = NULL;
	}

	Liste::enlever(&liste_groupes_caches_etage[etage], this);
	Liste::enlever(&liste_groupes_non_caches_etage[etage], this);
	Liste::enlever(&liste_groupes_etage[etage], this);
	Liste::enlever(&liste_groupes_caches, this);
	Liste::enlever(&liste_groupes_non_caches, this);
	Liste::enlever(&liste_groupes, this);

	nombre_gr[etage]--;
}


definir_destruction(Groupe)

/////////////////////////////////////////////////////////////////////////////
void Groupe::relier_empreinte(int sens)
{
	DOLISTINT(empreinte,i,
		switch (sens) {
		case VRAI:
		Liste::ou_logique(
			&(empreintes[etage]
				[i->numero_vertical]
				[i->numero_horizontal]),
			this);
			break;
		case FAUX:
		Liste::enlever(
			&(empreintes[etage]
				[i->numero_vertical]
				[i->numero_horizontal]),
			this);
			break;
		}
	)
}


/////////////////////////////////////////////////////////////////////////////
void Groupe::ajouter_ensemble_intersections(
Ensemble_intersection * li)
{
	DOLISTINT(li,i,
		ajouter_intersection(i);
	)
}

/////////////////////////////////////////////////////////////////////////////
void Groupe::enlever_ensemble_intersections(
Ensemble_intersection * li)
{
	DOLISTINT(li,i,
		enlever_intersection(i);
	)
}

/////////////////////////////////////////////////////////////////////////////
void Groupe::ajouter_intersection(Intersection * i)
{
	int v, h;

	if (i==NULL) return;
	v = i->numero_vertical;
	h = i->numero_horizontal;
```

```
        EI::ou_logique(&intersections, i);
        switch (couleur) {
              case NOIR:Liste::ou_logique(&noirs[etage][v][h], this);
                          break;
              case BLANC:Liste::ou_logique(&blancs[etage][v][h], this);
                          break;
        }
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::enlever_intersection(Intersection * i)
{
        int v, h;

        if (i==NULL) return;
        v = i->numero_vertical;
        h = i->numero_horizontal;
        EI::enlever(&intersections, i);
        switch (couleur) {
              case NOIR:Liste::enlever(&noirs[etage][v][h], this);break;
              case BLANC:Liste::enlever(&blancs[etage][v][h], this);break;
        }
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::de_i_a_liste(
Intersection * i, int c, int e, int ch, Liste ** lg)
{
        int v, h;
        Liste * l;

        v = i->numero_vertical;
        h = i->numero_horizontal;

        switch(c) {
        case NOIR: l = noirs[e][v][h]; break;
        case BLANC: l = blancs[e][v][h]; break;
        }
        de_lg_a_lg_cache(l, ch, lg);
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::de_lg_a_lg_cache(Liste * l, int ch, Liste ** lg)
{
        DOLIST(l,g,Groupe,
              if (g->cache == ch) Liste::ou_logique(lg, g);
        )
}

////////////////////////////////////////////////////////////////////////////////
Groupe * Groupe::de_i_a_adresse(
ntersection * i, int c, int e, int ch)
{
        int v, h;

        v = i->numero_vertical;
        h = i->numero_horizontal;
        switch(c) {
        case NOIR:
              return Groupe::chercher_groupe (noirs[e][v][h], ch);
        case BLANC:
              return Groupe::chercher_groupe (blancs[e][v][h], ch);
        }
}

////////////////////////////////////////////////////////////////////////////////
Groupe * Groupe::diaa(Intersection * i, int e, int ch)
{
        int v, h;
        Groupe * g;

        v = i->numero_vertical;
        h = i->numero_horizontal;
        g = (Groupe *) Groupe::chercher_groupe (noirs[e][v][h], ch);
        if (g!=NULL) return g;
        g = (Groupe *) Groupe::chercher_groupe (blancs[e][v][h], ch);
        return g;
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::de_liste_a_liste(
Ensemble_intersection * li, Liste ** lg, int c, int e, int ch)
{
        DOLISTINT(li,i,
              de_i_a_liste(i, c, e, ch, lg);
        )
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::de_liste_a_liste_incluant(
Ensemble_intersection * li, Liste ** lg, int c, int e, int ch)
{
        Liste * l;
        l = new Liste();
        de_liste_a_liste(li, &l, c, e, ch);
        de_lg_a_lg_incluant(li, l, lg);
        l->detruire();
}
```

```
////////////////////////////////////////////////////////////////////////////////
void Groupe::de_lg_a_lg_incluant(
Ensemble_intersection * li, Liste * l, Liste ** lg)
{
        DOLIST(l,g,Groupe,
              if (li->inclus_dans(g->intersections)==0)
                    Liste::ou_logique(lg, g);
        )
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::de_liste_a_liste_inclus(
Ensemble_intersection * li, Liste ** lg, int c, int e, int ch)
{
        Liste * l;

        l = new Liste();
        de_liste_a_liste(li, &l, c, e, ch);
        de_lg_a_lg_inclus(li, l, lg);
        l->detruire();
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::de_lg_a_lg_inclus(
Ensemble_intersection * li, Liste * l, Liste ** lg)
{
        DOLIST(l,g,Groupe,
              if (g->intersections->inclus_dans(li)==0)
                    Liste::ou_logique(lg, g);
        )
}

////////////////////////////////////////////////////////////////////////////////
Groupe * Groupe::chercher_groupe(Liste * lg, int ch)
{

        DOLIST(lg,g,Groupe,
              if (g->cache == ch) break;
        )
        return g;
}


////////////////////////////////////////////////////////////////////////////////
void Groupe::ajouter_is_g(Liste * lg)
{
        DOLIST(lg,g,Groupe,
              ajouter_ensemble_intersections(g->intersections);
        )
}

////////////////////////////////////////////////////////////////////////////////
int Groupe::occupant_C(Intersection * in, int e)
{
        Groupe * gr;

        gr = Groupe::de_i_a_adresse(in, BLANC, e, FAUX);
        if (gr!=NULL) {
              return BLANC;
        }
        else {
              gr = Groupe::de_i_a_adresse(in, NOIR, e, FAUX);
              if (gr!=NULL) {
                    return NOIR;
              }
              else return INDIFFERENT;
        }
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::creer_groupes_avec_liste_etage_inferieur(
Liste * l, int e)
{
        Groupe * gg;
        Liste * lga;

        DOLIST(l,g,Objet_pose,

              lga = new Liste();
              de_liste_a_liste_incluant(
                    g->intersections, &lga, BLANC, e, FAUX);
              de_liste_a_liste_incluant(
                    g->intersections, &lga, BLANC, e, VRAI);
              de_liste_a_liste_incluant(
                    g->intersections, &lga, NOIR, e, FAUX);
              de_liste_a_liste_incluant(
                    g->intersections, &lga, NOIR, e, VRAI);
              if (lga->longueur>0) ;
              else {
                    gg = new Groupe(NULL, g->couleur, e);
                    gg->ajouter_ensemble_intersections(g->intersections);
                    gg->intersections->frontiere_externe(
                                &(gg->intersections_limite));
                    gg->valoriser_empreinte_basique(e);
                    gg->valoriser_empreinte(NULL);
                    gg->relier_empreinte(VRAI);

                    if (e==ETAGE_0) {
                          Sante::etage = e;
```

```
                Sante::etudier_bonne_sante(gg);
            }

        }
        lga->detruire();
    )
}


/////////////////////////////////////////////////////////////////////////////////
void Groupe::valoriser_empreinte(Liste * lg)
{
    Liste * ljc;
    Liste * lgg;
    Jeu_chaine * jc;
    Groupe * g;
    Sante * s;

    switch (etage) {
    case ETAGE_0:
        ljc = new Liste();
        Jeu_chaine::de_liste_a_liste(
            intersections, &ljc, couleur, GAME_EVERY);
        for (;;) {
            jc = (Jeu_chaine*) ljc->premier();
            if (jc==NULL) break;
            jc->empreinte->ou_logique_liste(&empreinte);
            Liste::enlever(&ljc, jc);
        }
        ljc->detruire();

        break;
    case ETAGE_1:
    case ETAGE_2:
        lgg = new Liste();
        if ( (lg==NULL) || (lg->longueur==0) ) {
            Groupe::de_liste_a_liste(
                intersections, &lgg, couleur, etage-1, FAUX);
        }
        else {
            lg->ajouter_liste(&lgg);
        }
        for (;;) {
            g = (Groupe*) lgg->premier();
            if (g==NULL) break;
            g->empreinte->ou_logique_liste(&empreinte);
            s = g->sante;
            if (s!=NULL) {
                s->empreinte->ou_logique_liste(&empreinte);
            }

