POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

INF8215

Projet Avalam

Automne 2022

Groupe 01

Dépelteau, Nicolas - 2083544

Michaud, Elizabeth - 2073093

Soumis à Quentin Cappart

Remis le 8 décembre 2022

1. Titre du projet

Projet - Agent intelligent pour le jeu Avalam

2. Nom d'équipe sur Challonge

Équipe Nicolas et Elizabeth
Nicolas Dépelteau - 2083544
Elizabeth Michaud - 2073093

3. Méthodologie

Fonctionnement

Notre agent se base sur l'algorithme d'arbre de recherche Monte Carlo. L'idée est d'estimer la qualité des nœuds en utilisant une approche statistique à l'aide d'un échantillon composé de simulations. La première étape est de construire un nœud qui pourra être utilisé pour calculer la statistique et composé l'arbre des possibilités. Un nœud est composé de l'état de la table du jeu Avalam, du joueur à qui c'est le tour, d'un nœud parent, à l'exception du nœud racine, de nœuds enfants et de deux nombres pour calculer l'espérance du score. Le premier nombre est la somme de tous les scores obtenus par l'échantillon de ces nœuds fils et le second nombre est la taille de cet échantillon. Un exemple visuel de l'arbre binaire avec frères et soeurs qui a été utilisé est représenté ci-dessous:

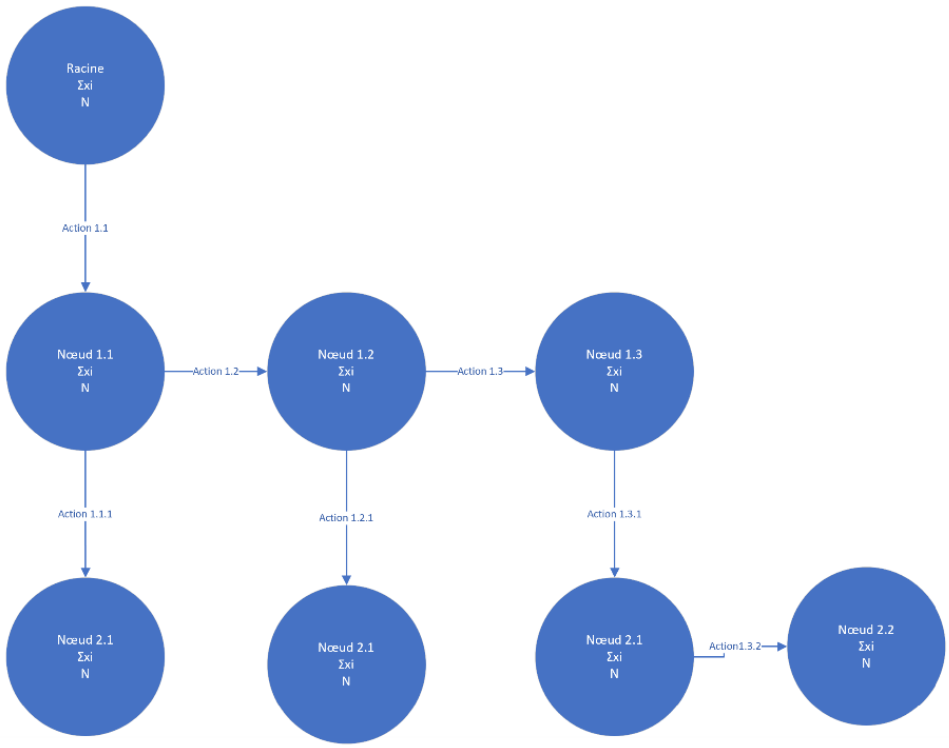

Figure I : Visualisation d'un arbre binaire contenant les noeuds utilisées dans l'agent Monte Carlo

Une fois que l'on a cet arbre, il faut pouvoir le construire. On utilise alors le processus de décision illustré ci-dessous (Figure II). On commence à partir de la racine et on descend graduellement d'un niveau de l'arbre à l'autre en suivant le nœud de meilleure qualité selon une règle de sélection appelée UCT jusqu'à atteindre une

feuille de l'arbre courant. Une fois rendu à la feuille, si celle-ci n'a jamais été simulée, alors on simule une partie à partir de cet état et on met à jour la statistique de ses parents. Dans le cas contraire, on ajoute à ce nœud tous ses enfants et son premier enfant est choisi pour être simulé et mettre à jour les statistiques de son nœud parent. On procède ainsi jusqu'à ce qu' un temps prédéterminé soit écoulé. Finalement, pour retourner l'action choisie par l'agent il suffit de sélectionner le nœud enfant de la racine dont sa taille d'échantillon est la plus grande.

