# Système Apprenant à Jouer au Go

**Tristan CAZENAVE**
**LAFORIA**
**Université Pierre et Marie Curie**
**4, place Jussieu**
**75252 PARIS CEDEX 05**
**e-mail : cazenave@laforia.ibp.fr**

## Résumé

Le but de cet article est de décrire un système qui apprend à effectuer la tâche éminemment complexe de jouer au Go. Le système apprend à atteindre des buts et à résoudre des problèmes spécifiques au Go, cependant les mécanismes d'apprentissage peuvent être utilisé pour apprendre à résoudre d'autres problèmes. Pour apprendre, il comprend ses erreurs puis se modifie afin de ne plus les commettre. Il peut apprendre en jouant contre lui-même, mais aussi en jouant contre un adversaire humain ou informatique, ou encore en analysant des parties jouées par d'autres.

## 1 Introduction

Les jeux de stratégie sont un bon terrain d'expérimentation pour L'Intelligence Artificielle. En effet, au Go comme aux Echecs, les experts sont nombreux, et entre experts et débutants existe une grande variété de niveaux bien répertoriés. L'intérêt de la diversité et de la fiabilité du classement est primordial lorsqu'on veut valider un système ou un modèle qui est sensé s'approcher d'un comportement cognitif humain. La programmation des jeux de stratégie permet de bien mesurer l'état d'avancement de l'Intelligence Artificielle.

La finalité de cet article est d'expliquer le fonctionnement d'un système modélisant l'apprentissage et le raisonnement d'un joueur de Go. Le modèle décrit ici a donné naissance à un système qui joue une partie entière de Go et qui s'améliore automatiquement partie après partie grâce à ses capacités d'apprentissage.

Dans cet article, je vais tout d'abord retracer l'évolution du système à travers ses divers modes d'apprentissage, puis je détaillerai le mécanisme de décision. Enfin, j'exposerai les performances de ce système contre d'autres systèmes jouant au Go.

## 2 Pourquoi travailler sur un programme de Go ?

Développer un programme qui puisse se classer parmi les meilleurs programmes actuels demande beaucoup d'efforts. Alors, pourquoi se lancer dans une telle entreprise ?

Une grande partie des connaissances que nous avons en Intelligence Artificielle vient des programmes de jeu. Les jeux de stratégie sont des micro-mondes qui ont juste la bonne complexité et qui peuvent bien être formalisés. Ils permettent de tester et d'améliorer les différentes techniques de l'Intelligence Artificielle et ce, de bien meilleure façon que d'autres disciplines moins biens définies.

Les progrès effectués par un programme peuvent être mesurés aisément grâce aux confrontations de celui-ci avec des joueurs humains dont le niveau d'expertise est très précisément quantifié du fait de la compétition permanente qui règne entre eux, et grâce à la possibilité de rencontrer d'autres programmes afin de déterminer lequel est le plus expert. Des programmes utilisant des méthodes ou des concepts différents peuvent ainsi être départagés.

Par ailleurs, les programmes de Go actuels ne dépassent pas le niveau de débutant de club, et ceci bien que certains programmes aient demandé plus de 10 années homme de travail. Faire un bon programme de Go dans le temps d'une thèse demande donc d'être à la fois innovant et

performant. On se doit de développer de nouvelles techniques, efficaces et rapides à mettre en oeuvre. Ces techniques de programmation des jeux pourront ensuite être utilisées pour le développement de programmes dans d'autres domaines de l'Intelligence Artificielle.

Peut-être peut-on rappeler que le calcul des probabilités fut imaginé par Pascal pour aider le Chevalier de Méré à mieux jouer aux dés, et qu'Euler inventa la théorie des graphes pour résoudre le problème jouet des ponts de Koenisberg. Ainsi, une découverte dans ce qui est actuellement le problème ouvert de la programmation du Go apportera certainement beaucoup à d'autres disciplines.