            Liste::enlever(&lgg, g);
        }
        lgg->detruire();
        break;
    }
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::detruire_empreinteurs(Intersection * i, int e)
{
    DOLIST(empreintes[e][i->numero_vertical][i->numero_horizontal],
                g, Groupe,
        g->detruire();
    )

}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::valoriser_empreinte_basique(int e)
{
    switch(e) {
    case ETAGE_0:
        break;
    case ETAGE_1:
        break;
    case ETAGE_2:
        valoriser_empreinte_avec_potentiel();
        valoriser_empreinte_avec_fenetre();
        break;
    }
}


/////////////////////////////////////////////////////////////////////////////////
void Groupe::valoriser_empreinte_avec_potentiel()
{
    Ensemble_intersection * l;
    l = new Ensemble_intersection();
    intersections_adherentes->ensemble_dilate_obstacle_nd(
        &l, Adherence::distance_incrementale);
    l->ensemble_dilate(&empreinte);
    l->detruire();
    relier_empreinte(VRAI);
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::valoriser_empreinte_avec_fenetre()
{
    DOLISTINT(intersections,i,
        Forme_reconnue::fenetre(i, &empreinte, 5);
    )
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::valoriser_empreinte_avec_ponts(Groupe * g)
{
    Liste * lp; // les ponts qui contribuent a la fusion

    lp = new Liste();
    Pont::de_liste_a_liste(g->intersections_limite, &lp,
                g->couleur, GAME_POSITIVE);
    valoriser_empreinte_avec_ponts_interne(lp);
    lp->detruire();
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::valoriser_empreinte_avec_ponts_interne(
Liste * lp)
{
    DOLIST(lp,pt,Pont,
        pt->empreinte->ou_logique_liste(&empreinte);
    )
}


/////////////////////////////////////////////////////////////////////////////////
// 'abstrait' signifie que le groupe a ete construit au
// dessus de quelquechose
/////////////////////////////////////////////////////////////////////////////////
int Groupe::abstrait()
{
    int r;
    Liste * lg;

    lg = new Liste();

    de_liste_a_liste_inclus(intersections, &lg, autre_couleur(couleur),
                etage, FAUX);
    de_liste_a_liste_inclus(intersections, &lg, autre_couleur(couleur),
                etage, VRAI);

    switch(lg->longueur) {
    case 0:
        r = FAUX;
        break;
    default:
        r = VRAI;
        break;
    }

    lg->detruire();
    return r;

}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_les_instances(Liste * lg)
{
    DOLIST(lg,g,Groupe,
        g->cacher(VRAI);
    )
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::determiner_groupes_voisins_de_liste_groupes(
Liste * lg, Liste ** lgv)
{
    DOLIST(lg,g,Groupe,
        g->determiner_groupes_voisins(lgv);
    )
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::determiner_groupes_voisins(Liste ** lgv)
{
    if (ptrok(amitie)) amitie->voir_amis(lgv);
    if (ptrok(inimitie)) inimitie->voir_ennemis(lgv);
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::detruire_les_attributs_interactifs_des_instances(
Liste * lg)
{
    DOLIST(lg,g,Groupe,
        g->detruire_attributs_interactifs();
    )
}

/////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher(int sens)
{
    cacher_instance(sens);
    cacher_attributs_locaux(sens);
    if (sens==VRAI) cacher_interactions(sens);
    cacher_attributs_interactifs(sens);
}
```

```
////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_instance(int sens)
{
	cache = sens;

	switch(sens) {
	case VRAI:
		Liste::enlever(&liste_groupes_non_caches_etage[etage], this);
		Liste::ou_logique(&liste_groupes_caches_etage[etage], this);
		break;
	case FAUX:
		Liste::enlever(&liste_groupes_caches_etage[etage], this);
		Liste::ou_logique(
			&liste_groupes_non_caches_etage[etage], this);
		break;
	}
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_attributs_locaux(int sens)
{
	switch(etage) {
	case ETAGE_1:
		if (ptrok(sante)) sante->cacher(sens);
		break;
	case ETAGE_2:
		if (ptrok(base_de_vie)) base_de_vie->cacher(sens);
		if (ptrok(encerclement)) encerclement->cacher(sens);
		if (ptrok(viditie)) viditie->cacher(sens);
		break;
	}
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_attributs_interactifs(int sens)
{
	switch(etage) {
	case ETAGE_1:
		break;
	case ETAGE_2:
		if (ptrok(amitie)) amitie->cacher(sens);
		if (ptrok(inimitie)) inimitie->cacher(sens);
		if (ptrok(sante)) sante->cacher(sens);
		break;
	}
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_interactions(int sens)
{
	switch(etage) {
	case ETAGE_1:
		break;
	case ETAGE_2:
		cacher_interactions_amies(sens);
		cacher_interactions_ennemies(sens);
		break;
	}
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_interactions_amies(int sens)
{
	DOLIST(interactions_amies, ia, Interaction_amie,
		ia->cacher(sens);
	)
	interactions_amies_a_jour = FAUX;
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::cacher_interactions_ennemies(int sens)
{
	DOLIST(interactions_ennemies, ie, Interaction_ennemie,
		ie->cacher(sens);
	)

	interactions_ennemies_a_jour = FAUX;

}

////////////////////////////////////////////////////////////////////////////////
void Groupe::detruire_attributs_interactifs()
{
	switch(etage) {
	case ETAGE_1:
		break;
	case ETAGE_2:
		if (ptrok(amitie)) amitie->detruire();
		if (ptrok(inimitie)) inimitie->detruire();
		if (ptrok(sante)) sante->detruire();
		break;
	}
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::mettre_a_jour_caches()
{
	mettre_a_jour_caches_etage(ETAGE_1);
	mettre_a_jour_caches_etage(ETAGE_2);
}


////////////////////////////////////////////////////////////////////////////////
void Groupe::mettre_a_jour_caches_etage(int e)
{
	int r;
	DOLIST(liste_groupes_caches_etage[e],g,Groupe,
		r = g->decachable();
		if (r==0) g->cacher(FAUX);
	)
}

////////////////////////////////////////////////////////////////////////////////
int Groupe::decachable()
{
	int r;
	Liste * lg;
	Groupe * g;

	lg = new Liste();
	de_liste_a_liste_incluant(
			intersections, &lg, BLANC, etage, VRAI);
	de_liste_a_liste_incluant(
			intersections, &lg, BLANC, etage, FAUX);
	de_liste_a_liste_incluant(
			intersections, &lg, NOIR, etage, VRAI);
	de_liste_a_liste_incluant(
			intersections, &lg, NOIR, etage, FAUX);
	switch(lg->longueur) {
	case 0:
		r = -1;
		fprintf(stdout, "Groupe::decachable:\n");
		fprintf(stdout, "\tetrange, etrange !!!\n");
		break;
	case 1:
		g = (Groupe *) lg->premier();
		if (g==this) r = 0;
		else {
			r = -1;
			fprintf(stdout, "Groupe::decachable:\n");
			fprintf(stdout, "\tbizarre, bizarre !!!\n");
		}
		break;
	default:
		r = -1;
		break;
	}
	lg->detruire();
	return r;
}


