Utilisation d'UCT au Hex

Abdallah Saffidine

2 Juin 2008 - 11 Juillet 2008

Merci à Tristan Cazenave pour son aide, son soutien, ses conseils.

Résumé

Le Hex est un jeu de plateau, le principal représentant des jeux de connexion. L'application de l'algorithme *Upper Confidence bound for Trees* (UCT) au Go a constitué une révolution dans le domaine, en effet les Intelligences Artificielles utilisant UCT ont rapidement rivalisé avec les meilleures Intelligences Artificielles (IA) produites jusqu'alors. Nous présentons ici le Hex, l'algorithme UCT, et surtout son application au Hex. L'IA qui en résulte, nommée *Yopt*, a été programmée en C++. Il est aisé de paralléliser ce programme pour l'utiliser sur une grappe d'ordinateur. *Yopt* participera aux olympiades des ordinateurs qui auront lieu à Pékin fin septembre 2008.

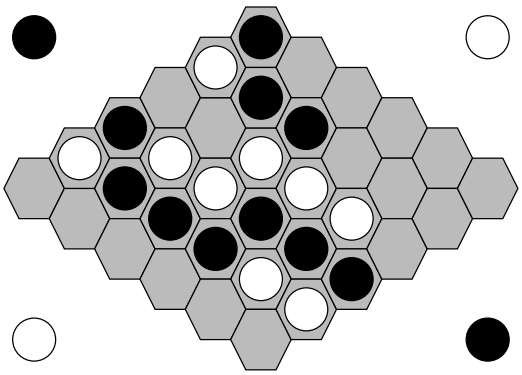

FIGURE 1 – Partie de Hex gagnée par Noir

1 Le jeu de Hex

1.1 Historique

Inventé indépendemment par Piet Hein en 1942 et John Nash en 1948, le jeu de Hex est le principal représentant de la classe des jeux de connexion. Le Hex a su attirer à lui une importante communauté de joueurs. Un championnat du monde a eu lieu à plusieurs reprises et le premier livre complètement dédié à la stratégie au Hex est paru en 2000 [4].

Il existe plusieurs généralisations du Hex (voir en particulier A.1.1), mais aucune ne parvient à susciter le même enthousiasme que le jeu original.

1.2 Règles

Le Hex se joue à deux sur un plateau ayant la forme d'un losange, le plateau est pavé par des hexagones. À chaque joueurs est attribué une pair de côtés opposés. Chacun pose à son tour un pion sur une case du plateau. Les pions ne sont jamais déplacés ni ôtés. Le premier joueur réussissant à relier ses bords par une chaîne de pions de sa couleur remporte la partie (voir figure 1).

Le fait de jouer au centre donne un avantage important au premier joueur, pour équilibrer le jeu une règle additionnelle dite du gâteau est ajoutée. Au moment de jouer le deuxième coup, le joueur B peux choisir d'inverser les couleurs auquel cas le joueur A joue en second.

1.3 Propriétés remarquables

Il a pu être démontré de nombreuses propriétés sur le Hex. La plupart de ces théorèmes peuvent intéresser les joueurs occasionnels ; notamment l'absence de partie nulle au Hex (voir A.1.2). De cela il découle l'existence d'une stratégie gagnante en début de partie pour le premier joueur (voir A.2).

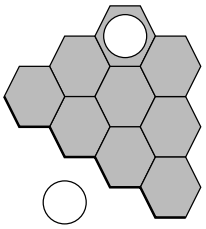

FIGURE 2 – La forme *Ziggurate* assure le lien entre le pion Blanc et le bord

Il est peut-être moins profitable au joueur non mathématicien d'apprendre que l'absence de partie nulle au Hex équivaut au théorème du point fixe de Brouwer en dimension deux [7]. Un informaticien pourra percevoir la difficulté pour programmer une IA performante à travers la complexité d'une généralisation du Hex. La généralisation du Hex avec des plateaux de taille arbitrairement grande est PSPACE-complète [11].

1.4 Éléments de stratégie

1.4.1 Formes

Dès que l'on repère un losange sur le plateau on relie mentalement les groupes qu'il joint. Mémoriser cette forme permet d'éviter le calcul certes rapide de la connexion entre les deux groupes. Il est possible de généraliser la notion de forme (en Anglais *template*), certaines assurent par exemple la liaison entre un pion et le bord du plateau (voir figure 2).

La méthode la plus simple pour prouver qu'une forme (par exemple figure 3) relie effectivement ses groupes est d'expliciter les menaces de connexions (figure 4), chaque menace de connexion possède un support. Le support est l'ensemble des cases vides dont l'occupation affecte la réalisation effective de la connexion. Ainsi pour empêcher la connexion de se faire, l'adversaire doit jouer dans le support de la menace afin de la parer. Si les menaces sont nombreuses, il lui faut alors jouer dans l'intersection des supports. Si l'intersection est vide alors la connexion est prouvée, sinon il est possible d'étudier de manière plus exhaustive les quelques cas restants (voir figure 5).