Les domaines de l'informatique concernés par la programmation du Go vont de la théorie des jeux à la représentation des connaissances en passant par la recherche heuristique, la résolution de problèmes, l'apprentissage, la métaconnaissance, les systèmes multi-agents, la logique floue, les réseaux de neurones, les algorithmes génétiques, l'optimisation de programmes et l'incrémentalisme. Cette programmation touche aussi aux fondements de l'informatique en utilisant la théorie mathématique des jeux créée par John Von Neumann ou les techniques de recherche en arbre formalisées par Alan Turing et Claude Shannon.

Longtemps les échecs ont été considérés comme le test parfait pour l'Intelligence Artificielle. Cependant, les bons programmes de jeu d'échec sont ceux qui calculent le plus vite en utilisant une fonction d'évaluation basique et un mécanisme de quiescence très simple [Hsu 1990]. Ils n'utilisent que très peu de connaissances et ils font plus de la recherche opérationnelle que de l'Intelligence Artificielle. Au Go, une telle approche est impossible car on ne peut pas bien approximer l'évaluation d'une position avec une fonction simple et rapidement calculable, et de plus le nombre moyen de coups possibles avoisine 200 alors qu'aux échecs il est plus proche de 36. Ainsi, le calcul Alpha-Béta étant exponentiel, même si on trouvait une fonction d'évaluation peu coûteuse, on ne pourrait pas, actuellement, calculer assez de coups à l'avance. La programmation du jeu de Go est d'ailleurs considérée aujourd'hui comme l'un des plus grands défis lancé aux informaticiens [Bradley 1979].

## 3 Le Go et l'apprentissage

Le Go est une bonne discipline pour faire des expériences sur l'apprentissage, et ceci pour au moins quatre raisons.

Premièrement, les expériences peuvent être faites sur de petits damiers, et, si elles sont concluantes, on peut agrandir le damier à volonté. On a ainsi accès à un grand éventail de difficultés [Pell 1991].

Deuxièmement, les méthodes traditionnelles de recherche en arbre [Lee 1988] sont inefficaces au Go. Pourtant, tous les joueurs de Go lisent plusieurs coups à l'avance pour les situations les plus simples (par exemple les prises et les connections), et peuvent lire jusqu'à 40 coups à l'avance pour des séquences que même les débutants connaissent (les Shishos) [Kierulf 1990]. Les programmes qui apprennent quels coups considérer pourraient améliorer les performances des programmes de Go.

Troisièmement, les bons programmes de Go utilisent des bases de règles indiquant ce qu'il est possible de faire, mais, la façon de les utiliser est programmée.

Mes recherches sur un programme apprenant à jouer au Go ont pour but de créer automatiquement ces bases, mais aussi de créer automatiquement des règles concernant l'utilisation des règles, et des règles concernant l'apprentissage. Je touche là au domaine de la métaconnaissance [Pitrat 1990]. De plus, les mécanismes d'apprentissage que j'étudie ne sont pas spécifiques au Go et ils pourraient être appliqués à d'autres jeux. J'ai choisi le Go parce que c'est un jeu assez complexe pour résister aux méthodes combinatoires sans intelligence. La nécessité d'utiliser des techniques d'Intelligence Artificielle pour modéliser les comportements humains en général est beaucoup plus visible lorsqu'on essaie de programmer le Go que lorsqu'on essaie de programmer les échecs (pour ceux-ci les meilleurs programmes sont combinatoires).

Des recherches très intéressantes ont été effectuées aux échecs sur l'apprentissage [Pitrat 1974] mais les programmes qui apprennent n'arrivent pas au niveau des programmes combinatoires. La meilleure manière connue de programmer le Go est d'essayer de mimer au mieux le comportement humain. Je m'attache tout particulièrement à modéliser l'apprentissage car cela me parait la méthode la plus efficace et la plus rapide pour

arriver au degré de connaissance nécessaire à un programme de bon niveau.

## 4 Représentation des connaissances :

### 1. Les jeux :

Pour représenter la connaissance du joueur de go, je me base sur la théorie des jeux de Conway [Berlekamp 1982].

