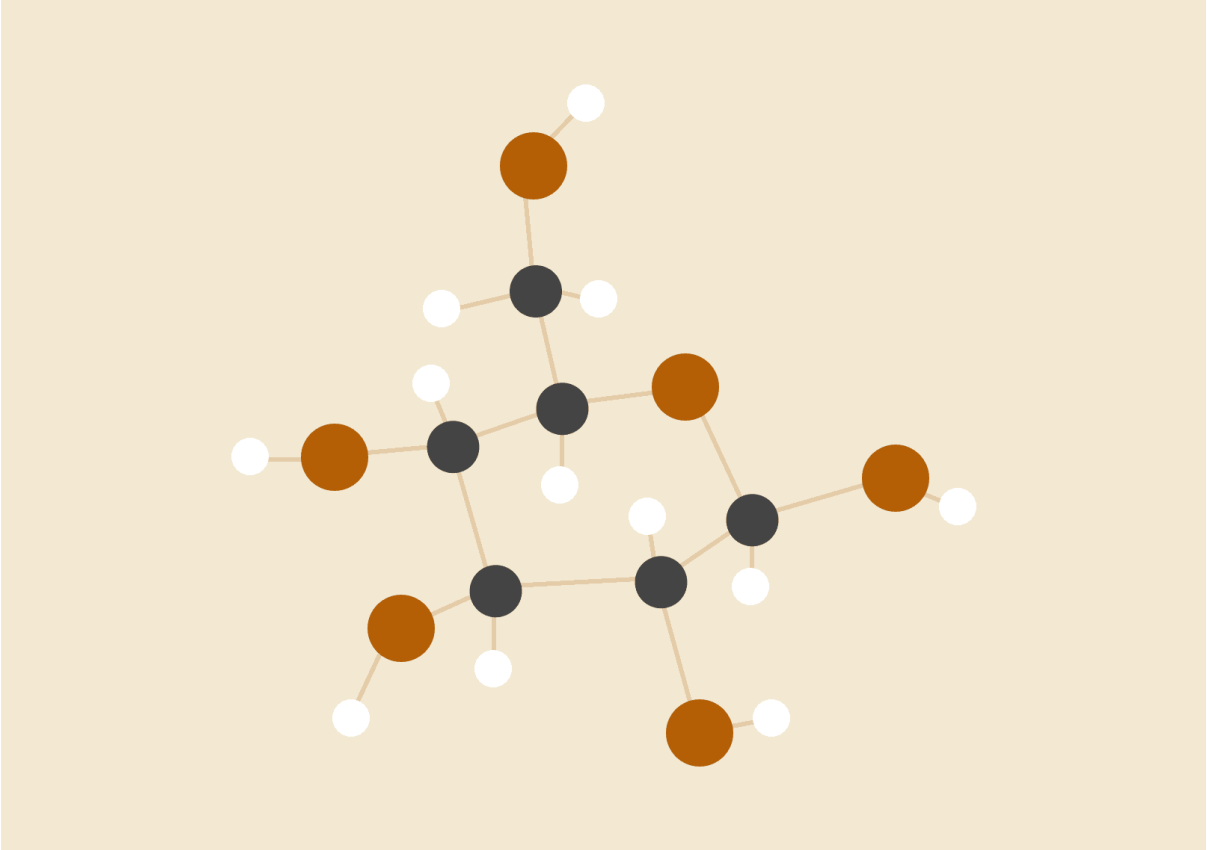
Projet – IA pour Avalam

INF 8215 – Intelligence artificielle : Méthodes et algorithmes



Idriss JEAIDI 2233451

Younes Allouch 2231842

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Nom d'équipe : "allouchi"

INTRODUCTION

Dans le cadre du projet du cours INF 8215 IA, il nous a été demandé de réaliser un agent capable de jouer au jeux avalam. Avalam est un jeu de stratégie à deux joueurs en un contre un. De ce fait il s'agit d'un jeu en tour par tour, déterministe, à information parfaite et enfin à somme nulle. Dit autrement, il n'y a pas de présence aléatoire dans le jeu, toutes les actions d'un joueur sont visibles par l'autre, et enfin, ce qui est considéré comme bon pour un joueur est forcément considéré comme mauvais pour l'autre.