1.4.2 Échelles

Les échelles, connues au Go également, imposent au joueur de Hex de considérer l'ensemble du plateau lors de son évaluation. Une échelle est la construction par l'adversaire d'un mur parallèle au bord vers lequel le groupe tend. La seule manière d'éviter d'être complètement bloqué consiste à disposer des sorties d'échelle (*ladder escape*) sur le chemin de l'échelle. Ces pions permettent alors de gagner l'initiative et de contourner le mur adverse (voir figure 6).

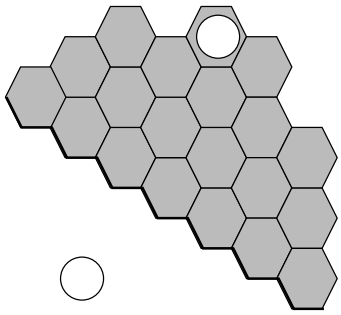

FIGURE 3 – La forme à distance IV

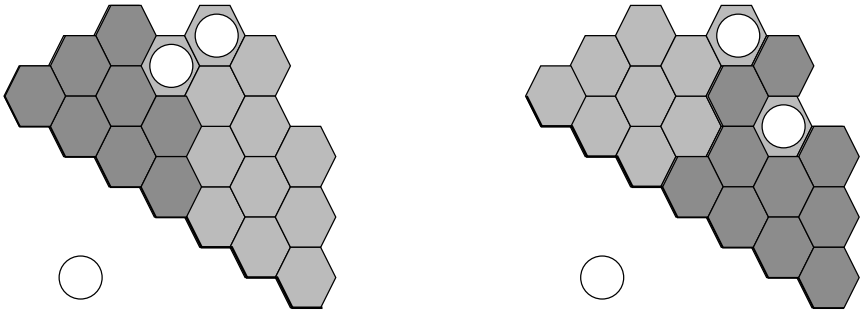

FIGURE 4 – Deux menaces blanches et leur support

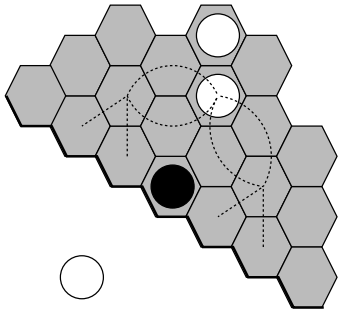

FIGURE 5 – Analyse d'une réponse de Noir dans la case restante

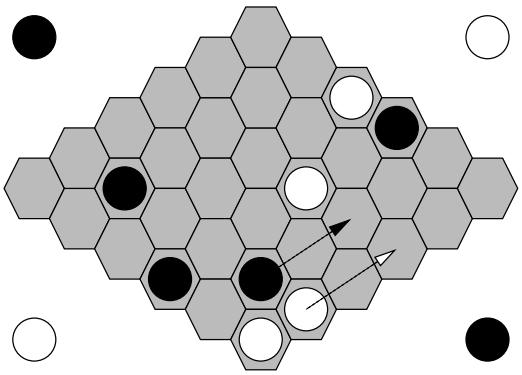

FIGURE 6 – Échelle imposée par Blanc pour laquelle Noir dispose d'une sortie. Noir au trait.

1.5 Les joueurs artificiels de Hex

Depuis sa création, le Hex a suscité le désir de réaliser des joueurs artificiels. Ainsi en 1953 Shannon construit la première machine capable de jouer. D'autres types d'IA sont possibles. Cependant aujourd'hui, la meilleur IA, *Six*, n'a pas le niveau d'un bon joueur de Hex. Le Hex partage en effet de nombreuses caractéristiques avec le Go qui rendent difficile la construction d'une IA performante.

D'une part le facteur de branchement est élevé : le nombre typique de coups légaux dépasse 100 (Échecs 40, Go 350), ceci empêche une exploration profonde de l'arbre des coups par min-max. D'autre part,le fait que deux situations proches puissent avoir une valeur différente qui empêche la plupart des méthodes d'apprentissage d'être pertinentes. Finalement la stratégie au Hex fait beaucoup appel à l'intuition et est moins formalisée que celle des échecs, il devient donc ardu de créer une fonction d'évaluation adéquate.

1.5.1 Machine de Shannon

L'automate de Claude Shannon assimile le plateau à un circuit électrique. Une case vide est une résistance unité, une case occupée par l'adversaire est un interrupteur ouvert tandis qu'une case occupée par soi est une résistance nulle. Il s'agit ensuite de jouer le coup qui minimise la différence de potentiels entre ses bords.

1.5.2 Hexy, Six

En 2000 Vadim V. Anshelevich [2] formalisait le concept des connexions virtuelles dont les losanges sont des exemples. Il proposait aussi une méthode pour les détecter par récurrence. Cela permettait de réduire le nombre de coups à

considérer. Il utilisait finalement la fonction d'évaluation de Shannon, en passant par une résolution de systèmes linéaires pour le calcul de la différence de potentiels. Le programme de Anshelevich, *Hexy*, fut champion aux Olympiades en 2000.