////////////////////////////////////////////////////////////////////////////////
void Groupe::construire_is_adherentes()
{
	Liste * lt;

	intersections->ou_logique_liste(&intersections_adherentes);

	lt = new Liste();
	Territoire::de_liste_a_liste(
		ntersections_limite, &lt, couleur, etage);
	construire_is_adherentes_avec_liste_territoires(lt);
	lt->detruire();

	valoriser_empreinte_avec_potentiel();
}

////////////////////////////////////////////////////////////////////////////////
void Groupe::construire_is_adherentes_avec_liste_territoires(
Liste * lt)
{
	DOLIST(lt, t, Territoire,
		t->intersections->ou_logique_liste(
			&intersections_adherentes);
	)
}

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// amitie.h

#ifndef AMITIE_H
#define AMITIE_H

#include "objet.h"
#include "go_objet.h"
#include "objet_pose.h"
#include "goban_taille_max.h"
#include <stdio.h>

class Liste;
class Ensemble_intersection;
class Intersection;
class Groupe;
class Amitie : public Objet_pose
{
public:
	static int debug_as;
```

```
static Liste * liste_amities;
static Liste * liste_amities_cachees;
static Liste * liste_amities_non_cachees;
static int compteur;
static int nombre_am;
static int etage;
static Liste * empreintes[T_MAX][T_MAX];
static Amitie * focus;

int cache;
int resultat;
Groupe * groupe;
Ensemble_intersection * attaques;
Ensemble_intersection * defenses;

Amitie(Groupe *);
~Amitie();

static int etudier_amities(int);
static int amitie_G(int, Groupe *);
static int etudier_amitie(Groupe *);

int analyse_statique();
void voir_amis(Liste **);
virtual void detruire();
void cacher(int);
void cacher_instance(int);
void cacher_interactions(int);
void valoriser_empreinte(Liste *);
static void detruire_empreinteurs(Intersection *);
void relier_empreinte(int);

};

#endif
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

// amitie.c

#include "amitie.h"
#include "amitie_init.h"
#include "amitie_ihm.h"

#include "attribut_groupe.h"
#include "goban.h"
#include "groupe.h"
#include "etage.h"
#include "interaction_amie.h"
#include "interaction_amie_init.h"
#include "pont_define.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "ensemble_intersection_macro.h"
#include "objet_echelonne.h"
#include "intersection.h"
#include "jeu_intersection.h"
#include "couleur.h"
#include "utile.h"
#include "jeu_define.h"
#include "print_define.h"
#include "pont.h"
#include "controle.h"
#include "fenetre_texte.h"

#include <stream.h>
#include <stdio.h>
#include <stddef.h>

int Amitie::debug_as;
Liste * Amitie::liste_amities;
Liste * Amitie::liste_amities_cachees;
Liste * Amitie::liste_amities_non_cachees;
int Amitie::compteur;
int Amitie::nombre_am = 0;
int Amitie::etage;
Liste * Amitie::empreintes[T_MAX][T_MAX];
Amitie * Amitie::focus;

////////////////////////////////////////////////////////////////////////////
void amitie_init()
{
	if (ptrok(Amitie::liste_amities)) {
		Go_objet::detruire_les_instances(
			Amitie::liste_amities);
		Amitie::liste_amities->detruire();
		Amitie::liste_amities = NULL;
	}
	if (Amitie::liste_amities==NULL)
		Amitie::liste_amities = new Liste();
	if (ptrok(Amitie::liste_amities_cachees)) {
		Amitie::liste_amities_cachees->detruire();
		Amitie::liste_amities_cachees = NULL;
	}
	if (Amitie::liste_amities_cachees==NULL)
		Amitie::liste_amities_cachees = new Liste();
	if (ptrok(Amitie::liste_amities_non_cachees)) {
		Amitie::liste_amities_non_cachees->detruire();
		Amitie::liste_amities_non_cachees = NULL;
	}
	if (Amitie::liste_amities_non_cachees==NULL)
		Amitie::liste_amities_non_cachees = new Liste();
	Amitie::debug_as = FAUX;
	Amitie::focus = NULL;
	Amitie::compteur = 0;
	Amitie::nombre_am = 0;
}

////////////////////////////////////////////////////////////////////////////
void amitie_init_tab(int v, int h)
{
	Amitie::empreintes[v][h] = new Liste();
}

////////////////////////////////////////////////////////////////////////////
void amitie_termin_tab(int v, int h)
{
	if (Amitie::empreintes[v][h]!=NULL)
		Amitie::empreintes[v][h]->detruire();
}

////////////////////////////////////////////////////////////////////////////
Amitie::Amitie(Groupe * g)
	: Objet_pose(NULL, INDIFFERENT)
{
	Liste::ou_logique(&liste_amities, this);
	Liste::ou_logique(&liste_amities_non_cachees, this);
	compteur++;
	nombre_am++;
	numero = compteur;
	cache = FAUX;

	resultat = GAME_UNCALCULATED;

	attaques = new Ensemble_intersection();
	defenses = new Ensemble_intersection();

	groupe = g;
	couleur = g->couleur;
}

////////////////////////////////////////////////////////////////////////////
Amitie::~Amitie()
{
	relier_empreinte(FAUX);

	if (ptrok(groupe)) groupe->amitie = NULL;

	if (attaques!=NULL) {
		attaques->detruire();
		attaques = NULL;
	}
	if (defenses!=NULL) {
		defenses->detruire();
		defenses = NULL;
	}

	Liste::enlever(&liste_amities, this);
	Liste::enlever(&liste_amities_cachees, this);
	Liste::enlever(&liste_amities_non_cachees, this);

	nombre_am--;
}

definir_destruction(Amitie)
definir_liaison_empreinte(Amitie)

////////////////////////////////////////////////////////////////////////////
int Amitie::analyse_statique()
{
	int nombre = 0;
	Ensemble_intersection::enlever_tout(&attaques);
	Ensemble_intersection::enlever_tout(&defenses);
	DOLIST(groupe->interactions_amies,ia,Interaction_amie,
		if (ia->cache==FAUX) {
			ia->attaques->ou_logique_liste(&attaques);
			ia->defenses->ou_logique_liste(&defenses);
			ia->empreinte->ou_logique_liste(&empreinte);
			nombre++;
		}
	)
	groupe->empreinte->ou_logique_liste(&empreinte);
	switch (nombre) {
	case 0:
		resultat = GAME_NEGATIVE;
		break;
	case 1:
	case 2:
	case 3:
	case 4:
	case 5:
		resultat = GAME_STAR;
		break;
	default:
		resultat = GAME_POSITIVE;
		Ensemble_intersection::enlever_tout(&attaques);
		Ensemble_intersection::enlever_tout(&defenses);
		break;
	}
```

```
        relier_empreinte(VRAI);
        return 0;
}

////////////////////////////////////////////////////////////////////////////////////
void Amitie::voir_amis(Liste ** lag)
{
        DOLIST(groupe->interactions_amies,ia,Interaction_amie,
              if (ia->cache==FAUX) {
                    if (ptrok(ia->g2)) Liste::ou_logique(lag, ia->g2);
                    if (ptrok(ia->g1)) Liste::ou_logique(lag, ia->g1);
              }
        )
        Liste::enlever(lag, groupe);
}

////////////////////////////////////////////////////////////////////////////////////
int Amitie::etudier_amities(int e)
{
        int r = -1;
        etage = e;
        DOLIST(Groupe::liste_groupes_non_caches_etage[etage],
                    g,Groupe,
              r = etudier_amitie(g);
              if (r==0) break;
        )
        return r;