Un jeu est associé à un but à atteindre. Il peut prendre différents états, selon

- que le joueur peut atteindre le but quoiqu'il arrive (état >),

- que le joueur peut atteindre le but s'il joue en premier mais que son adversaire peut l'empêcher d'atteindre le but si c'est à lui de jouer en premier (état *),

- que le but ne peut pas être atteint même si le joueur joue en premier (état <).

### 2. Les buts utilisés :

- Le but **faire un oeil**, c'est à dire atteindre la position ci-dessous sur une partie du damier ou goban :

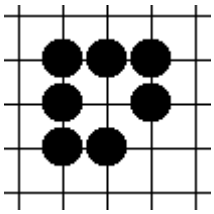

Figure 1 : Oeil noir

La possibilité d'atteindre ou non ce but donne naissance au jeu de l'oeil.

- Le but **connecter 2 pierres** :

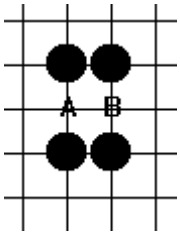


Figure 2 : Noeud de bambou

Deux pierres sont connectées si elles sont reliées entre elles par une chaîne de pierres de la même couleur ( les pierres voisines sont les pierres horizontalement et verticalement voisines et non pas diagonalement voisines). Dans l'exemple ci-dessus le jeu de la connexion est > car si Blanc joue en A, Noir joue en B et vice et versa. Un proverbe dit que le noeud de bambou est incassable.

- le but **prendre un bloc de pierres** ; un bloc de pierres est retiré du jeu s'il n'a plus de libertés, les libertés d'un bloc de pierres étant les intersections vides voisines de ce bloc.

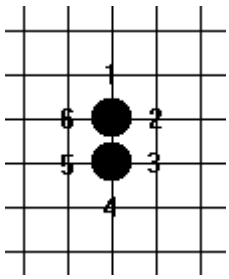


Figure 3 : Le bloc ci-dessus a six libertés numérotées de 1 à 6.

Pour retirer le bloc du jeu, Blanc devra poser des pierres sur les six libertés.

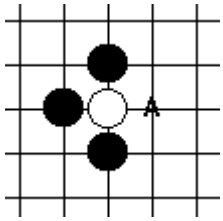


Figure 4 : Le jeu de la prise est * pour la pierre blanche.

Dans l'exemple ci-dessus, si Blanc joue en A, il sauve sa pierre. Si c'est Noir qui joue en A, il fait prisonnier la pierre blanche qui est alors retirée du damier. L'état du jeu de la prise pour la pierre blanche est donc *.

Pour chacun de ces trois buts existent aussi les buts inverses qui sont **empêcher un oeil**, **déconnecter**, **sauver une chaîne**. Enfin existent également des buts correspondant aux menaces amies ou ennemies d'atteindre un but.

Lorsqu'un jeu est * ou lorsqu'on a une règle de menace, on retient les coups qui permettent de faire changer l'état du jeu.

### 3. Les règles :

Elles ont pour prémisse une configuration particulière d'une partie du goban que j'appellerai par la suite **pattern**. A chaque pierre de ce pattern peut être associée une information additionnelle sur le nombre minimum et maximum de libertés que peut prendre le bloc dont fait partie la pierre.

Elles ont en conclusion un but, l'état du jeu correspondant à ce but, ainsi que si besoin est, les coups qui permettent de changer l'état du jeu.

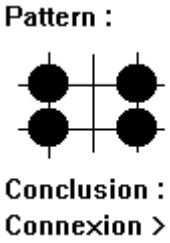


Figure 5 : Règle sur la connexion, le jeu de la connexion est > dans ce cas

## 4 Apprentissage par énumération

Sachant que les buts décrits précédemment sont des buts importants à atteindre pour le joueur de Go [Bouzy 1993] et que la reconnaissance d'un jeu * permet de faire basculer l'état du jeu en sa faveur ou en la défaveur de l'autre, j'ai pensé à répertorier tous les patterns comportant des jeux * ou > (les jeux > ne conseillent pas de coups mais ils sont aussi utiles pour le mécanisme de décision que les jeux *).

