POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

Département de génie informatique et génie logiciel

INF8215

Intelligence artificielle: méthodes et algorithmes

Projet - Agent intelligent pour le jeu Avalam

Nom de l'équipe Challonge: **tomtom103**

Slimane Boussafeur - 2017001
Thomas Caron - 1944066

8 décembre 2022

Méthodologie

En ce qui concerne notre agent remis lors de la compétition sur Challonge, nous avons opté pour un algorithme **min-max avec alpha-beta pruning**, ce qui nous permettait de battre les agents de base qui nous avaient été fournis (random et greedy), ainsi que certains agents de nos camarades.

Après plusieurs essais entre les différentes variations de min-max et autres algorithmes que nous avions implémentées, nous avons remarqué que celles-ci n'offrent aucun gain en performance. Nous sommes donc plutôt tournés vers la conception d'une meilleure heuristique, en gardant un algorithme relativement simple.

Heuristique

Notre heuristique prend en compte en **2 composantes** : l'**état du *board*** après qu'une certaine action soit jouée ainsi que l'**action** en question.

Pour ce qui est du **board** on ne fait que **calculer son score**.

Pour ce qui est de l**'action**, on évalue de quel type d'action il s'agit et on lui accorde un score en conséquence. En premier lieu, on regarde si cette action nous permet de **faire une tour** et comme il existe plusieurs configuration (certaine meilleur que d'autre), voici un tableau résumant celles-ci avec leurs scores :

Configuration de l'action (origin. -> dest.)	Score
1 -> -4	160
1 -> 4	150
4 -> -1	160
4 -> 1	150
2 -> -3	140
2 -> 3	135
3 -> -2	140
3 -> 2	135

Bien évidemment qu'en début de partie, il très peu probable d'obtenir ce genre d'actions. De ce fait, le comportement prioriser est de faire **des tours de deux** avec les pièces de l'adversaire ou bien les nôtres selon une certaine probabilité. Cela à pour but d'amener un imprévisibilité à notre agent. Les paires ont un score de **20**. Ce mécanisme va comme suit :

Probabilité	Action
[0.9; 1.0]	Nos paires
] 0.9; 0.85 [	Ne fait pas de paires
[0; 0.85]	Paires adverses

Au cours de la partie, le nombre de pairs va augmenter selon notre heuristique jusqu'à ce qu'on ne puisse plus en faire. Ainsi, nous allons commencer à prioriser l'**empilement des paires adverses pour en faire des tours de 4**. Ceci à pour but de réduire le nombre de pièces disponibles de l'adversaire sur le *board*. Ce type d'action à un score attribué de **15**.

Finalement, nous avons implémenté un mécanisme qui se déclenche en fin de partie (c-à-d. *nb_actions_jouées*/2 > 10). Nous avons remarqué que **lors des fins de parties**, nous pouvions gagner des tours de manière sécuritaire en faisant **des tours de 4**, car les pièces uniques avaient disparues du *board* ou bien elles étaient isolées à ce stade du jeu. De ce fait, voici les différent cas de figure ainsi que leurs scores associés :

Configuration de l'action (origin. -> dest.)	Score
2 -> 2	25
2 -> -2	35
1 -> 3	25
1 -> -3	40
3 -> 1	25
3 -> -1	35

Le score retourné par l'heuristique est une addition du score du board ainsi que celui de l'action.

Gestion des 15 minutes et prise en compte du timeout:

Nous prenons en compte le timeout dans le code de notre agent. En effet, lorsqu'une limite de temps est définie, s' il ne reste plus assez de temps, on coupe l'exécution de notre algorithme, et l'agent devra choisir l'action qui lui rapporte le plus de points sans continuer sa recherche. Cette considération fait directement partie de notre implémentation de min-max.

Nous avons d'ailleurs fait plusieurs tests en ce qui concerne la profondeur de récurrence pour trouver une valeur idéale qui nous donnait un gain de performance sans dépasser le temps alloué de 15 minutes. Nous avons trouvé qu'une valeur de 4 semblait être un juste milieu.

Résultats et évolution de l'agent

En ce qui concerne notre agent, nous avons dans un premier temps implémenté un simple algorithme **min-max** selon le pseudo code présent dans les notes du cours. Cet agent était relativement naïf, et parcourait tous les chemins possibles dans l'arbre de recherche, ce qui nous faisait dépasser le temps maximal de 15 minutes. L'agent pouvait difficilement dépasser une profondeur de récurrence de 2-3.

Par la suite, nous avons choisi d'implémenter un deuxième agent utilisant min-max, cependant cette fois nous avons aussi ajouté le **alpha-beta pruning**, encore une fois selon le pseudo code qui se trouve sur les notes de cours. L'introduction de cette technique n'avait aucun impact sur le résultat de la partie comparé à min-max, cependant il nous a permis de "prune" les branches de l'arbre de recherche qui n'ont aucune influence sur la décision finale. Ceci a grandement contribué à la vitesse de notre agent, celui-ci pouvait maintenant atteindre une profondeur de récurrence de 4 facilement en moins de 15 minutes.