}

////////////////////////////////////////////////////////////////////////////////////
int Amitie::etudier_amitie(Groupe * g)
{
        if (g->etage!=etage) return -1;
        if (g->amitie!=NULL) return -1;
        g->amitie = new Amitie(g);
        return g->amitie->analyse_statique();
}

////////////////////////////////////////////////////////////////////////////////////
int Amitie::amitie_G(int a, Groupe * g)
{
        switch (a) {
        case CALCUL_OBEIGATOIRE:
              if (g->amitie!=NULL) {
                     g->amitie->detruire();
              }
              g->amitie = new Amitie(g);
              g->amitie->analyse_statique();
              break;
        case CALCUL_CONDITIONNEL:
              if (g->amitie==NULL) {
                    g->amitie = new Amitie(g);
                    g->amitie->analyse_statique();
              }
              break;
        case CALCUL_INTERDIT:
              if (g->amitie==NULL) return GAME_UNCALCULATED;
              break;
        }

        return g->amitie->resultat;
}


////////////////////////////////////////////////////////////////////////////////////
void Amitie::valoriser_empreinte(Liste * lg)
{
        not_used(lg);
}

////////////////////////////////////////////////////////////////////////////////////
void Amitie::detruire_empreinteurs(Intersection * i)
{
        Liste * l;
        int v, h;
        Amitie * b;

        v = i->numero_vertical;
        h = i->numero_horizontal;
        if ( empreintes[v][h] != NULL ) {
              l = new Liste();
              empreintes[v][h]->ajouter_liste(&l);
              for (;;) {
                    b = (Amitie *) l->premier();
                    if (b==NULL) break;
                    Liste::enlever(&l, b);
              }
              l->detruire();
              l = new Liste();
              empreintes[v][h]->ajouter_liste(&l);
              Go_objet::detruire_les_instances(l);
              l->detruire();
        }

}

////////////////////////////////////////////////////////////////////////////////////
void Amitie::cacher(int sens)
{
        cacher_instance(sens);
}

////////////////////////////////////////////////////////////////////////////////////
void Amitie::cacher_interactions(int sens)
{
        DOLIST(groupe->interactions_amies,ia,Interaction_amie,
              ia->cacher(sens);
        )
}

////////////////////////////////////////////////////////////////////////////////////
void Amitie::cacher_instance(int sens)
{
        cache = sens;

        switch(sens) {
        case VRAI:
              Liste::enlever(&liste_amities_non_cachees, this);
              Liste::ou_logique(&liste_amities_cachees, this);
              break;
        case FAUX:
              Liste::enlever(&liste_amities_cachees, this);
              Liste::ou_logique(&liste_amities_non_cachees, this);
              break;
        }
}

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

// interaction_amie.h

#ifndef INTERACTION_AMIE_H
#define INTERACTION_AMIE_H

#include "objet.h"
#include "go_objet.h"
#include "objet_pose.h"
#include "goban_taille_max.h"
#include <stdio.h>

class Liste;
class Ensemble_intersection;
class Intersection;
class Groupe;
class Amitie;
class Interaction_amie : public Objet_pose
{
public:
        static int debug_as;
        static Liste * liste_interactions_amies;
        static Liste * liste_interactions_amies_cachees;
        static Liste * liste_interactions_amies_non_cachees;
        static int compteur;
        static int nombre_ia;
        static int etage;
        static Liste * empreintes[T_MAX][T_MAX];
        static Interaction_amie * focus;

        int cache;
        int resultat;
        Liste * ponts;
        Liste * groupes_instables;
        Groupe * g1;
        Groupe * g2;

        Ensemble_intersection * attaques;
        Ensemble_intersection * defenses;

        Interaction_amie(Groupe *, Groupe *);
        ~Interaction_amie();

        static int etudier_interactions_amies(int, int);
        static int etudier_interactions_amies_un_groupe(
                    Groupe *, Liste **, int);
        static void determiner_amis_un_groupe(
                          Groupe *, int, Liste **);
        static void creer_analyser_interactions_amies_un_groupe(
              Groupe *, Liste *);
        static void determiner_groupes_avec_ponts_un_groupe (
              Groupe *, Liste *, Liste **);
        static void
              determiner_groupes_avec_groupes_instables_un_groupe (
              Groupe *, Liste *, Liste **);
        static void determiner_groupes_instables_un_groupe (
              Liste *, Liste **);

        int analyse_statique();
        int analyse_statique_avec_ponts();
        int analyse_statique_avec_groupes_instables();
        void determiner_ponts();
        void enlever_ponts_faux();
        void determiner_groupes_instables();
        void enlever_groupes_instables_faux();
        void detruire();
        void cacher(int);
        static Interaction_amie * existe_deja(Groupe *, Groupe *);
        void valoriser_empreinte();
```

```
	static void detruire_empreinteurs(Intersection *);
	void relier_empreinte(int);
	static void mettre_a_jour_caches();
	int decachable();
	static Groupe * fusionner(Liste *, int);
	void verifier_coups();
	void verifier_coups_attaque();
	void verifier_coups_defense();


};

#endif
```

=+

```
// interaction_amie.c

#include "interaction_amie.h"
#include "interaction_amie_init.h"
#include "interaction_amie_ihm.h"

#include "attribut_groupe.h"
#include "goban.h"
#include "groupe.h"
#include "amitie.h"
#include "pont_define.h"
#include "liste.h"
#include "ensemble_intersection.h"
#include "objet_echelonne.h"
#include "intersection.h"
#include "couleur.h"
#include "utile.h"
#include "jeu_intersection.h"
#include "jeu_define.h"
#include "print_define.h"
#include "pont.h"
#include "sante.h"
#include "fenetre_texte.h"

#include <stdio.h>
#include <stddef.h>

int Interaction_amie::debug_as;
Liste * Interaction_amie::liste_interactions_amies;
Liste * Interaction_amie::liste_interactions_amies_cachees;
Liste * Interaction_amie::liste_interactions_amies_non_cachees;
int Interaction_amie::compteur;
int Interaction_amie::nombre_ia = 0;
int Interaction_amie::etage;
Liste * Interaction_amie::empreintes[T_MAX][T_MAX];
Interaction_amie * Interaction_amie::focus;

////////////////////////////////////////////////////////////////////////////
void interaction_amie_init()
{
	if (ptrok(Interaction_amie::liste_interactions_amies)) {
		Go_objet::detruire_les_instances(
			Interaction_amie::liste_interactions_amies);
		Interaction_amie::liste_interactions_amies->detruire();
		Interaction_amie::liste_interactions_amies = NULL;
	}
	if (Interaction_amie::liste_interactions_amies==NULL)
		Interaction_amie::liste_interactions_amies = new Liste();

	if (ptrok(Interaction_amie::liste_interactions_amies_cachees)) {
		Interaction_amie::liste_interactions_amies_cachees
			->detruire();
		Interaction_amie::liste_interactions_amies_cachees = NULL;
	}
	if (Interaction_amie::liste_interactions_amies_cachees==NULL)
		Interaction_amie::liste_interactions_amies_cachees =
			new Liste();

	if (ptrok(Interaction_amie::liste_interactions_amies_non_cachees))
{
		Interaction_amie::liste_interactions_amies_non_cachees
			->detruire();
		Interaction_amie::liste_interactions_amies_non_cachees
			= NULL;
	}
	if (Interaction_amie::liste_interactions_amies_non_cachees
			==NULL)
		Interaction_amie::liste_interactions_amies_non_cachees
			= new Liste();


	Interaction_amie::debug_as = FAUX;
	Interaction_amie::focus = NULL;
	Interaction_amie::compteur = 0;
	Interaction_amie::nombre_ia = 0;
}

////////////////////////////////////////////////////////////////////////////
void interaction_amie_init_tab(int v, int h)
{
	Interaction_amie::empreintes[v][h] = new Liste();
}