J'ai donc créé un programme qui engendre tous les patterns envisageables dans un espace restreint (rectangles 4-3 ou 3-3). Sur chacun de ces patterns, le programme calcule l'état du jeu pour un but choisi parmi ceux exposés plus haut et retient la règle ainsi déduite si elle est intéressante, c'est à dire si le jeu est * ou > et éventuellement les coups associés.

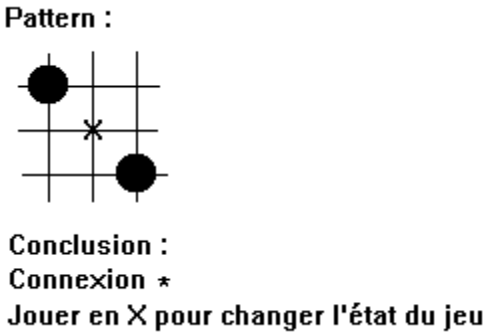


Figure 6 :Exemple de pattern apprise par le système

Dans cet exemple, si noir joue en X il connecte ses deux pierres alors que si c'est blanc qui joue en X les deux pierres noires ne seront plus connectables.

Il m'est apparu après avoir fait apprendre au programme toutes les règles ayant des patterns 4 sur 3 en prémisse, que ce nombre de règles était très grand alors qu'en général la fréquence d'apparition d'une règle (et donc son utilité) décroît avec sa taille. On remarque cependant qu'il en va différemment pour quelques règles que l'on utilise très souvent en début de partie et qui ont une grande taille.

Cette façon d'apprendre est donc bien adaptée à l'apprentissage de règles ayant en prémisse un pattern de petite taille. Mais elle est limitée dès que l'on veut obtenir des règles statiques de plus grande taille qui sont aussi utilisées par les experts humains du jeu de Go.

En outre, au jeu de Go, il existe des situations pour lesquelles un calcul limité dans une sous-partie du damier ne permet pas de connaître la solution d'un problème. Dans le cas du Shisho, qui est un calcul que même les débutants savent effectuer, on doit poser mentalement des pierres sur toute une diagonale du damier. Le calcul est nécessaire pour savoir si on peut prendre une pierre ou non, car il existe un très grand nombre de shishos différents et essayer de les répertorier serait vain. Mais, les coups à essayer lors de la construction mentale de l'arbre de calcul sont très simples et très faciles à repérer et la plupart du temps, l'arbre se réduit à un tronc (par exemple le Shisho décrit au chapitre suivant).

J'ai donc trouvé pertinent et judicieux d'introduire dans le système des règles concernant les coups à essayer dans un calcul ayant un but précis, autrement dit, des règles dynamiques pour pallier

les faiblesses des règles statiques, qui sont nécessaires mais pas suffisantes.

## 5 Apprentissage à partir de ses erreurs

Les règles utilisées pour faire les calculs sont appelées règles dynamiques, elles sont de la forme :

Pattern vérifié

**ET**

Certaines libertés > nombre de libertés minimum

**ET**

Certaines libertés < nombre de libertés maximum

**IMPLIQUE**

Coup à essayer pour atteindre le but

La principale différence d'utilisation entre les règles statiques apprises par énumération et les règles dynamiques est que les règles statiques donnent un coup à jouer de façon certaine, alors que les règles dynamiques conseillent un coup à essayer dans un calcul. Elles sont de ce fait plus puissantes.

Pour chacune de ces règles dynamiques, on tient à jour des statistiques concernant la fréquence avec laquelle le pattern a permis d'atteindre le but. Par la suite, lors de l'utilisation de ces règles, on essaie en priorité dans l'arbre celles qui ont le plus de chances d'atteindre le but. Ceci, grâce à l'élagage Alpha-Béta, permet des calculs plus rapides que si l'on avait essayé les coups proposés par les règles dans n'importe quel ordre. Ces statistiques sur l'utilisation des règles permettent donc au résolveur de problèmes d'être plus efficace.

