Université du Québec à Montréal

BotGammon: Joueur artificiel de backgammon

Travail présenté à
M. Éric Beaudry

Dans le cadre du cours
INF4230-10 – Intelligence Artificielle

Marc-Étienne Déry DERM06099201
Dominick Latreille LATD19099100
François Bilodeau BILF31089104
Philippe Pépos Petitclerc PEPP03049109
Équipe O4L

15 décembre 2014

1 Introduction

L'application que nous présentons est un joueur artificiel de backgammon. Plusieurs techniques d'intelligence artificielle ont été développées pour résoudre ce genre de jeu, et de nouvelles techniques apparaissent encore dans les temps récents au fur et à mesure que diverses techniques d'IA évoluent. Nous utilisons le logiciel libre GNU Backgammon afin d'avoir une interface nous permettant de tester notre application contre différents niveaux d'intelligence artificielle. Ce logiciel est très complet et comprend une panoplie de fonctions nous permettant de tester différentes situations.

L'idée d'un joueur artificiel pour le jeu de backgammon nous a semblé intéressante tant au niveau du processus de création qu'au niveau du résultat final. La présence du hasard, associée à un bon niveau de stratégie rend ce jeu de table original et stimulant. Le joueur artificiel permet à un joueur apprenti de se pratiquer et d'évaluer ses capacités, en fonction du niveau de l'intelligence de l'agent.

2 Problématique

Le type de problème à résoudre lorsqu'on développe un joueur artificiel de backgammon est un problème de prise de décision avec probabilité. Il s'agit d'évaluer le meilleur coup dans la limite des coups possibles en tenant compte des probabilités des lancers de dés.

Le backgammon est un environnement entièrement observable, puisqu'on voit le plateau de jeu en tout temps. Le joueur artificiel fait face à un environnement stochastique puisqu'il doit brasser deux dés avant de pouvoir jouer. Il ne sait donc jamais avec certitude la distance que ses pions parcourront, mais il a accès aux probabilités des différents lancers de dés possibles. Ensuite, l'environnement est multiagent, car le backgammon se joue à deux joueurs. Il est également séquentiel, puisque chaque décision a un effet sur les décisions futures des deux joueurs. De plus, le backgammon est un environnement statique, car aucun changement à l'environnement ne peut être effectué par un agent externe et parce qu'il n'y a pas d'horloge, contrairement aux échecs dans certains cas. Pour finir, le joueur artificiel fait face à un environnement discret, puisque les endroits où l'on peut déposer les pions sont bien définis et que chaque coup joué est distinct d'un autre.

Le défi dans la conception d'un joueur artificiel de backgammon réside dans le fait que le nombre de positions est très élevé, alors la solution naïve qui consiste à vérifier le meilleur coup dans une table de positions n'est pas réalistement faisable. Pour calculer le nombre de positions légales possibles sur le plateau de jeu, posons :

$$C(n, m) = n!m!(n - m)! \tag{1}$$

représentant le nombre de façons de sélectionner m pions dans un ensemble de n pions et :

$$D(n,m) = C(n+m-1,m) \tag{2}$$

représentant le nombre de façons de distribuer m pions de la même couleur sur n *flèches*

En supposant que les pions noirs occupent m flèches sur les 24 possibles, il y a donc $C(24,m)$ façons de sélectionner les flèches, pour m pions. Le reste des $15-m$ pions peuvent être sur les mêmes flèches que les m précédents, sur le bar ou à l'extérieur du plateau ; le moyen de les distribuer est alors $D(m+2,15-m)$. Les pions noirs ont $26-m$ endroits disponibles, alors ils peuvent être distribués en $D(26-m,15)$ façons. Le nombre total de positions possibles est donc :

$$m = \sum_{m=0}^{15} C(24,m) \times D(m+2,15-m) \times D(26-m,15) \tag{3}$$

$$m = 18528584051601162496 \tag{4}$$

Ajoutons à cela le hasard qu'amènent les dés, le backgammon est loin d'être trivial à jouer pour un ordinateur.