////////////////////////////////////////////////////////////////////////////
void interaction_amie_termin_tab(int v, int h)
{
	if (Interaction_amie::empreintes[v][h]!=NULL)
		 Interaction_amie::empreintes[v][h]->detruire();
}

////////////////////////////////////////////////////////////////////////////
Interaction_amie::Interaction_amie(
Groupe * gun, Groupe * gdeux)
	: Objet_pose(NULL, INDIFFERENT)
{
	Liste::ajouter(&liste_interactions_amies, this);
	Liste::ajouter(&liste_interactions_amies_non_cachees, this);

	compteur++;
	nombre_ia++;
	numero = compteur;

	cache = FAUX;
	resultat = GAME_UNCALCULATED;

	g1 = gun;
	g2 = gdeux;
	couleur = gun->couleur;
	ponts = new Liste();
	groupes_instables = new Liste();

	attaques = new Ensemble_intersection();
	defenses = new Ensemble_intersection();

	if (ptrok(g1)) {
		if (ptrok(g1->interactions_amies)) {
			Liste::ajouter(&(g1->interactions_amies), this);
		}
	}
	if (ptrok(g2)) {
		if (ptrok(g2->interactions_amies)) {
			Liste::ajouter(&(g2->interactions_amies), this);
		}
	}
}

////////////////////////////////////////////////////////////////////////////
Interaction_amie::~Interaction_amie()
{
	relier_empreinte(FAUX);

	if (ptrok(g1)) {
		if (ptrok(g1->interactions_amies)) {
			Liste::enlever(&(g1->interactions_amies), this);
		}
		g1->interactions_amies_a_jour = FAUX;
	}

	if (ptrok(g2)) {
		if (ptrok(g2->interactions_amies)) {
			Liste::enlever(&(g2->interactions_amies), this);
		}
		g2->interactions_amies_a_jour = FAUX;
	}

	if (ptrok(ponts)) {
		ponts->detruire();
	}

	if (ptrok(groupes_instables)) {
		groupes_instables->detruire();
	}

	if (ptrok(attaques)) {
		attaques->detruire();
	}
	if (ptrok(defenses)) {
		defenses->detruire();
	}

	Liste::enlever(&liste_interactions_amies, this);
	Liste::enlever(&liste_interactions_amies_cachees, this);
	Liste::enlever(&liste_interactions_amies_non_cachees, this);

	nombre_ia--;
}

definir_destruction(Interaction_amie)
definir_liaison_empreinte(Interaction_amie)

////////////////////////////////////////////////////////////////////////////
int Interaction_amie::etudier_interactions_amies(int e, int s)
{
	Liste * lag;
	int r = -1;
	Groupe * gg;

	etage = e;
	lag = new Liste();

	DOLIST(Groupe::liste_groupes_non_caches_etage[etage],
			g,Groupe,
		r = etudier_interactions_amies_un_groupe(g, &lag, s);
```

```
		if (r==0) break;
		if (r==1) break;
	)

	if ( (ptrok(g)) && (r==1) && (lag!=NULL) ) {
		Liste::ou_logique(&lag, g);
		gg = fusionner(lag, e);
		lag->detruire();
		return 1;
	}

	lag->detruire();
	return r;
}

////////////////////////////////////////////////////////////////////////
int Interaction_amie::etudier_interactions_amies_un_groupe(
Groupe * g, Liste ** lag, int s)
{
	if (g->etage!=etage) return -1;
	switch (s) {
	case GAME_POSITIVE:
		if (g->fusions_a_jour==VRAI) return -1;

		determiner_amis_un_groupe(g, GAME_POSITIVE, lag);
		g->fusions_a_jour = VRAI;
		if ((lag!=NULL) &&
			(*lag!=NULL) && ((*lag)->longueur!=0)) {
			return 1;
		}
		else return 0;
	case GAME_STAR:
		if (g->interactions_amies_a_jour==VRAI) return -1;

		determiner_amis_un_groupe(g, GAME_STAR, lag);
		creer_analyser_interactions_amies_un_groupe(g, *lag);
		g->interactions_amies_a_jour = VRAI;
		return 0;
	}


}


////////////////////////////////////////////////////////////////////////
void Interaction_amie::determiner_amis_un_groupe(
Groupe * g, int r, Liste ** lag)
{
	Liste * lp;
	Liste * lq;
	lp = new Liste();
	Pont::de_liste_a_liste(g->intersections_limite, &lp,
			g->couleur, r);
	determiner_groupes_avec_ponts_un_groupe(g, lp, lag);
	lp->detruire();
	if (r==GAME_STAR) {
		lp = new Liste();
		lq = new Liste();
		Groupe::de_liste_a_liste(
			g->intersections_limite, &lp, autre_couleur(g->couleur),
			ETAGE_0, FAUX);
		determiner_groupes_instables_un_groupe(lp, &lq);
		determiner_groupes_avec_groupes_instables_un_groupe(
			g, lq, lag);
		lp->detruire();
		lq->detruire();
	}
	if (*lag!=NULL) Liste::enlever(lag, g);
}


////////////////////////////////////////////////////////////////////////
void Interaction_amie::
determiner_groupes_instables_un_groupe(
Liste * lg, Liste ** lh)
{
	DOLIST(lg,gg,Groupe,
		if (gg->sante==NULL) ;
		else {
			switch (gg->sante->etat) {
			case JEU_INFINI:
			case JEU_PERDU_OU_INCONNU:
			case JEU_INSTABLE_MAIS_DOUBLE_EST_PERDU:
				Liste::ou_logique(lh, gg);
				break;
			default:
				break;
			}
		}
	)

}

////////////////////////////////////////////////////////////////////////
void Interaction_amie::
determiner_groupes_avec_ponts_un_groupe(
Groupe * g, Liste * lp, Liste ** lag)
{
	Liste * lg;
	DOLIST(lp,gg,Groupe,
		lg = new Liste();
		Groupe::de_liste_a_liste(
			gg->intersections_limite, &lg,
			g->couleur,
			etage, FAUX);
		if (lg->rechercher(g)==0) {
			lg->ou_logique_liste(lag);
		}
		lg->detruire();
	)
}

////////////////////////////////////////////////////////////////////////
void Interaction_amie::
determiner_groupes_avec_groupes_instables_un_groupe(
Groupe * g, Liste * lp, Liste ** lag)
{
	Liste * lg;
	DOLIST(lp,gi,Groupe,
		lg = new Liste();
		Groupe::de_liste_a_liste(
			gi->intersections_limite, &lg,
			autre_couleur(gi->couleur),
			etage, FAUX);
		if (lg->rechercher(g)==0) {
			lg->ou_logique_liste(lag);
		}
		lg->detruire();
	)
}

////////////////////////////////////////////////////////////////////////
void Interaction_amie::
creer_analyser_interactions_amies_un_groupe(
Groupe * g, Liste * lag)
{
	Interaction_amie * ia;
	if (lag==NULL) return;
	DOLIST(lag,gg,Groupe,
		ia = existe_deja(g, gg);
		if (ia==NULL) {
			ia = new Interaction_amie(g, gg);
			ia->determiner_ponts();
			ia->determiner_groupes_instables();
			ia->analyse_statique();
			ia->valoriser_empreinte();
		}
	)
}


////////////////////////////////////////////////////////////////////////
Interaction_amie * Interaction_amie::existe_deja(
Groupe * g, Groupe * gg)
{
	DOLIST(g->interactions_amies, ia, Interaction_amie,
		if (ia->g2==gg) break;
		if (ia->g1==gg) break;
	)
	return ia;
}

////////////////////////////////////////////////////////////////////////
int Interaction_amie::analyse_statique()
{
	int r;
	r = analyse_statique_avec_groupes_instables();
	if (r==-1) analyse_statique_avec_ponts();
	resultat = GAME_STAR;
	return 0;
}