L'apprentissage à partir de ses erreurs permet aussi au programme d'apprendre à résoudre de nouveaux problèmes. En effet, lorsqu'il joue une partie ou qu'il observe une partie jouée par d'autres, le programme a la possibilité de progresser par lui-même. Pour cela, il possède un mécanisme qui lui permet de quantifier la surprise engendrée par un coup. On peut aussi remarquer que ce mécanisme lui permet de se rendre compte de ses erreurs. Ainsi, à chaque coup joué dans une partie, il calcule l'état des jeux présents sur le damier. Il se souvient des jeux tels qu'ils étaient avant le coup, puis il regarde l'évolution de ces jeux. Si un jeu qu'il croyait > passe à un état <, cela signifie que le coup qui vient d'être joué n'avait pas été envisagé lors du calcul de l'état du jeu. Le programme apprend alors à l'envisager en créant une règle correspondant à la configuration du damier avant le coup surprenant. Il se rendra compte désormais du danger que peut représenter ce coup. Il pourra aussi l'utiliser contre ses futurs adversaires. Le programme agit de la même manière pour les jeux qui passent de < à >.

Une règle se présente sous la forme suivante pour le système :


Figure 7 : Règle pour prendre une pierre (règle numéro 1)

Cette règle s'applique très souvent. Elle très utile pour calculer ce que les joueurs de Go appellent les Shishos. Le cas suivant montre un exemple pour lequel cette règle n°1 s'applique :

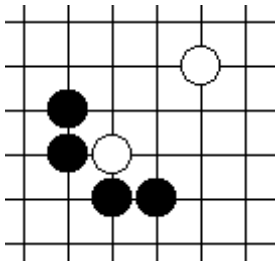

Figure 8

Une fois que le système a reconnu qu'un Shisho pouvait marcher, il le calcule pour savoir s'il peut effectivement prendre la pierre :

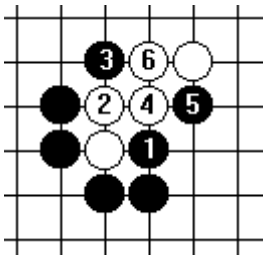


Figure 9 : Calcul de la prise en utilisant la règle numéro 1

On voit dans cet exemple que Noir ne peut pas prendre la pierre blanche en jouant les coups suggérés par la règle numéro 1 (qui a été appliquée 3 fois). Le calcul du Shisho s'arrête lorsque le bloc blanc a plus de 2 libertés (sinon il y a une explosion combinatoire).

Cependant, si Noir est joué par un joueur adverse, ou si en analysant une partie, le système tombe sur le coup numéro 1 de l'exemple suivant il se rend alors compte par lui même qu'avant le coup le jeu de la prise de la pierre blanche était <, alors qu'il est maintenant >. Le coup numéro 1 n'avait donc pas été envisagé alors qu'il permettait de prendre la pierre blanche. Le système va alors créer une nouvelle règle pour pallier cette défaillance et ne plus refaire l'erreur de calcul.