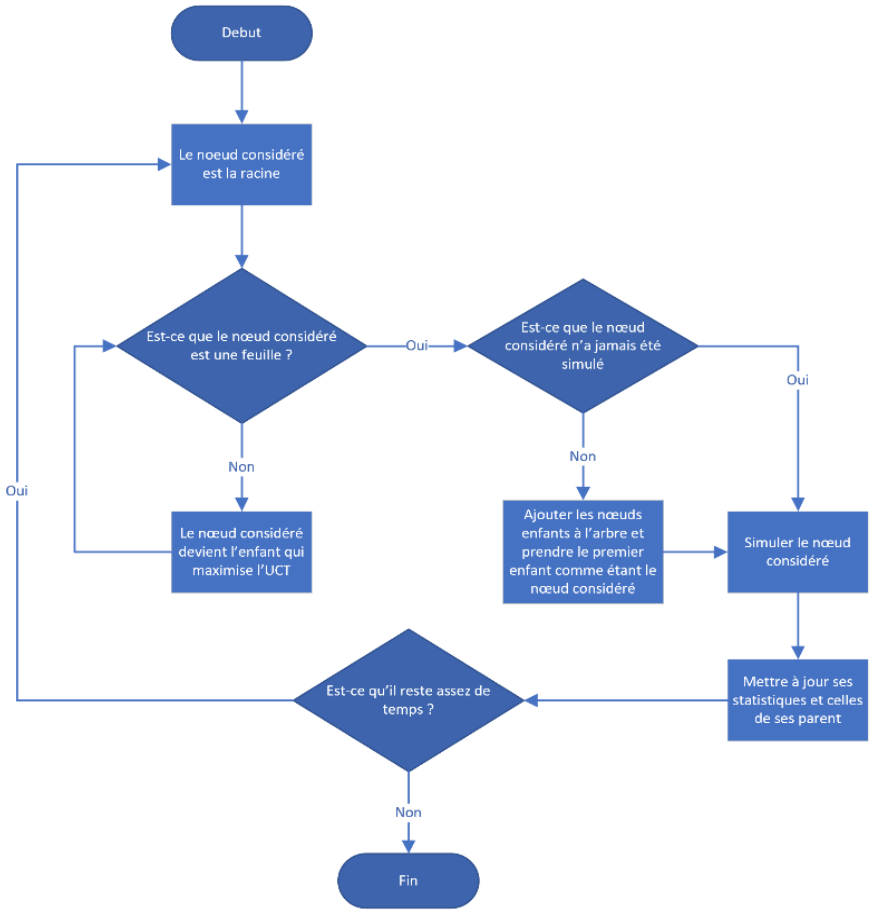

Figure II : Schéma du processus de décision de l'agent Monte Carlo

Choix de conception

Le choix principal de conception est la façon dont les nœuds sont simulés et la fonction qui calcule l'UCT. Notre agent simule les nœuds en choisissant une action au hasard parmi les actions disponibles jusqu'à ce que la partie soit finie. Ce choix a été réalisé puisque l'opération du hasard est très rapide à exécuter et elle n'inflige pas de biais à notre agent. Nous avons essayé des heuristiques pour cette étape, mais le temps d'exécution était trop lent et celle qui était assez rapide avait un biais faisant en sorte que l'action qui était choisie au final était celle de l'heuristique. Pour le choix de l'UCT nous avons utilisé infini si le nombre de simulations est nul, sinon nous avons utilisé l'équation $\frac{\sum_{i=1}^{N} x_i}{N} + \sqrt{2}\sqrt{\frac{ln(N_p)}{N}}$ où x_i est le score obtenu par la i^e simulation, N est le nombre de simulations et N_p est le nombre de simulations de son parent. L'intuition derrière cette équation est pour le premier terme, il s'agit de l'espérance de l'échantillon du score et le deuxième terme sert à l'exploration plus large de l'arbre. Plus le nœud parent et le nœud considéré n'ont pas beaucoup été simulés, c'est-à-dire un petit nombre de simulations, plus le

deuxième terme sera grand et sera favorisé pour être exploré vu qu'il a plus de chance d'être le maximum de son niveau.