Gábor Melis a utilisé les résultats d'Anshelevich, pour construire *Six*, ce programme libre et disponible sur le site de Gábor Melis est actuellement le meilleur. Il fut médaille d'or des Olympiades en 2003 2004 et 2006.

1.5.3 Yopt

En 2006 l'application de l'algorithme UCT au jeu de Go a conduit à un bouleversement dans le domaine. Le plus clair exemple est l'IA *MoGo* qui a atteint en quelques mois le niveau des meilleurs programmes [9].

Il était donc naturel de se demander quels résultats pouvaient être obtenus de l'adaptation d'UCT au Hex. L'IA réalisée, nommée *Yopt*, utilise l'algorithme UCT pour créer l'arbre des coups, et effectue des simulations Monte-Carlo aux feuilles. Ces simulations, toutefois, prennent en compte certaines spécificités du Hex.

2 L'algorithme UCT

2.1 Upper Confidence Bound

L'application des méthodes Monte-Carlo aux jeux de réflexion fait appel à de nombreuses simulations aléatoires. L'exploitation standard de ces simulations se fait maintenant par l'algorithme UCT [10], plus adapté que le min-max. Cet algorithme est apparue dans le cadre des problèmes de bandit-manchots où la seule manière d'obtenir des informations sur un état consiste à faire des tirages aléatoires.

Ces problèmes inspirés des machines à sous du casino proposent différentes manettes à actionner, chacune offrant une rétribution aléatoire selon une loi qui lui est propre. Il s'agit de trouver un équilibre entre exploiter les bras pour lesquels la loi de distribution des rétributions semble généreuse ; et explorer les bras qui ne l'ont pas encore été suffisamment pour obtenir des informations plus fiables sur leur espérance. Mieux explorer certain bras peut permettre de confirmer que le fait que leur distribution n'est pas favorable au joueur.

Kocsis et Szepesvari proposent dans [10] d'adapter l'algorithme *Upper Confidence Bound 1* aux problèmes dans lesquels apparaissent des arbres (voir algorithmes 1, 2, 3) ; les problèmes de recherche de chemin ou les jeux de plateau en sont des exemples concrets. Dans le cas des jeux de plateau, on considère chacune des positions possibles comme un bras, les différentes évaluations effectuées depuis la position sont les rétributions de la manette. Toutefois il est possible de prendre en compte la structure arborescente de ces jeux : faire profiter les positions mères des simulations effectuées sur leurs filles. On aboutit à l'algorithme *Upper Confidence bound for Trees*(UCT)

Il est montré dans [10] que la probabilité de choisir le meilleur coup convergeait vers 1 lorsque le nombre de simulation tendait vers l'infini. La formule qui donne la valeur UCT d'un noeud, en fonction de la moyenne μ des scores obtenus et du nombre s de simulations qui sont passées par ce noeud et du nombre n de simulations effectuées dans tout l'arbre est :

$$\mu + C\sqrt{\frac{\log n}{s}}$$

avec C la *Constante UCT* choisie par l'utilisateur (voir tableau 2).

Algorithm 1 Pseudo-code pour UCT

recevoir une position p
Soit A un arbre UCT vide à la racine près
pour Le nombre de descentes dans l'arbre choisi **faire**
  Soit $p\prime$ une copie de p
  Soit N le noeud résultat d'une descente dans l'arbre (A , $p\prime$)
  Soit R l'évaluation de $p\prime$
  On effectue une remontée de l'arbre (A , N , R)
fin de pour
renvoyer le coup fils de la racine de A qui a la meilleure valeur UCT

Algorithm 2 Descente dans l'arbre

recevoir un arbre UCT,une position p
Soit N la racine de l'arbre
tant que Tous les fils de N ont déjà été exploré au moins une fois **faire**
  Soit F le fils de N ayant la plus grande valeur UCT
  On joue sur p le coup qui mène de N à F
  $N \leftarrow F$
fin de tant que
Soit F un fils de N tiré au hasard parmi les fils non exploré
F devient exploré
On joue sur p le coup qui mène de N à F
$N \leftarrow F$ {N est maintenant une feuille}
renvoyer le noeud N

2.2 Monte-Carlo

Pour profiter au mieux de l'algorithme UCT, les évaluations des feuilles doivent être le plus rapide possible. En effet, dans la mesure où les rétributions des filles sont des rétributions pour la mère, il est profitable d'explorer l'arbre le plus en profondeur possible, le noeud racine recevant dans tous les cas la même quantité de rétribution.

Algorithm 3 Remontée dans l'arbre

recevoir un arbre UCT, un noeud N, le résultat R d'une évaluation de N
{On suppose que pour l'évaluation d'une position équilibrée est 0}
tant que N n'est pas la racine de l'arbre **faire**
  On incrémente le compteur de parties de N
  si R a été réalisé alors que c'était le tour du joueur dont c'est le tour en N **alors**
    On incrémente la valeur cumulée de N de R
  sinon
    On incrémente la valeur cumulée de N de $-R$
  fin de si
  $N \leftarrow$ le père de N
fin de tant que

Pour obtenir des évaluations rapides, le plus simple reste l'utilisation de simulations Monte-Carlo (MC) (voir algorithme 4). Elles consistent à simuler une continuation possible de la partie et renvoyer le résultat comme valeur de la position. La justification tient au fait qu'une position presque gagnée donnera lieu plus souvent à une victoire aléatoire qu'une position défavorable.