L'heuristique utilisée restait cependant encore simple, notre score pour chaque action correspondant au **calcul du score du *board***. On arrivait à des victoires de l'ordre des 7-5 ou 8-6. Nous avons ensuite complexifié l'heuristique en ajoutant la logique de **la détection de tours ainsi que la formation de pairs (avec différents seuils)**. Cela nous à amené vers des victoires présentant des gros écarts tel qu'un mythique 11-2. En ajoutant la logique **empilement des paires adverses**, nous n'avons pas vu grande évolution dans le score. En parallèle, on fait affronter l'agent face à lui-même à chaque étape de l'évolution de l'heuristique. Nous avons constaté que les parties étaient toujours très serrées et résultent souvent en des matchs nuls.

Étant donné que Avalam est un jeu à somme nulle (ce qui veut dire que ***max(a, b) = -min(-a,-b)***), nous avons décidé d'implémenter un troisième agent en utilisant une variante de l'algorithme min-max qui s'appelle **Negamax**. Celle-ci est principalement une simplification du code comparé à min-max, et ne vise pas à offrir d'avantages pour le jeu. Nous avons d'ailleurs aussi ajouté le alpha-beta pruning dans cette version de l'algorithme.
Nous avons testé notre agent face à ce nouvel agent, les deux avec la même heuristique. Nous avons remarqué que le résultat des matchs était souvent nul, donc l'ajout de negamax n'offrait aucun gain important. Nous avons par la suite affronté notre agent face à d'autres **agents conçus par d'autres équipes**.
Nous avons vite constaté que la mauvaise gestion des fins parties nous causait beaucoup de torts et qu'on perdait beaucoup de parties qu'on dominait jusque là. C'est à ce moment-ci que nous avons intégré à l'heuristique les **tours de 4 lors des fins de parties**. Ceci à permis de limiter la casse et de capitaliser sur les bon débuts de parties.

EnLa prochaine étape dans la conception de nos agents pour avoir un avantage sur les autres équipes était l'ajout de "**Transposition Tables**", ce qui nous permettrait de mettre les actions possibles dans une cache, pour éviter de les re-calculer par la suite. Ceci devait en principe nous donner un gain de performance étant donné que pour les jeux comme Avalam, plusieurs actions sont possibles pour arriver à un même état du plateau de jeu. Après avoir testé

plusieurs manières de représenter la table ainsi que plusieurs algorithmes de hashing pour minimiser les collisions nous avons ajouté l'utilisation de ces tables dans un nouveau joueur utilisant **Negamax avec alpha-beta pruning**. À notre grande surprise, cette amélioration ne donnait aucun gain de performance à notre agent. Ceci est principalement dû au fait que pour avoir des gains importants dans notre algorithme, il faut idéalement avoir des données de parties précédentes, étant donné que notre profondeur n'est pas assez importante pour revoir un même scénario à plusieurs reprises. Il aurait été intéressant de sauvegarder de nombreuses parties dans un fichier pour les réutiliser par la suite, mais cette option n'a pas été explorée par manque de temps.

Après plusieurs tests, dans une partie entre notre agent **min-max alpha-beta pruning** et l'agent **negamax alpha-beta pruning** avec les transposition tables, l'agent qui gagnait la partie était toujours celui qui commençait la partie (les agents avaient le même nombre de tours, mais l'agent qui commençait avait toujours l'avantage qu'une de ses tours était plus grande, ce qui lui donnait la victoire).

L'utilisation de Transposition Tables est cependant intéressante pour de futures améliorations pour introduire le principe de "***move ordering***", qui nous permet de trier les actions possibles selon leur possibilité de gain de points. Ceci nous aurait permis d'introduire d'autres possibilités intéressantes, comme par exemple le "***Principal variation search***", ainsi que d'autres méthodes de tri d'actions comme le "***Killer Heuristic***". Nous n'avons cependant pas eu le temps d'implémenter ces différents algorithmes pour les tester.

Discussion

Avantages :

Après avoir fait de nombreuses parties, nous nous sommes rendu compte que nous avions un agent qui **perdait peu en player 1**. Cela est dû au fait que notre première action est fixe et se fait en haut à gauche du *board*. Ainsi, grâce à la nature de notre heuristique, on arrive à nettoyer la partie haute du board et accumuler le plus de tours possible pour compenser nos mauvaises fin de parties.

Un des autres avantages de notre agent, c'est qu'il **joue très rapidement en fin de partie**. En effet, moins il y a d'actions possibles à évaluer, plus il va rapidement, notamment grâce à la nature de notre heuristique. De ce fait, on évite des fins de parties interminables qui peuvent nous coûter une défaite par *time-out* .

Limites :

En contrepartie d’être bon en player 1, notre agent ne **semble pas être très bon en player 2**. Nous aurions dû avoir une variation de l’heuristique dépendamment de si nous étions en player 1 ou player 2.
Le plus grand talon d’achille de notre heuristique est son inefficacité face aux agents ayant une ouverture. La majorité des agents que nous avons affrontés ayant une ouverture nous ont battu à plat de couture. Car ces ouvertures font généralement des paires de nos pièces de sorte à ce qu’elles soient isolées les unes des autres et entourées des pièces adverses. De ce fait, il nous est difficile de faire des tours et donc d’efficacement utiliser notre heuristique. Nous n’avons pas pu trouver un moyen efficace d'atténuer l’effet qu’on les ouvertures face à notre agent.