Par conséquent, les agents que nous avons implémentés se basent sur ces modélisations du jeu, afin de trouver une des combinaisons d'action capable d'aboutir à une victoire. En outre, le jeu se joue sur un plateau de 48 positions supportant chacune un pions. Il y'a donc 24 pions de couleurs rouge appartenat au joueur 1, et 24 de couleurs jaune appartenat au joueurs 2. Par ailleurs, à chaque tour, chaque joueur doit réaliser un déplacement d'un pion ou d'une tour de n'importe quel couleurs dans son pourtours (une distance d'une seule positions) afin de l'empiler sur un autre pion ou tour et ainsi former une tour de taille maximum 5.Une tours est considérée comme appartenant à un joueur si celle-ci est surmonté par un pions de la couleur du joueur en question. Cependant ces mouvement doivent suivre quelque règle importante:

- Les déplacement sur des positions vide sont interdit
- Lors de déplacement, la tour complète doit se mouvoir
- Une tours de taille 5 ne peut se déplacer

Enfin, le jeu est considéré comme fini lorsque aucun déplacement n'est possible. Et suite à cela, le décompte des tours est réalisé, ainsi attribuant la victoire au joueur possédant le plus de tours. Dans le cas d'une égalité, le décompte de la taille des turs est réalisé, et le joueur ayant les tours les plus grandes se voit attribuer la victoire.

Etant donné le nombre important de positions sur Avalam qui est de 48, et de la possibilité de superposer des tours de 5 tailles différentes, le nombre d'états possible est beaucoup trop important, ce qui limite les profondeurs atteignables, sii l'on souhaite rester dans les limites de temps. Ainsi, notre stratégie est simple, réduire le nombre d'actions explorées le plus possible afin d'atteindre une profondeur plus élevée.

Méthodologie

1) Alpha Beta pruning

Pour cette compétition, nous avons choisi d'utiliser l'algorithme alpha beta pruning. Cet algorithme est une amélioration de l'algorithme Min Max dont le principe est le suivant : À chaque tour, une simulation de toutes les actions possibles par notre joueur est effectuée. Un simulation signifie que l'algorithme va jouer l'action et mesurer si cela à mener à un bon score pour notre joueur. S'il s'arrête là, on dit que l'algorithme à une profondeur de 1. Si l'on augmente la profondeur à 2 par exemple, l'on cherchera donc à considérer quelle action aurait jouer l'adversaire en réponse à la nôtre.

Par conséquent, plus la profondeur est élevée, plus le score attribué à une action est digne de confiance. Un action qui garantit une bonne position pour notre joueur après une recherche jusqu'en profondeur 2 est beaucoup moins fiable que si l'algorithme nous garantit une position avantageuse après une recherche à la profondeur 4.

Ainsi, les performances d'une Intelligence Artificielle basée sur l'Alpha Beta Pruning dépend de 2 éléments :

- ➔ Les heuristiques d'action, qui permettent de filtrer les actions à simuler.
- ➔ L'heuristique de score, qui permet de juger un état du jeu, afin de déterminer par la suite si une action est prometteuse ou non.

Dans les parties suivantes, nous allons expliquer notre stratégie, les heuristiques qui en découlent et les différents agents que nous avons testé.

2) Stratégie

Notre stratégie se base sur trois points importants que nous avons observés :

Dans un premier temps, comme nous venons de le voir, à heuristique de score égales, deux algorithmes d'Alpha Beta pruning seront départagés par la profondeur qu'ils atteignent. Dans un premier lieu, nous avons testé différentes profondeurs afin de déterminer les échelles de temps pour ces dernières.

Une seconde observation importante est que le nombre d'états diminue au fur et à mesure de la partie, ce qui implique que le temps nécessaire à l'agent pour jouer diminue aussi.

Et troisièmement, nous avons observé que pour la majorité de la partie, il est impossible de dépasser une profondeur de 3. À partir de 4, les coups prennent en moyenne 10 minutes à être joués, ce qui n'est pas envisageable car le temps disponible est de 15 minutes. Par conséquent, il est très peu probable que d'autres personnes arrivent à atteindre une profondeur de 4 durant la première moitié de la partie, car il faudrait filtrer un nombre considérables d'actions. Cependant, une fois qu'un certain nombre de coups ont été joués, il est possible d'augmenter la profondeur à 4 sans que l'agent ne prenne trop de temps à jouer.

Par conséquent, à l'exception de l'heuristique de score qui risque de différer, les agents des autres élèves devraient tous explorer jusqu'à une profondeur de 3 au début de partie, ce qui signifie que la gagnant sera celui qui passe à la profondeur 4 le plus tôt (à heuristique de score égale).. La clé de la victoire se trouve donc au moment à partir duquel un agent peut se permettre d'augmenter sa profondeur. Etant donné que le temps est limité, plus un agent joue vite au début, plus tôt il pourra se permettre de passer à la profondeur 4.