2.2.1 Simulations aux noeuds

La fiabilité de l'évaluation de la position augmente avec le nombre de simulations. Au Hex, le temps de calcul est principalement consacré aux simulations, et les descentes dans l'arbre UCT n'en prennent qu'une part relativement faible. Il est donc aussi rapide d'effectuer n descentes suivies de k lancés MC que d'effectuer $n \times k$ descentes suivies d'un seul lancé MC.

Toutefois dans le cadre d'une exploitation du programme sur une grappe de machines parallèles il peut être utile de simuler plusieurs parties aux noeuds (voir 3.3.2).

2.2.2 Biaiser les simulations

Une autre manière d'augmenter la fiabilité de l'évaluation consiste à diminuer la part du hasard dans les simulations. Une partie jouée entre de bons joueurs reflète mieux la valeur d'une position qu'une partie jouée de manière complètement aléatoire. Il peut donc être profitable d'introduire des connaissance humaine dans les simulations. Ces connaissances peuvent prendre la forme de suites de coups forcées, de captures dès que possible, ou tout autre schéma qui permettre à la simulation de garder une certaine rapidité (algorithme 4).

Il est important cependant de veiller cela dit à introduire des directives qui soient valables dans le plus grand nombre de cas possible. Il a été montré en effet [6] qu'il était possible d'avoir un joueur aléatoire A meilleur qu'un joueur aléatoire B grâce aux connaissances dont il disposait, mais que le joueur UCT issu de A soit plus faible que le joueur UCT issu de B (voir 3.1.2). Une explication

serait que certains aspects ou pièges ne soient pas repérés par les connaissances introduites, mais le soient parfois par la variété issu des tirages aléatoires.

Algorithm 4 Lancé MC biaisé

```
recevoir une position
tant que la partie n'est pas finie faire
  si il y a un coup urgent alors
    jouer ce coup urgent
  sinon
    jouer un coup légal aléatoire
  fin de si
  changer le joueur courant
fin de tant que
renvoyer le gagnant
```

2.3 RAVE

Dans certains jeux les transpositions sont fréquemment possible, et ont souvent lieu. Au Hex par exemple, le fait de jouer un coup n'affecte l'ensemble des coups légaux que par le retranchement du coup joué. De plus dans une partie, inverser deux séquences d'attaque défense résulte souvent en une continuation de partie équivalente à celle qui se produit sans l'inversion.

L'algorithme *All Moves As First* (AMAF) était utilisé avant l'apparition d'UCT, il consiste à rétribuer non plus seulement le coup qui donne lieu à la partie aléatoire, mais tous les coups qui ont été joué à un moment ou un autre au cours de la partie. Dans le cas de simulations MC non biaisées et sans descente d'arbre, l'utilisation d'AMAF au Hex revient presque à jouer un nombre de simulations multiplié par le nombre de coup légaux.

L'algorithme *Rapid Action Value Estimation* (RAVE) [8] permet d'adapter AMAF à l'architecture UCT. Même dans le cas de simulations biaisées, l'utilisation de RAVE augmente sensiblement le niveau du joueur artificiel (voir tableau 3).

3 Yopt

Yopt est le nom de l'IA réalisé au cours du stage en vue de tester l'algorithme UCT pour le jeu du Hex. Elle intègre un biais dans les simulations MC et utilise l'algorithme RAVE. Les divers paramètres pour UCT et RAVE ont été testé suivant un protocole strict (voir B.1) afin de comprendre et d'interpréter leur influence sur la qualité de jeu produite.

TABLE 1 – Variation de la prise en compte des formes

- type 1 : sans forme
- type 2 : avec les losanges seulement
- type 3 : avec les losanges et les formes de bord distance II (type de l'IA standard)
- type 4 : avec les losanges, les formes de bord distance II et les ziggurates.

Formes prises en compte	type 1	type 2	type 4
Pourcentage de gain	22%	42%	71.5%

3.1 Modifications de MC

Les différentes modifications que l'on peut apporter aux simulations MC sont la condition d'arrêt de la simulation (voir 3.2.3), les séquences de coups forcés, la restriction aux cases pertinentes. Cette dernière modification n'a pas été implémentée.

3.1.1 Templates

Lors des simulations MC, l'apparition de certaines formes et prise en compte. Dès que l'adversaire s'introduit dans le support d'une de ces formes, une réponse automatique permet d'assurer que la connexion permise par la forme est préservée. Ainsi dès que l'un des joueurs relie ses bords par des losanges, la nature semi-aléatoire de la simulation permet de conserver cette victoire et reflète mieux la valeur de la position.