3 Pertinence de la technique d'intelligence artificielle

Parmi les multiples techniques d'intelligence artificielle existantes, celles qui sont les plus utilisées pour le cas du jeu de Backgammon sont la stratégie de prise de décisions expectiminimax et l'apprentissage par réseaux de neurones (comme c'est le cas pour GNU Backgammon). Ces approches sont très intéressantes, mais nous avons décidé d'opter pour l'algorithme expectiminimax avec élagage alpha-bêta et profondeur itérative pour notre IA puisqu'il fonctionne avec les jeux à somme nulle comme le backgammon, en plus d'intégrer des éléments de hasard. C'est aussi une question de temps ; nous désirions obtenir un bon résultat le plus rapidement possible, sans avoir à laisser un algorithme d'apprentissage tourner pendant des heures ou des jours. L'heuristique au backgammon peut être sans cesse améliorée, et c'est donc ce sur quoi nous nous sommes penchés.

L'algorithme expectiminimax est une variation de minimax qui possède en plus du noeud *min* et *max* un noeud *chance*. Ce noeud chance représente l'élément de probabilité du problème associé, qui est dans notre cas les deux dés lancés avant chaque coup. Tout comme le minimax, cet algorithme va créer un arbre récursif et alterner entre le joueur *min* pour qui l'on cherche à minimiser son gain et le joueur *max* que l'on veut maximiser. Par contre, le noeud chance sera appliqué avant chaque position hypothétique du plateau dans l'arbre, donc avant le joueur *min* et avant le joueur *max*. Contrairement aux noeuds *min* et *max* qui

prennent les valeurs d'utilité de leurs enfants, le noeud de chance prend comme valeur la probabilité d'avoir une combinaison de dés quelconques multipliée par la valeur du noeud *min* ou *max*. Pour le backgammon, nous avons une probabilité de 1/18 d'avoir deux dés différents ou 1/36 d'avoir un double. La complexité temporelle d'expectiminimax est $O(n^d b^d)$ plutôt que $O(b^d)$ pour minimax, où n équivaut au nombre de possibilités des résultats de chance. Dans le cas du backgammon, il y a 21 combinaisons de dés possibles.

Pour la profondeur itérative, c'est une méthode qui nous permet de faire une recherche le plus profond possible avec le temps alloué pour l'IA. Dans le cas où l'on ne réussit pas à terminer une itération à une certaine profondeur par manque de temps, on prend le résultat de la précédente itération. Nous utilisons également l'élagage alpha-bêta pour notre expectiminimax afin de réduire le nombre de noeuds à visiter. Cela permet donc de couper certaines branches dans l'arbre de recherche lorsque l'on voit qu'il est impossible qu'un joueur le sélectionne.

Afin de rendre la tâche possible, nous avons pris en considération certaines hypothèses. Tout d'abord, nous considérons qu'il faut 1 point seulement pour gagner (1 partie = 1 point), contrairement aux parties de tournoi qui demandent habituellement entre 3 et 7 points pour gagner. Cela nous facilite la tâche pour les tests et est tout de même un bon indicateur pour la prise de décisions. De plus, nous n'utilisons pas le dé doubleur. Le fait de ne pas pouvoir doubler nous permet d'avoir un gain en rapidité pour chaque coup puisque nous n'avons pas besoin de calculer la probabilité de gagner avant de brasser les dés.

4 Résultats

Afin d'évaluer l'algorithme implémenté, nous avons effectué un grand nombre de parties contre l'algorithme de base de GNU Backgammon. Nous avons testé avec deux heuristiques différentes, avec élagage alpha-bêta des noeuds *min* et *max* et à différentes profondeurs. Nous avons aussi fait quelques tests avec une profondeur itérative. La première heuristique que nous avons réalisée, surnommée Franklin, pénalise les pions non protégés, récompense les pairs et les groupes de pairs (qui bloquent le passage), récompense lorsqu'il mange un ennemi, et récompense lorsqu'il sort un pion du plateau de jeu. Le tout est pondéré selon la position des pions. Pour la deuxième heuristique, plus simple, nous pénalisons les pions non protégés, récompensons les pions protégés, et récompensons lorsqu'il y a des pions ennemis dans son bar. Certains résultats que nous avons obtenus sont disponibles en annexe.