Notre stratégie est donc de filtrer le plus d'actions inutiles possible afin d'accélérer notre agent pour pouvoir augmenter la profondeur le plus tôt possible, nous donnant un avantage sur nos adversaires.

3) Heuristique d'action

Nous avons décidé de ne filtrer que les mauvaises actions, car nous pensons que filtrer de potentielles bonnes actions pour gagner du temps n'en valait pas la peine. De plus, nous avons remarqué qu'il était très difficile de déterminer si un type d'action est bon ou mauvais dans l'absolue, nous avons parfois retirer certains types d'actions qui paraissaient très mauvaises pour ensuite se rendre compte que l'agent était considérablement plus mauvais qu'avant. Nous avons donc seulement conservé les heuristiques qui rendaient l'agent meilleur. Comme baseline pour nos comparaisons, nous utilisons un algorithme Alpha Beta Pruning sans heuristique d'action, avec comme heuristique de score la fonction ***get_score()*** fournie dans le fichier avalam.py, et avec une profondeur constante de 3.

La première heuristique que nous utilisons (appelons la H1) consiste à pruner toutes les actions qui posent une tour de l'adversaire sur l'une des nôtres. Il s'agit de l'heuristique principale de notre algorithme, car elle permet de diviser le temps de recherche par 2 comparé à la baseline. Cependant cette heuristique n'à pas augmenter les performances de jeu de notre agent. En effet, en utilisant H1 l'agent prend 2 fois moins de temps mais joue toujours de la même manière.

La deuxième heuristique (H2) consiste à pruner toutes les actions qui mènent à la création d'une tour de 4 qui nous appartient. Nous utilisons cette heuristique car nous avons remarqué que dans la grande majorité, si notre agent joue une tour de 4 de sa couleur, alors il suffit juste au joueur adverse de poser une tour dessus pour gagner une tour de 5. Cette heuristique permet un gain de temps modéré et un gain de performance majeur, car notre agent avait tendance à souvent construire des tours de 4.

La troisième heuristique (H3) consiste à pruner toutes les actions qui vont créer des tours de notre couleur à côté de tours adverses de tailles complémentaires (dont la somme est 5). Cette heuristique est celle qui permet le plus gros gain de performance (en terme de niveau de l'agent), cependant elle prend beaucoup de temps de calculs, au point de perdre tout le temps gagné par H1 et H2. Nous la prenons en compte pour l'instant car le gain de performances est vraiment intéressant, mais nous verrons dans l'analyse des résultats pourquoi elle n'est pas si bonne que ça.

4) Heuristique de score

Durant nos phases de recherche d'heuristique , nous avons trouvé pertinent de penser à la réalisation de l'heuristique de score. En Effet, nous nous somme attribué comme baseline la fonction "get_score", celle-ci renvoie le score à n'importe quel état du plateau, de plus elle est entre autre utilisée lors de la détermination du vainqueur en fin de partie. Nous avons aussi réalisé d'autres heuristiques qui prenaient en compte différents autres critères de réussite, et nous les avons testé les unes contre les autres. Cependant, les résultats étaient très décevants. Toutes les heuristique testé en utilisant les même agents perdait fermement contre l'heuristique "get_score".

De ce fait, nous avons donc décidé de la laisser pour la compétition, afin de la tester dans un vrai environnement compétitif. Et en convenance avec nos essaie, l'agent avec l'heuristique "get_score" a bien performer et est arrivé en finale.

Néanmoins, après l'analyse de plusieurs parties, nous avons remarqué que l'heuristique baseline ("get_score") affecte un score de 1 lorsque l'on possède une tour de plus que l'adversaire et renvoie la différence des taille de tours lors d'égalité. Par conséquent, le score obtenu lors d'une égalité sera toujours supérieur ou égale à 1. Ce phénomène induit donc l'agent à prioriser les états ou il ya égalité.

Pour trouver une solution à ce problème de priorisation, nous avons décidé de réaliser une petite modification à notre baseline. Au lieu de rendre la différence des tailles des tours lors d'égalité, nous avons décidé de diviser le résultat de cette soustraction par 50 (nous avons choisi ce chiffre car il y a un total de 48 tours sur le plateau), afin que le résultat soit compris entre 0 et 1 exclu. Cette opération nous permet donc de donner un score plus important aux états où il y'a victoire directe (plus grand nombre de tours que l'adversaire). Enfin,nous avons appelé cette heuristique "get_score2".