////////////////////////////////////////////////////////////////////////
int Interaction_amie::analyse_statique_avec_ponts()
{
	DOLIST(ponts,pt,Pont,
		pt->attaques->ou_logique_liste(&attaques);
		pt->defenses->ou_logique_liste(&defenses);
	)
	return 0;
}

////////////////////////////////////////////////////////////////////////
int Interaction_amie::analyse_statique_avec_groupes_instables()
{
	int r;
	r = -1;
	DOLIST(groupes_instables,gi,Groupe,
		if (gi->sante!=NULL) {
			gi->sante->attaques->ou_logique_liste(&defenses);
			gi->sante->defenses->ou_logique_liste(&attaques);
			if (gi->intersections->taille()>1) r = 0;
		}
	)
	return r;
}
```

```
///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::verifier_coups()
{
	verifier_coups_defense();
	verifier_coups_attaque();
}


///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::verifier_coups_attaque()
{
	Jeu_intersection * ji;
	DOLISTINT(attaques,i,
		ji = Jeu_intersection::de_i_a_adresse(i);
		if (ji==NULL) {
			ji = new Jeu_intersection(i, couleur);
			ji->determiner_etat2(100);
		}

		switch(ji->resultat) {
		case GAME_KO:
		case GAME_STAR:
			break;
		case GAME_POSITIVE:
			if (ji->couleur==couleur) {
				EI::enlever(&attaques, i);
			}
			break;
		case GAME_NEGATIVE:
			if (ji->couleur==autre_couleur(couleur)) {
				EI::enlever(&attaques, i);
			}
			break;
		default:
			EI::enlever(&attaques, i);
			break;
		}

	)
}


///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::verifier_coups_defense()
{
	Jeu_intersection * ji;
	DOLISTINT(defenses,i,
		ji = Jeu_intersection::de_i_a_adresse(i);
		if (ji==NULL) {
			ji = new Jeu_intersection(i, couleur);
			ji->determiner_etat2(100);
		}

		switch(ji->resultat) {
		case GAME_STAR:
		case GAME_KO:
			break;
		case GAME_POSITIVE:
			if (autre_couleur(couleur) == ji->couleur) {
				EI::enlever(&defenses, i);
			}
			break;
		case GAME_NEGATIVE:
			if (couleur == ji->couleur) {
				EI::enlever(&defenses, i);
			}
			break;
		default:
			EI::enlever(&defenses, i);
			break;
		}

	)

}

///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::determiner_ponts()
{
	Liste * lp1;
	Liste * lp2;

	lp1 = new Liste();
	lp2 = new Liste();
	Pont::de_liste_a_liste(g1->intersections_limite, &lp1,
				couleur, GAME_STAR);
	Pont::de_liste_a_liste(g2->intersections_limite, &lp2,
				couleur, GAME_STAR);
	lp1->ou_logique_liste(&ponts);
	lp2->et_logique_liste(&ponts);
	lp1->detruire();
	lp2->detruire();
	enlever_ponts_faux();
}

///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::determiner_groupes_instables()
{
	Liste * lg1;
	Liste * lg2;
	Liste * lh1;
	Liste * lh2;

	lg1 = new Liste();
	lg2 = new Liste();
	lh1 = new Liste();
	lh2 = new Liste();
	Groupe::de_liste_a_liste(g1->intersections_limite, &lg1,
				autre_couleur(couleur), ETAGE_0, FAUX);
	determiner_groupes_instables_un_groupe(lg1, &lh1);
	Groupe::de_liste_a_liste(g2->intersections_limite, &lg2,
				autre_couleur(couleur), ETAGE_0, FAUX);
	determiner_groupes_instables_un_groupe(lg2, &lh2);
	lh1->ou_logique_liste(&groupes_instables);
	lh2->et_logique_liste(&groupes_instables);
	lg1->detruire();
	lg2->detruire();
	lh1->detruire();
	lh2->detruire();
	enlever_groupes_instables_faux();
}

///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::enlever_ponts_faux()
{
	Liste * lg;
	DOLIST(ponts,pt,Pont,
		lg = new Liste();
		Groupe::de_liste_a_liste(pt->intersections_limite, &lg,
			couleur, Amitie::etage, FAUX);
		if (lg->longueur==1) Liste::enlever(&ponts, pt);
		lg->detruire();
	)
}


///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::enlever_groupes_instables_faux()
{
	Liste * lg;
	DOLIST(groupes_instables,gi,Groupe,
		lg = new Liste();
		Groupe::de_liste_a_liste(gi->intersections_limite, &lg,
			couleur, Amitie::etage, FAUX);
		if (lg->longueur==1) Liste::enlever(&groupes_instables, gi);
		lg->detruire();
	)
}

///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::valoriser_empreinte()
{
	DOLIST(ponts,pt,Pont,
		pt->empreinte->ou_logique_liste(&empreinte);
	)
	g1->empreinte->ou_logique_liste(&empreinte);
	g2->empreinte->ou_logique_liste(&empreinte);
	relier_empreinte(VRAI);
}

///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::detruire_empreinteurs(Intersection * i)
{
	int v, h;
	v = i->numero_vertical;
	h = i->numero_horizontal;
	Go_objet::detruire_les_instances(empreintes[v][h]);
}


///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::cacher(int sens)
{
	cache = sens;
	switch(sens) {
	case VRAI:
		Liste::enlever(&liste_interactions_amies_non_cachees, this);
		Liste::ou_logique(&liste_interactions_amies_cachees, this);
		break;
	case FAUX:
		Liste::enlever(&liste_interactions_amies_cachees, this);
		Liste::ou_logique(
			&liste_interactions_amies_non_cachees, this);
		break;
	}
	g1->interactions_amies_a_jour = FAUX;
	g2->interactions_amies_a_jour = FAUX;
}


///////////////////////////////////////////////////////////////////////////////////
void Interaction_amie::mettre_a_jour_caches()
{
	int r;

	DOLIST(liste_interactions_amies_cachees,ia,Interaction_amie,
		r = ia->decachable();
		if (r==0) {
```

```
			ia->cacher(FAUX);
		}
	)
}


////////////////////////////////////////////////////////////////////////////////
int Interaction_amie::decachable()
{
	if ((g1->cache==FAUX)&&(g2->cache==FAUX)) return 0;
	else return -1;
}





////////////////////////////////////////////////////////////////////////////////
// cette fonction fusionne des groupes entre eux.
// a l'origine un pont >.
// atenton: par convention, le premier groupe  de 'lgl' est la source.
////////////////////////////////////////////////////////////////////////////////
Groupe * Interaction_amie::fusionner(Liste * lgl, int etage)
{
	Groupe * g; // le groupe origine, (le premier de la liste)
	Groupe * ng;
	Liste * lgll;

	g = (Groupe*) lgl->premier();
	ng = new Groupe(NULL, g->couleur, etage);
	ng->ajouter_is_g(lgl);
	ng->intersections->frontiere_externe(&(ng->intersections_limite));
	ng->valoriser_empreinte(lgl);
	ng->valoriser_empreinte_avec_ponts(g);
	lgll = new Liste();
	Groupe::de_liste_a_liste(
		ng->intersections_limite, &lgll, BLANC, etage, FAUX);
	Groupe::de_liste_a_liste(
		ng->intersections_limite, &lgll, NOIR,  etage, FAUX);
	Groupe::determiner_groupes_voisins_de_liste_groupes(lgl,
&lgll);
	lgl->enlever_liste(&lgll);
	Groupe::detruire_les_attributs_interactifs_des_instances(lgl);
	Groupe::detruire_les_attributs_interactifs_des_instances(lgll);
	lgll->detruire();
	Groupe::cacher_les_instances(lgl);
	ng->intersections->frontiere_externe(&(ng->intersections_limite));
	ng->relier_empreinte(VRAI);
	return ng;
}
```

=+

5 - "Optimisation" la morphologie mathematique:

- objet_quadrille.h
- objet_quadrille.c
- peigne.h
- peigne.c

=+