5 Conclusion

Comme on peut le constater, notre algorithme, s'il est effectué à profondeur 1, est très rapide. Il joue environ une partie à la seconde, avec un taux de victoire d'environ 30% pour l'heuristique Franklin, et 39% pour l'heuristique Simple. Quant à la profondeur 2, le temps de calcul est augmenté de façon exponentielle,

avec plus d'une seconde par coup (environ 3 minutes par partie). Nous avons donc testé avec moins de parties par manque de temps. La différence de résultats entre la profondeur 1 et 2 est malheureusement minime et donc peu concluante. Lorsque l'on a essayé de faire des tests avec la profondeur itérative, nous n'avons remarqué aucune amélioration. On note même une diminution du taux de victoire (même si l'échantillon est faible) lorsque nous atteignions des profondeurs supérieures à 2 et beaucoup plus. Ce problème est sûrement dû à une heuristique peu efficace dans les noeuds plus profonds ou à un défaut dans l'expectiminimax.

Parmi les pistes de solution, il est très probable que nous n'ayons pas réussi à trouver les deux constantes qui permettent d'évaluer adéquatement les feuilles en considérant les noeuds chances. La fonction d'évaluation des noeuds chances devrait seulement modifier le poids relatif des meilleures actions et pas leur ordre.

5.1 Identification de pistes pour l'amélioration de Botgammon

— Simulation de Monte-Carlo :

À chaque position, faire jouer l'algorithme contre lui-même des milliers de fois en utilisant des dés aléatoires. En évaluant le pourcentage de victoire, cela peut pratiquement remplacer l'heuristique.

— Il serait même possible de construire une base de données pour toute les positions rencontrées et leur chance de victoire. Nous pourrions ainsi facilement inclure le dé doubleur ainsi que la possibilité d'abandonner une partie si on est sûr de perdre. Si la position est dans la base de donnée, on vérifie, sinon on fait une simulation de Monte-Carlo.

— Élagage pour les noeuds chance

— Meilleure heuristique

6 Répartition des tâches

— **Dominick** : Document, présentation, fonctions pour la grille de jeu, expectiminimax, tests et résultats

— **François** : Heuristique, liste des coups possibles, mouvement de pion, document, tests et résultats, présentation

— **Marc-Étienne** : Expectiminimax, liste des coups possibles, structure du programme, interaction avec GNU Backgammon, tests et résultats, heuristique

— **Philippe** : Document, test et résultat, généricisation du code, expectiminimax sans alpha-bêta, profondeur itérative avec et sans alpha-bêta

Annexe

15 décembre 2014

1 Tableau des résultats

Fonction d'évaluation	**Élagage alpha-bêta**	**Pourcentage de victoire**	**Nombre de parties**	**Profondeur**	**Temps moyen des coups**
Aucune	-	0.5%	1000	-	$<$ 1ms
Heuristique Franklin	✓	30,0%	1000	1	2ms
Heuristique Franklin	✗	27,8%	1000	1	2,5ms
Heuristique Simple	✓	38,8%	1000	1	2ms
Heuristique Simple	✗	39,1%	1000	1	2ms
Heuristique Franklin	✓	30,0%	10	2	3,1s
Heuristique Franklin	✗	0,0%	10	2	1,7s
Heuristique Simple	✓	40,0%	10	2	1,2s
Heuristique Simple	✗	0,0%	10	2	2,1s
Heuristique Simple	✓	30,0%	10	Profondeur itérative	1s (constant)
Heuristique Simple	✗	0,0%	10	Profondeur itérative	1s (constant)