Résultats pertinents et évolutions de l'agent

Comme mentionné à plusieurs reprises durant ce rapport, nous avons réalisé des tests avec différents agents, selon les heuristiques d'action ou de score que nous leur avons fournies. Voici un tableau récapitulatif des résultats des différentes parties réalisées entre les agents. Veuillez noter que ces résultats sont réalisés dans des conditions bien spécifiques : nous n'avons pas considéré les contraintes de temps, et les résultats entre agents ont été obtenus en réalisant 5 matchs en inversant à chaque fois l'agent en player 1 et celui en player 2.

	P1	P2	P3	P4
P1		3/2	4/1	5/0
P2	2/3		3/2	4/1
P3	1/4	2/3		4/1
P4	0/5	1/4	1/4	

Légende : le premier des 2 chiffres dans une cellule correspond aux nombres de parties gagnées par l'agent sur la colonne, et le deuxième correspond à celui sur la ligne.
P1 : agents sans heuristique d'action (baseline)
P2 : agents avec l'heuristique H1
P3 : agents avec l'heuristique H1 et H2
P4 : agents avec l'heuristique H1, H2 et H3

Dans les conditions de test que nous nous sommes fixées, nous pouvons clairement remarquer une dominance de l'heuristique H3 par rapport aux autres. Nous pouvons aussi remarquer que l'heuristique H2 est en deuxième position et reste malgré tout très efficace.

De la même manière, nous avons aussi testé les 2 heuristiques de scores "get_score"(nommée HS1) et "get_score2"(nommé HS2) sur 10 matchs. Les résultats étaient les suivants : pour HS1(3/10) et HS2 (7/10). Ainsi, nous pouvons voir que l'heuristique HS2 est supérieur à l'heuristique HS1.

Néanmoins, et comme mentionné précédemment, notre objectif principal dans la conception de notre agent, était de réussir à aller à une profondeur plus grande le plus rapidement possible. Et lors de la réalisation des tests ci-avant, la notion de temps n'avait pas été prise en compte. Nous avons donc, décidé de de prendre les 2 meilleurs agents obtenu (P3,l'agent avec l'heuristique H1 et H2 et P4 celui avec l'heuristique H1,H2, et H3) et les dresser l'un contre l'autre dans un match avec les contraintes de temps imposées.

Suivant notre stratégie énoncée dans la partie antérieure, nous avons choisi pour chacun des agents une profondeur de départ de 3, puis après avoir dépassé le 24ème tours, nous les passons à une profondeur de 5. Contrairement à nos attentes, l'agent P3 à batut l'agent P4 à chaque partie. Après analyse des parties, nous avons pu conclure que la raison principale est que l'agent prenait trop de temps à réaliser ces décisions et perdait au temps. De ce fait, nous avons réussi à diagnostiquer que l'heuristique H3 prenait trop de temps à calculer. Nous avons donc décidé d'appliquer cette dernière entre les tours 10 et 16. Cette fois-ci, nous avons remarqué que l'agent qui gagne était P4. Nous attribuons cela au fait que l'heuristique H3 empêche l'agent de faire des actions à son détriment pendant la durée de son application. De ce fait, notre dernière version de l'agent était le P4 avec les heuristique H1, et H2 et avec H3 entre les tours 10 et 16.

DISCUSSION

En conclusion, nous voulons d’abord dire que les résultats de notre agent semblent très convenables et satisfaisants à notre égard, que ce soit dans le cas de la compétition ou dans nos tests personnels. Cependant, nous avons pu détecter certaines pistes d'amélioration la plus pertinentes d’entre elles est la suivante :

Tout d’abord notre management de la profondeur pourrait être amélioré. En effet,nous avons supposé, pour la réalisation de passage à des profondeur plus grande, que les partie dure en moyenne 35 tours. Cependant, en pratique certaines des parties peuvent durer plus de 40 tours alors que d'autres durent moins de 30 tours.Dans le cas où les partie durent plus de 40 tours notre agents prendra trop de temps pour réaliser ces décision et à plus de chances de perdre à cause d’une surconsommation de ces ressources temps. Alors que dans le cas où les parties durent moins de 30 tours, notre agent reste à une faible profondeur.

Malgrés ces pistes d’améliorations, nous avons remarqué que notre agents performes bien et nous aurions aimé le voir performer avec les améliorations