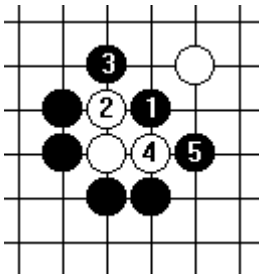


Figure 10 : Nouveau calcul après un coup surprenant

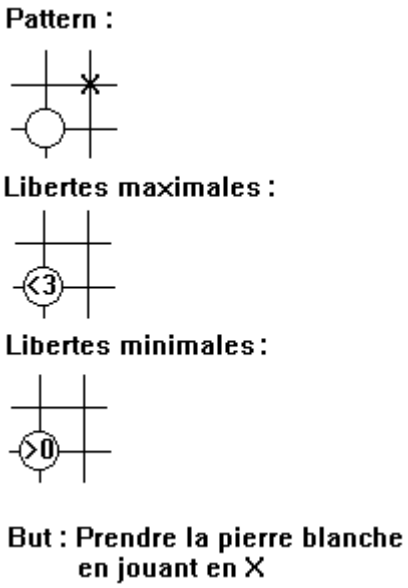


Figure 11 : Nouvelle règle créée pour ne plus refaire l'erreur de calcul

De cette façon, le système apprend automatiquement une nouvelle règle, et cela uniquement en jouant ou en observant une partie. Ce mécanisme a été implémenté, et à chacune de ses parties, le système apprend de nouvelles façons de prendre des blocs de pierres.

La prochaine étape du fonctionnement du programme est d'appliquer ce mécanisme aux autres buts utiles aux joueurs de Go. Ces buts sont décrits dans le paragraphe suivant.

## 6 Le mécanisme de décision

Pour des raisons de rapidité, au vu du grand nombre de règles statiques pouvant être reconnues sur le damier, on utilise une fonction de hachage des patterns qui permet de diviser le temps d'analyse du damier par un facteur 100.

Après cette analyse, le système recherche, à l'aide des règles dynamiques, les calculs devant être effectués par le résolveur de problèmes. Celui-ci effectue alors ces calculs et les résultats sont utilisés par le mécanisme de décision.

Ces résultats consistent en un ensemble d'informations sur les possibilités d'atteindre différents buts qui sont :

- Connecter deux pierres amies ensembles,

- Déconnecter deux pierres ennemies,

- Sauver un bloc de pierres ami,

- Prendre un bloc de pierres ennemi,

- Faire fuir un bloc de pierres ami,

- Encercler un bloc de pierres ennemi,

- Protéger une intersection amie,

- Attaquer une intersection ennemie,

- Faire un oeil ami,

- Empêcher un oeil ennemi,

- Faire vivre un groupe ami,

- Tuer un groupe ennemi,

- Menacer d'atteindre un des buts précédents.

A partir de ces informations, le programme construit des groupes de pierres connectées. Il note ensuite chacun des coups pouvant atteindre un but en fonction de l'importance et de l'urgence de ce but. Pour cela, il utilise des informations telles que le nombre de pierres, l'influence et la vitalité de chaque groupe. Il utilise comme heuristique principale d'évaluation des coups le principe suivant : attaquer un groupe faible vaut deux fois son nombre de pierres plus deux fois son influence.

La détection des menaces lui permet de détecter les "fourchettes" entre buts. La note associée aux fourchettes est le minimum des notes de chacune des menaces.

## 7 Résultats

Mon programme est écrit en C++. Il comporte actuellement plus de 20 000 lignes et plus de vingt classes d'objets.

Il utilise des fichiers de règles statiques comportant plus de 3000 règles.

Les règles dynamiques dépassent la cinquantaine.

La très grande majorité des règles que le programme utilise ont été créées par lui grâce aux mécanismes d'apprentissage décrits dans les paragraphes précédents.

Chaque coup est joué en moins d'une minute sur une Sparcstation10.

J'ai fait jouer mon programme contre Gnugo, un programme de référence accessible dans le domaine public. Mon programme a remporté une victoire écrasante de 271 points. (Les parties entre joueurs de Go de même niveau se jouent en général, sauf catastrophe, à une dizaine de points d'écart.)

Par ailleurs, en six mois de parties régulières contre une version de référence figée de Parigo, le programme de B. Bouzy, les résultats de mon programme se sont améliorés de façon fulgurante ; il est passé de parties perdues de 290 points à des parties gagnées de 170 points (le résultat maximum théorique pour une partie de Go 'normale' est de 361 points). Mon programme a donc réalisé rapidement de très grand progrès aussi bien au niveau du mécanisme de décision qu'au niveau de la résolution de problèmes.