```
//  objet_quadrille.h

#ifndef OBJET_QUADRILLE_H
#define OBJET_QUADRILLE_H

#include  "goban_taille_max.h"
#include  "entier.h"

#include  <stdio.h>

class Intersection;
class Objet_echelonne;
class Liste;
class Ensemble_intersection;
class Peigne;
class Objet_quadrille
{
public:
	static Objet_quadrille * oqp;
	static Objet_quadrille * oqq;

	static Entier_32 masque;
	static Entier_32 bord;

private:
	Peigne * vertical;
	Peigne * horizontal;

public:
	Objet_quadrille();
	~Objet_quadrille();

	static void init_masque();

	static void ou_logique(Objet_quadrille **, Objet_quadrille *);
	static void et_logique(Objet_quadrille **, Objet_quadrille *);
	static void enlever(Objet_quadrille **, Objet_quadrille *);

	void dilate(Objet_quadrille **);
	void dilate_nd(Objet_quadrille **, int);
	void dilate_obstacle(Objet_quadrille **);
	void dilate_obstacle_nd(Objet_quadrille **, int);

	void erode(Objet_quadrille **);
	void erode_ne(Objet_quadrille **, int);

	void frontiere_externe(Objet_quadrille **);
	void frontiere_interne(Objet_quadrille **);

	int inclus_dans(Objet_quadrille *);

	static void initialiser_OQ_avec_zero(Objet_quadrille **);
	static void initialiser_OQ_avec_EI(
		Objet_quadrille **, Ensemble_intersection *);
	static void initialiser_OQ_avec_intersection(
		Objet_quadrille **, Intersection *);
	static void ou_logique_OQ_avec_intersection(
		Objet_quadrille **, Intersection *);
	static void initialiser_OQ_avec_liste(
		Objet_quadrille **, Liste *);
	static void ou_logique_OQ_avec_liste(
		Objet_quadrille **, Liste *);

	void ou_logique_liste_avec_this(Liste **);
};

#endif
```

=+

```
//  objet_quadrille.c

#include  "objet_quadrille.h"

#include  "peigne.h"
#include  "intersection.h"
#include  "liste.h"
#include  "goban.h"
#include  "pierre.h"
#include  "utile.h"

#include  <stddef.h>
#include  <stdio.h>

Objet_quadrille  *  Objet_quadrille::oqp;
Objet_quadrille  *  Objet_quadrille::oqq;
Entier_32  Objet_quadrille::masque;
Entier_32  Objet_quadrille::bord;

////////////////////////////////////////////////////////////////////////////////
Objet_quadrille::Objet_quadrille()
{
	vertical = new Peigne();
	horizontal = new Peigne();
}

////////////////////////////////////////////////////////////////////////////////
Objet_quadrille::~Objet_quadrille()
{
	if (vertical!=NULL) delete vertical;
	if (horizontal!=NULL) delete horizontal;
}

////////////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::init_masque()
{
	switch(Goban::taille) {
	case 9:
		masque = 1022;
		bord = 1025;
		break;
	case 13:
		masque = 16382;
		bord = 16385;
		break;
	case 19:
		masque = 1048574;
		bord = 1048577;
		break;
	}
	Peigne::init_masque();
}


////////////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::initialiser_OQ_avec_zero(
Objet_quadrille ** oq)
{
	Peigne::initialiser_P_avec_zero(&((*oq)->vertical));
	Peigne::initialiser_P_avec_zero(&((*oq)->horizontal));
}

////////////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::initialiser_OQ_avec_intersection(
Objet_quadrille ** oq, Intersection * i)
{
	int v, h;
	int j;
```

```
	Entier_32  b;

	if (i==NULL) return;

	v = i->numero_vertical;
	h = i->numero_horizontal;

	b = 1;
	b <<= h;
	for (j=1; j<=Goban::taille; j++) {
		if (j!=v) (*oq)->vertical->t[j] = 0;
		else (*oq)->vertical->t[j] = b;
	}
	b = 1;
	b <<= v;
	for (j=1; j<=Goban::taille; j++) {
		if (j!=h) (*oq)->horizontal->t[j] = 0;
		else (*oq)->horizontal->t[j] = b;
	}
}

///////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::ou_logique_OQ_avec_intersection(
Objet_quadrille ** oq, Intersection * i)
{
	int v, h;
	Entier_32  b;

	if (i==NULL) return;

	v = i->numero_vertical;
	h = i->numero_horizontal;

	b = 1;
	b <<= h;
	(*oq)->vertical->t[v] = ((*oq)->vertical->t[v]) | b;
	b = 1;
	b <<= v;
	(*oq)->horizontal->t[h] = ((*oq)->horizontal->t[h]) | b;
}

///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::initialiser_OQ_avec_liste(
Objet_quadrille ** oq, Liste * lh)
{
	initialiser_OQ_avec_zero(oq);
	ou_logique_OQ_avec_liste(oq,  lh);
}

///////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::ou_logique_OQ_avec_liste(
Objet_quadrille ** oq, Liste * lh)
{
	DOLIST(lh,i,Intersection,
		ou_logique_OQ_avec_intersection(oq,  i);
	)
}

///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::initialiser_OQ_avec_EI(
Objet_quadrille ** oq, EI * li)
{
	Intersection * i;
	int j, k;

	initialiser_OQ_avec_zero(oq);

	for (j=1; j<=Goban::taille; j++) {
		for (k=1; k<=Goban::taille; k++) {
			i = Goban::courant->tableau[j][k];
			if (li->rechercher(i)==0) {
				ou_logique_OQ_avec_intersection(oq,  i);
			}
		}
	}
}

///////////////////////////////////////////////////////////////////////////
// cette fonction remplit 'li' avec l'objet quadrille
///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::ou_logique_liste_avec_this(Liste  **  li)
{
	Intersection * i;
	int j, k;
	Entier_32 b, c;

	if ((*li)==NULL) return;

	for (j=1; j<=Goban::taille; j++) {
		if (vertical->t[j]!=0) {
			b = 1;
			for (k=1; k<=Goban::taille; k++) {
				b <<= 1;
				c = b & vertical->t[j];
				if (b == c) {
					i = Goban::courant->tableau[j][k];
					if (ptrok(i)) Liste::ou_logique(li, i);
				}
			}
			}
	}

}

///////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::ou_logique(
Objet_quadrille ** oq,  Objet_quadrille * chose )
{
	Peigne::ou_logique(&((*oq)->vertical), chose->vertical);
	Peigne::ou_logique(&((*oq)->horizontal), chose->horizontal);
}

///////////////////////////////////////////////////////////////////////////
void  Objet_quadrille::et_logique(
Objet_quadrille ** oq,    Objet_quadrille * chose )
{
	Peigne::et_logique(&((*oq)->vertical), chose->vertical);
	Peigne::et_logique(&((*oq)->horizontal), chose->horizontal);
}

///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::enlever(
Objet_quadrille ** oq, Objet_quadrille * chose )
{
	Peigne::enlever(&((*oq)->vertical), chose->vertical);
	Peigne::enlever(&((*oq)->horizontal), chose->horizontal);
}

///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::dilate( Objet_quadrille ** chose )
{
	int i, j;
	Entier_32 a, b, c, d;

	vertical->dilate(&((*chose)->vertical));
	horizontal->dilate(&((*chose)->horizontal));

	b = 1;
	for(i=1; i<=Goban::taille; i++) {
		b <<= 1;
		a = 1;
		for(j=1; j<=Goban::taille; j++) {
			a <<= 1;
			c = ((*chose)->vertical->t[i]) & a ;
			d = ((*chose)->horizontal->t[j]) & b;
			if ( (d == b) || (c == a) ) {
		(*chose)->horizontal->t[j] = ((*chose)->horizontal->t[j]) | b;
		(*chose)->vertical->t[i] = ((*chose)->vertical->t[i]) | a;
			}
		}
	}