Il existe un nombre arbitrairement grand de formes qui peuvent être reconnues, mais si l'on tente de reconnaître trop de formes, le temps de recherche de ces formes devient une contrainte. Une série de tests (tableau 1) a été effectué pour mesurer l'utilité de la reconnaissance de formes et des réponses automatiques associées. Il apparaît que la prise en compte des formes les plus basiques augmente de manière considérable le niveau de l'IA. Ceci est tout à fait cohérent avec le fait que les joueurs humains attachent une grande importance aux formes et à leur reconnaissance [1].

3.1.2 Connexions

Il est aussi possible de forcer la connexion et/ou la déconnexion entre les groupes indépendants : s'il existe un coup permettant de relier deux chaînes d'un joueur i distinctes, alors la simulation le joue, si le tour était celui de i, les chaînes sont connectées, sinon les chaînes ne sont plus virtuellement semi-connectées [2]. Il faudrait d'abord repérer si les chaînes sont déjà virtuellement connectées, auquel cas il est inutile de forcer leur connexion explicite.

Il n'a malheureusement pas été trouvé de méthode permettant ce test de manière rapide. Les tests effectués semblent montrer que forcer la connexion de tous les groupes sans distinguer s'ils sont déjà virtuellement connectés n'apporte pas d'augmentation du niveau de jeu.

3.1.3 Cases mortes

Il existe dans certaines positions des cases dites *mortes* (voir figure 3.1.3) [3]. L'issue de la partie ne dépend pas de l'occupation de ces cases. Il est donc possible de ne pas les évaluer dans la construction de l'arbre UCT, et d'empêcher les simulations aléatoires de jouer sur ces cases. Il est possible d'étendre ce raisonnement aux cases *capturées* [3], une case capturée par un joueur J est une case telle que si l'adversaire de J joue cette case alors il y a un coup pour J qui fait de cette case une case morte (voir figure 8).

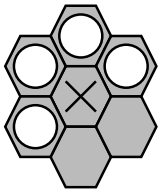

FIGURE 7 – La case marquée d'un X est *morte*

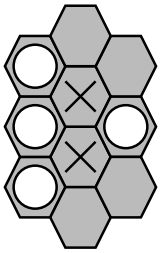

FIGURE 8 – Les cases marquées d'un X sont *capturées*

La détection des cases capturées et des cases mortes peut se faire efficacement en utilisant une bibliothèque de formes de petite taille et en les combinant pour en obtenir de nouvelles de taille plus grande. Cette détection n'a toutefois pas été implémentée.

3.2 Paramètres UCT

3.2.1 Constante UCT

Pour pouvoir choisir de quelle manière est effectuée la descente dans l'arbre UCT, on associe à chaque noeud une valeur. Cette valeur traduit l'urgence qu'il y a à jouer ce coup lors des prochaines descentes. Elle est facteur de la moyenne des résultats obtenus avec ce coup (c'est à dire, s'il a l'air d'être un bon coup), et de la proportion de descentes qui sont passée par ce coup (c'est à dire, s'il

Table 2 – Variation de la constante UCT

Constante UCT	−0.1	0	0.1	0.2	0.4	0.5	0.6	0.7
Pourcentage de gain	38.5%	61%	60%	55.5%	42%	41%	35.5%	32.5%

Table 3 – Variation de la constante RAVE

Constante RAVE	0	8000	32000	64000	∞
Pourcentage de gain	20%	40.5%	53%	52.5%	34%

a été déjà exploré souvent). La *Constante UCT* permet d'ajuster le compromis entre exploration et exploitation : proche de 0 elle incite à une exploration plus en profondeur sur les quelques meilleurs coups. L'algorithme min-max avec l'utilisation de la méthode de profondeur grandissante (*iterative deepening*) est exactement l'algorithme UCT simple avec une constante infinie.

Il semble que la meilleur valeur sur taille 11 soit proche de 0.0 (voir tableau 2).

3.2.2 Constante RAVE

L'utilisation de RAVE permet de prendre en compte pour un noeud le résultat de descentes dans l'arbre qui ne passent pas par ce noeud. Il s'agit alors de faire une moyenne pondérée par la *Constante RAVE* de ces résultats et de la valeur UCT pour obtenir la valeur du noeud. Lorsque le nombre de descentes passant par le noeud est petit le poids des autres descentes est élevé et il devient de plus en plus petit lorsque le nombre de descentes passant par le noeud augmente.

Comme le montrent les résultats des tests (voir tableau 3) l'utilisation de RAVE (lorsque la constante est non nulle) augmente le niveau de l'IA considérablement ; mais elle doit rester une première approche et faire place à la valeur UCT pour les noeuds les plus exploités.

3.2.3 Condition de fin de partie

Il est possible d'arrêter la simulation MC à plusieurs moments. On peut attendre que le plateau soit rempli, et profiter en ce sens de l'absence de partie nulle A.1.2 qui garantit que l'évaluation de la partie ne changera pas par rapport à l'arrêt lorsque la partie est finie. Une troisième option est de cesser le jeu dès que l'un des joueurs est parvenu à relier ses bords par des formes (voir figure 9), en particulier des losanges. Dans ce cas non plus le résultat de la partie ne varie pas puisque les losanges sont automatiquement défendus (voir 3.1.1).