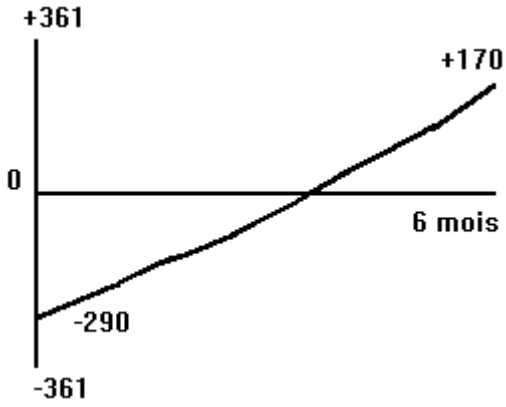


Figure 12 : Progression des résultats contre Parigo

## 8 Conclusion

Mon travail de recherche s'est nourri de l'émulation constante due aux confrontations régulières entre mon programme et celui de B. Bouzy, ainsi que de l'expérience accumulée par d'autres chercheurs dans les domaines de la programmation des jeux, de la résolution de problèmes et de l'apprentissage.

Actuellement, le mécanisme d'apprentissage à partir des erreurs du programme ne fonctionne que pour la prise de blocs de pierres ; lorsque ce mécanisme sera totalement implémenté, les performances du programme seront considérablement améliorées, et la meilleure façon de les améliorer encore sera de lui faire jouer des parties.

Il est très encourageant de voir le programme s'améliorer régulièrement et rapidement. Les techniques d'apprentissage et la séparation du programme en trois modules distincts à savoir le mécanisme de décision, le résolveur de problèmes et le mécanisme d'apprentissage sont sans doute pour beaucoup dans ces résultats.

Pour finir, mon programme baptisé Gogol comprend ses erreurs et utilise cette compréhension pour s'améliorer en apprenant de nouvelles règles. Presque toutes les règles qu'il utilise ont été apprises par lui même, mon travail n'ayant consisté qu'à écrire les fonctions d'apprentissage et d'utilisation des règles et à formaliser les buts à atteindre.

## 9 Remerciements

Je tiens tout particulièrement à remercier Jacques Pitrat, mon directeur de thèse. Je remercie également Bruno Bouzy pour l'émulation amicale qu'il a su créer, Jean-Marc Nigro, Michel Cazenave et Valérie Sion pour leur relecture et leurs conseils ainsi que toute l'équipe Métaconnaissance du LAFORIA.

## Bibliographie

[Berlekamp 1982] E. Berlekamp, J.H. Conway, R.K. Guy. Winning Ways. Academic Press, Londres 1982.

[Bradley 1979] Bradley, M.B. The Game of Go - The Ultimate Programming Challenge ? Creative Computing 5, 3 (Mars 1979), 89-99.

[Bouzy 1993] Bruno Bouzy. Modélisation des groupes au jeu de Go. Laforia, 1993.

[Fotland 1993] David Fotland. Knowledge Representation in The Many Faces of Go. Second Cannes/Sophia-Antipolis Go Research Day, Février 1993.

[Hsu 1990] Feng-Hsiung Hsu, Thomas Anantharaman, Murray Campbell, Andreas Nowatzyk. Un ordinateur parmi les grands maîtres d'échecs. Pour la Science n° 156, Octobre 1990.

[Kierulf 1990] Ander Kierulf, Ken Chen, and Jurg Nievergelt. Smart Game Board and Go explorer : A case study in software and knowledge engineering. *Communications of the ACM*, 33(2), Février 1990.

[Lee 1988] Kai-Fu Lee and Sanjoy Mahajan. A pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36:1-25, 1988.

[Pell 1991] Barney Pell. Exploratory Learning in the Game of GO. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad*. Ellis Horwood, 1991.

[Pitrat 1974] Jacques Pitrat. Realization of a program learning to find combination in chess. Computer orientated learning processes. NATO Advanced Study Institute.

[Pitrat 1990] Jacques Pitrat. Métaconnaissance futur de l'intelligence artificielle. Hermès, 1990.