}

///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::dilate_nd(
Objet_quadrille ** chose, int nd )
{
	int i;
	Objet_quadrille * lin;
	Objet_quadrille * lout;

	lin = new Objet_quadrille();
	lout = new Objet_quadrille();

	ou_logique(&lin, this);

	for (i=0; i<nd; i++) {
		ou_logique(&lin, lout);
		lin->dilate(&lout);
	}

	ou_logique(chose, lout);

	delete lin;
	delete lout;
}


///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::dilate_obstacle(
Objet_quadrille ** chose )
{
	frontiere_externe(chose);
	et_logique(chose, Pierre::vide);
	ou_logique(chose, this);
}

///////////////////////////////////////////////////////////////////////////
void Objet_quadrille::dilate_obstacle_nd(
Objet_quadrille ** chose, int nd )
{
	int i;
	Objet_quadrille * lin;
	Objet_quadrille * lout;

	lin = new Objet_quadrille();
	lout = new Objet_quadrille();
```

```
	ou_logique(&lin, this);

	for (i=0; i<nd; i++) {
		ou_logique(&lin, lout);
		lin->dilate_obstacle(&lout);
	}

	ou_logique(chose, lout);

	delete lin;
	delete lout;
}

//////////////////////////////////////////////////////////////////////////////////
void Objet_quadrille::erode( Objet_quadrille ** chose )
{
	int i, j;
	Entier_32 a, b, c, d;

	vertical->erode(&((*chose)->vertical));
	horizontal->erode(&((*chose)->horizontal));

	b = 1;
	for(i=1; i<=Goban::taille; i++) {
		b <<= 1;
		a = 1;
		for(j=1; j<=Goban::taille; j++) {
			a <<= 1;
			c = ((*chose)->vertical->t[i]) & a;
			d = ((*chose)->horizontal->t[j]) & b;
			if (d == 0) {
			(*chose)->vertical->t[i] =
				(*chose)->vertical->t[i] & (~a) & masque;
			}
			if (c == 0) {
			(*chose)->horizontal->t[j] =
				(*chose)->horizontal->t[j] & (~b) & masque;
			}
		}
	}
}

//////////////////////////////////////////////////////////////////////////////////
void Objet_quadrille::erode_ne(
Objet_quadrille ** chose, int ne )
{
	int i;
	Objet_quadrille * lin;
	Objet_quadrille * lout;

	lin = new Objet_quadrille();
	lout = new Objet_quadrille();

	ou_logique(&lin, this);

	for (i=0; i<ne; i++) {
		lin->erode(&lout);
		initialiser_OQ_avec_zero(&lin);
		ou_logique(&lin, lout);
	}

	ou_logique(chose, lout);

	delete lin;
	delete lout;
}

//////////////////////////////////////////////////////////////////////////////////
void Objet_quadrille::frontiere_interne(
Objet_quadrille ** chose )
{
	Objet_quadrille * oq;
	oq = new Objet_quadrille();
	erode(&oq);
	ou_logique(chose, this);
	enlever(chose, oq);
	//oq->detruire();
	delete oq;
}

//////////////////////////////////////////////////////////////////////////////////
void Objet_quadrille::frontiere_externe(
Objet_quadrille ** chose )
{
	dilate(chose);
	enlever(chose, this);
}

//////////////////////////////////////////////////////////////////////////////////
int Objet_quadrille::inclus_dans(Objet_quadrille * chose)
{
	int i;
	Entier_32 b;
	for(i=1; i<=Goban::taille; i++) {
		b = vertical->t[i] & chose->vertical->t[i];
		if (vertical->t[i] != b) return -1;
	}
	return 0;
}

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
// peigne.h

#ifndef PEIGNE_H
#define PEIGNE_H

#include "goban_taille_max.h"
#include "entier.h"

#include <stdio.h>

class Peigne
{
public:

	static Entier_32 masque;
	static Entier_32 bord;

	Entier_32 t[T_MAX];

	Peigne();
	~Peigne();

	static void init_masque();

	static void ou_logique(Peigne **, Peigne *);
	static void et_logique(Peigne **, Peigne *);
	static void enlever(Peigne **, Peigne *);

	void dilate(Peigne **);
	void dilate_obstacle(Peigne **);
	void dilate_obstacle_nd(Peigne **, int);
	void erode(Peigne **);
	void frontiere_externe(Peigne **);
	void frontiere_interne(Peigne **);

	static void initialiser_P_avec_zero(Peigne **);
};

#endif

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

// peigne.c

#include "peigne.h"

#include "goban.h"
#include "utile.h"

#include <stddef.h>
#include <stdio.h>

Entier_32 Peigne::masque;
Entier_32 Peigne::bord;

//////////////////////////////////////////////////////////////////////////////////
Peigne::Peigne()
{
	int i;
	for (i=0; i<T_MAX; i++) {
		t[i] = 0;
	}
}

//////////////////////////////////////////////////////////////////////////////////
Peigne::~Peigne()
{
}

//////////////////////////////////////////////////////////////////////////////////
void Peigne::init_masque()
{
	switch(Goban::taille) {
	case 9:
		masque = 1022;
		bord = 1025;
		break;
	case 13:
		masque = 16382;
		bord = 16385;
		break;
	case 19:
		masque = 1048574;
		bord = 1048577;
		break;
	}
}

//////////////////////////////////////////////////////////////////////////////////
void Peigne::initialiser_P_avec_zero(Peigne ** p)
{
	int i;
	for (i=0; i<T_MAX; i++) {
		(*p)->t[i] = 0;
	}
}
```

```
///////////////////////////////////////////////////////////////////////////
void  Peigne::ou_logique(
Peigne ** oq,   Peigne * chose )
{
        int i;
        for(i=1; i<=Goban::taille; i++) {
              (*oq)->t[i] =
                    ((*oq)->t[i]) | chose->t[i];
        }
}

///////////////////////////////////////////////////////////////////////////
void  Peigne::et_logique(
Peigne ** oq,    Peigne * chose )
{
        int i;
        for(i=1; i<=Goban::taille; i++) {
              (*oq)->t[i] =
                    (*oq)->t[i] & chose->t[i];
        }
}

///////////////////////////////////////////////////////////////////////////
void Peigne::enlever(
Peigne ** oq, Peigne * chose )
{
        int i;
        Entier_32 a;
        for(i=1; i<=Goban::taille; i++) {
              a = ~(chose->t[i]);
              a = a & masque;
              (*oq)->t[i] = (*oq)->t[i] & a;
        }
}

///////////////////////////////////////////////////////////////////////////
void Peigne::dilate( Peigne ** chose )
{
        int i;
        Entier_32 e, f, g;
        for(i=1; i<=Goban::taille; i++) {
              e = t[i];
              f = t[i]; f >>= 1;
              g = t[i]; g <<= 1;
              (*chose)->t[i] = g | e | f;
              (*chose)->t[i] = ((*chose)->t[i]) & masque;
        }

}

///////////////////////////////////////////////////////////////////////////
void Peigne::erode( Peigne ** chose )
{
        int i;
        Entier_32 e, f, g;
        for(i=1; i<=Goban::taille; i++) {
              e = t[i] | bord;
              f = t[i] | bord; f >>= 1;
              g = t[i] | bord; g <<= 1;
              (*chose)->t[i] = g & e & f;
              (*chose)->t[i] = ((*chose)->t[i]) & masque;
        }

}
```

=+

FIN de l'annexe E

=+