La distinction entre les trois possibilités se fait lors de la propagation du résultat par RAVE. Par exemple on peut croire que continuer à jouer une fois que la partie est finie risque de bruiter la propagation par RAVE en récompensant des coups qui n'ont pas eu d'influence sur la partie. Les résultats expérimentaux

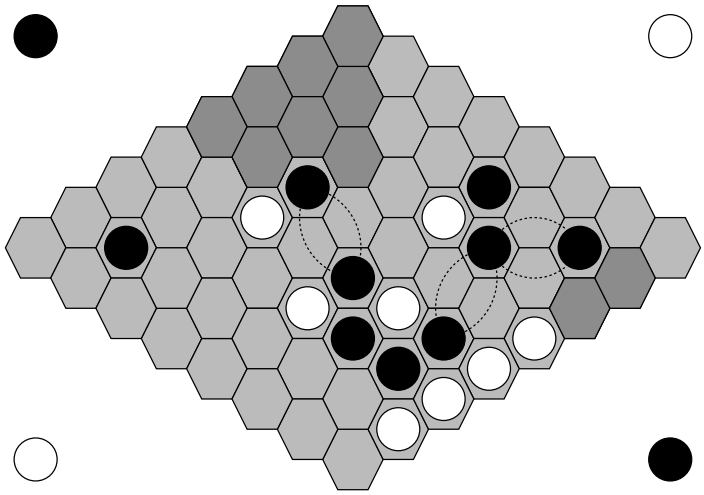

FIGURE 9 – Partie de Hex gagnée par Noir

ne confirment pas ce jugement. Ainsi le seul apport concret semble être le temps de réflexion qui diminue dans les positions presque gagnées.

3.3 Implémentation

3.3.1 Choix du langage

Yopt est la réalisation concrète des méthodes proposées ici. Elle a été écrite en langage C++ (4500 lignes) et en OCaml (3100 lignes). La version OCaml fut la première développée, elle offre la possibilité de jouer au Y. La version C++ dispose de quelques optimisations de vitesse et a servi aux tests.

La présence de deux versions s'explique premièrement par le fait que le langage C++ était inconnu au début du stage et qu'il était plus facile d'écrire les prototypes en OCaml. Deuxièmement la parallélisation d'une version C++ est réputée plus aisée, et l'apprentissage du C++ semble être très recommandé dans le domaine de la programmation des jeux. Troisièmement la volonté de pouvoir comparer les deux programmes, tant sur le plan du temps de réflexion nécessaire de part et d'autre, que sur le plan de la lisibilité et de la réutilisabilité des codes sources, a conduit à quelques mises à jour de la version OCaml. Toutefois ces comparaisons dépassent le cadre du stage. Elles se poursuivront et une conclusion leur sera apporté ultérieurement.

3.3.2 Parallélisation

La parallélisation du programme est possible par l'utilisation de processus légers, et surtout par l'utilisation de la bibliothèque MPI. Lorsque le nombre de processeur visé est compris entre 8 et 20, l'architecture retenue est du type

TABLE 4 – Variation du nombre de simulations aux feuilles

Nombre de simulations aux feuilles	2	4	8	16
Pourcentage de gain	56%	67.5%	77.5%	74.5%

maître-esclaves, seul le maître a connaissance de l'arbre UCT sous-jacent, il effectue les descentes et ordonne aux esclaves la réalisation des simulations MC.

Dans l'architecture maître-esclaves, les communications sont le facteur de ralentissement, on peut alors augmenter dans une certaine mesure le nombre de simulations aux feuilles à temps de calcul constant. Il en résulte une importante augmentation du niveau de jeu (voir tableau 4).

3.3.3 Olympiades

La validation de l'approche choisie est la participation de la version C++ de *Yopt* aux Olympiades de l'International Computer Game Association (ICGA) à Pékin du 28 Septembre au 5 Octobre. Dans cette compétition de nombreux jeux de plateaux abstraits sont représentés. La cadence est de 30 minutes par joueur par partie. Le moteur de jeu peut se trouver à distance, ceci permet l'utilisation de grappes.

Outre *Yopt* sont inscrits dans la section Hex cette année *Six*, *Wolve* et *MoHex*. *MoHex* devrait utiliser l'algorithme UCT avec probablement la reconnaissance des cases mortes (voir 3.1.3). *Wolve* ainsi que *Six* sont fondés sur les connexions virtuelles [2] et diffèrent par la reconnaissance des cases mortes, et les choix dans le compromis entre recherches des connexions virtuelles et exploration de l'arbre par min-max.

4 Conclusion

L'application de UCT au Hex semble prometteuse. En effet une configuration avec 500000 simulations par coups, soit une moyenne de 30 minutes par parties, permettait à *Yopt* de remporter plus de 40% des parties (près de 500) contre *Six*. Par ailleurs un stage de 6 semaines ne permet pas la construction d'un programme testant puis exploitant toutes les techniques qui semblent prometteuses. En particulier la reconnaissance des cases mortes, capturées et dominées pourrait être proposée dans une version ultérieure. L'utilisation des recherches λ [12] ou des menaces généralisées [5] devrait permettre de pallier l'une des principales lacunes de *Yopt* : la mauvaise gestion des échelles (ce problème se pose aussi pour les méthodes MC au Go). Finalement l'application des connections virtuelles, au lieu de schémas de reconnaissances prédéfinis, dans les simulations MC pourrait les biaiser de manière plus pertinente.

Sur le plan personnel, ce stage a certes permis le début de l'apprentissage du C++ et une progression en OCaml, mais surtout un approfondissement des connaissances en Intelligence Artificielle pour les jeux de réflexion. En outre être acteur du monde de la recherche est très enrichissant et complémentaire de la

formation reçue à l'ENS Lyon. La présentation du sujet du stage aux membres du LIASD Paris 8, la co-rédaction d'un article avec Tristan Cazenave, pour la *Revue d'Intelligence Artificielle* (Volume 1/2009) ainsi que l'inscription de *Yopt* aux Olympiades concluent le stage de la manière la plus valorisante qui soit.

Références

[1] Hexwiki. Website. http ://www.hexwiki.org.

[2] Vadim V. Anshelevich. A hierarchical approach to computer hex. *Artificial Intelligence*, 134(1-2) :101–120, 2002.

[3] Yngvi Björnsson, Ryan Hayward, Michael Johanson, and Jack van Rijswijck. Dead cell analysis in hex and the shannon game. In *Graph Theory in Paris : Proceedings of a Conference in Memory of Claude Berge (GT04 Paris)*, pages 45–60, 2007.

[4] Cameron Browne. *Hex Strategy : Making the Right Connections*. Natick, MA, 2000.

[5] Tristan Cazenave. A Generalized Threats Search Algorithm. In *Computers and Games 2002*, volume 2883 of *Lecture Notes in Computer Science*, pages 75–87, Edmonton, Canada, 2003. Springer.

[6] Louis Chatriot, Sylvain Gelly, Jean-Baptiste Hoock, Julien Perez, Arpad Rimmel, and Olivier Teytaud. Including expert knowledge in bandit-based monte-carlo planning, with application to computer-go. 2008.

[7] David Gale. The game of Hex and the Brouwer fixed-point theorem. *Amer. Math. Monthly*, 86(10) :818–827, 1979.

[8] Sylain Gelly and David Silver. Combining online and offline knowledge in uct. In *Proceedings of the 24th International Conference on Machine Learning*, pages 273–280. ACM Press, 2007.

[9] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of uct with patterns in monte-carlo go. Technical report, INRIA, November 2006.

[10] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. In *15th European Conference on Machine Learning*, pages 282–293, 2006.

[11] Stefan Reisch. Hex ist pspace-vollständig. In *Acta Informatica*, volume 15, pages 167–191. 1981.

[12] Thomas Thomsen. Lambda-search in game trees - with application to Go. In T. Anthony Marsland and I. Frank, editors, *Computers and Games*, volume 2063 of *Lecture NoDf.tes in Computer Science*, pages 19–38. Springer, 2002.

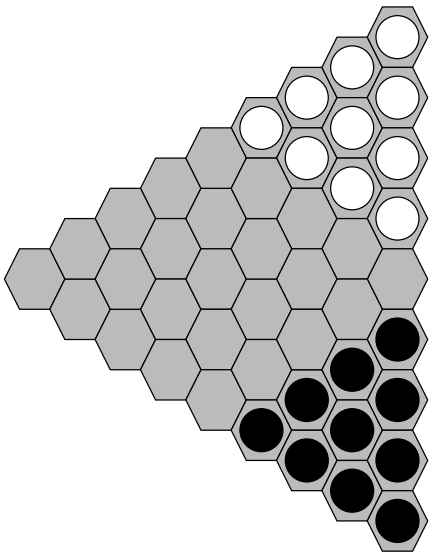

FIGURE 10 – Partie de Y prête à simuler une partie de Hex

A Démonstrations de quelques propriétés

A.1 À propos des parties nulles

A.1.1 Le Y

Le Y est une généralisation du Hex inventée en 1970 par Charles Titus et Craige Schensted. Les règles sont aussi simple qu'au Hex, on joue sur un plateau triangulaire à pavage hexagonale, le gagnant est le premier joueur parvenant à relier les trois bords par une même chaîne. Le Y est plus symétrique que le Hex : il n'y a pas de notion d'appartenance des bords. On y retrouve la plupart des notion de Hex : importance du centre, utilisation des formes... Un bon joueur de Hex est souvent un bon joueur de Y et vice et versa.

Toute partie de Hex peut être vue comme un cas particulier d'une partie de Y dans laquelle un certain nombre de coups préliminaires auraient été effectués (voir figure 10).

A.1.2 Absence de partie nulle

Il suffit de montrer que le Y n'admet pas de partie nulle pour avoir ce résultat au Hex. L'absence de partie nulle couvre en fait deux propriétés distinctes. Il n'y a jamais deux chaînes gagnantes sur un plateau de Y ; sur tout plateau complètement rempli il existe une chaîne gagnante.

La première propriété ne pose pas de difficulté. Supposons une position de Y avec deux chaînes gagnantes. Chaque bord est en contact avec les deux autres et avec les deux chaînes, qui sont elles-mêmes en contacts avec les trois bords et l'une avec l'autre (cette dernière affirmation peut être obtenue par un processus de gonflement qui ne change rien au problème). On vient donc d'obtenir le graphe complet à 5 sommets. Or ce graphe n'est pas planaire, ce qui montre la propriété par l'absurde.

Il n'est développé ici qu'une esquisse de la démonstration, pour la démonstration complète voir [1]. On défini le statut d'un plateau comme la valeur de vérité de la proposition "Il existe au moins une chaîne gagnante". Soit un plateau P rempli, on va montrer que le statut de P est le même que le statut d'un autre plateau pour lequel on peut connaître le statut de manière triviale. Il est possible de classer les groupes de pions suivant le nombre de côtés qu'ils touchent. Le plateau $P\prime$dans lequel tous les groupes ne touchant pas de bord sont remplacés par la couleur les encerclant a le même statut que P. De ce plateau $P\prime$, on peut remplacer tous les groupes ne touchant qu'un bord par la couleur les encerclant tout en conservant le statut. De même avec les groupes touchant deux bords. Le statut est toujours conservé, et dans chaque cas il reste toujours au moins un groupe (celui qui encerclait) sur le plateau. Dans le dernier plateau, tous les groupes touchent 3 bords, et il y a au moins un groupe. Ainsi sur le plateau P il y avait déjà une chaîne gagnante.

A.2 Stratégie Gagnante pour le premier

Le Hex est un jeu déterministe et à information parfaite et complète, il y a un nombre fini de parties possibles. Il n'y a pas de partie nulle. Il est donc possible de créer l'arbre de jeu entier issu de la position initiale. Les feuilles sont notés suivant l'issue de la partie. Il s'agit ensuite de noter par récurrence tous les noeuds jusqu'à arriver à la racine de l'arbre. Du point de vue du joueur A, un noeud obtenu après un coup de A est gagnant si et seulement si tous ses fils sont gagnant (c'est à dire que tous les coups de B l'emmènent vers la défaite). Un noeud obtenu après un coup de B est gagnant (pour A) si et seulement si il existe un fils gagnant. Si la racine est notée comme défaite pour le joueur A, alors il existe une stratégie gagnante pour le joueur B. Sinon il en existe une pour le joueur A.

Montrons par l'absurde qu'il n'existe pas de stratégie gagnante pour le joueur B en début de partie. Pour cela remarquons d'abord que pour toute position où un joueur est gagnant, l'ajout de pions pour ce joueur ne change pas l'issue de la partie en cas de jeu parfait. Autrement dit il n'y a pas de zugzwang au Hex. Supposons maintenant que le joueur B ait une stratégie gagnante. Le joueur A commence par jouer un coup aléatoire, puis applique la stratégie gagnante en tant que deuxième joueur comme si son coup aléatoire n'avait pas été joué et que la case était encore vide. Dès que le joueur A doit jouer sur la case de son premier coup, il joue un autre coup aléatoire qu'il ignorera aussi bien. Ainsi les deux joueurs possèdent une stratégie gagnante ce qui n'est pas possible. Le

Table 5 – Paramètres de l'IA standard

Paramètre	Standard
Descentes	16000
Simulations aux feuilles	1
Formes prises en compte	type 3
Constante RAVE	16000
Constante UCT	0.3

joueur B n'a donc pas de stratégie gagnante. Comme les parties nulles sont impossible, alors il existe une stratégie gagnante pour le joueur A.

Cette démonstration est appelée vol de stratégie, et peut être appliquée à tout jeu à information complète et parfaite, ne présentant pas de hasard, pour lequel les rôles de A et B sont symétriques, et qui ne présente pas de zugzwang.

B Tests

B.1 Protocole

Pour pouvoir juger efficacement de l'évolution du niveau entre plus de deux configurations de paramètres, le plus simple est de les faire jouer contre un adversaire immuable de niveau semblable. Pour atteindre le niveau de *Six* il fallait un grand nombre de simulations par coup, il n'était donc pas possible d'obtenir des statistiques fiables rapidement. Par ailleurs la différence de niveau se maintient avec le nombre de simulations. Il a donc été choisi de fixer une configuration rapide (2 secondes par coups, tableau 5) de *Yopt*, et de faire jouer contre cette version standard les configurations à tester.

Le protocole est le suivant, 200 parties sont jouées sur un plateau de taille 11, 100 avec les blancs et 100 avec les noirs, la partie commence avec le premier coup noir a3. Ce coup permet un début de partie équilibré en contrebalançant l'avantage du trait (c'est un coup joué par les humains lorsqu'ils utilisent la règle du gâteau). En général seul un paramètre est différent des paramètres standards.