Gestion du temps

Notre agent gère son temps selon la répartition représentée avec le graphique ci-dessous. L'idée est d'allouer peu de temps au début et en fin de la partie. Il y a 30 secondes non alloué servant de précaution pour pas que notre agent perdre par faute de temps.

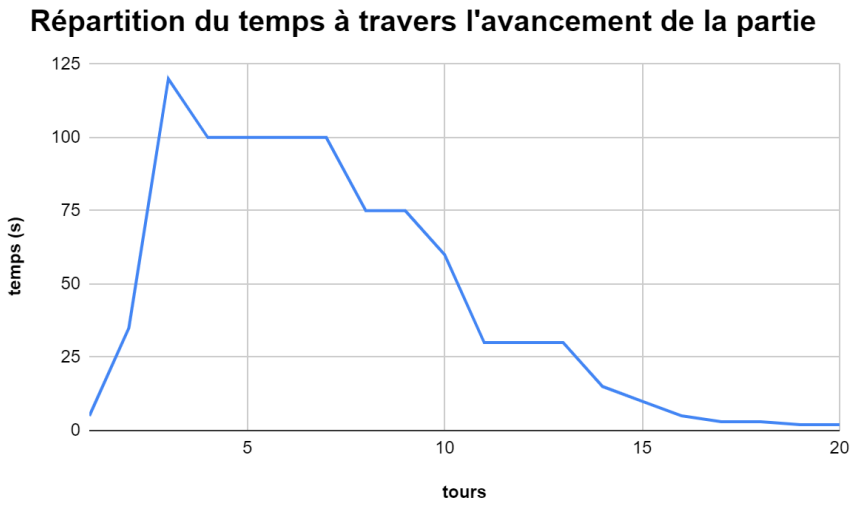

Figure III : Graphique du temps alloué pour les simulation selon le tour

Mécanisme de récupération en cas d'erreur

Notre agent est pourvu d'un mécanisme de récupération en cas d'erreur. Une des erreurs possibles que l'agent peut lancer est qu'il reste moins de 2 secondes. Dans le cas échéant, une action aléatoire est retournée.

4. Résultats et évolution de l'agent

La première version de notre agent Monte Carlo était limitée à 6 secondes par tour pour effectuer le plus de simulations possibles. Cette version de l'agent remportait facilement toutes les parties contre les implémentations fournies (agent Random et agent Greedy). Lorsque nous avons fait s'affronter notre agent Monte Carlo contre nos différents agents Minimax, les résultats étaient beaucoup trop serrés pour décider quel agent était le meilleur. Nous avons donc décidé de répartir les 15 minutes dont nous disposions en 20 tours (40 *step*). Les tours les plus critiques pour le déroulement de la partie se sont vus attribuer beaucoup plus de temps que les tours moins importants. Cette modification a énormément amélioré les performances de l'agent. Les résultats qui sont présentés ci-dessous viennent de parties jouées avec l'agent Monte Carlo comportant cette modification.

L'agent Monte Carlo a tout d'abord été testé contre les implémentations fournies. Lors des tests effectués, nous avons alterné l'agent *Player 1* et l'agent *Player 2* pour que chaque agent joue autant de parties en tant que *Player 1* que de parties en tant que *Player 2*.

a) Agent Monte Carlo contre Agent Random
   Sur les 6 parties entre l'agent Monte Carlo et l'agent Random, l'agent Monte Carlo a remporté toutes les parties avec au moins 11 points de plus que l'agent Random.

b) Agent Monte Carlo contre Agent Greedy
   Sur les 6 parties entre l'agent Monte Carlo et l'agent Greedy, l'agent Monte Carlo a remporté toutes les parties avec en moyenne 5 points de plus que l'agent Greedy.

À la suite de ces parties, nous avons conclu que notre agent Monte Carlo était nettement meilleur que les implémentations fournies.

Nous avons conçu plusieurs autres agents avant la conception de notre agent Monte Carlo final. Toutes ces implémentations ont été testées entre elles et notre agent Monte Carlo final s'est avéré être le meilleur de tous. Encore une fois, lors des tests effectués, nous avons alterné l'agent *Player 1* et l'agent *Player 2* pour que chaque agent joue autant de parties en tant que *Player 1* que de parties en tant que *Player 2*.

Un agent Minimax avec *alpha beta pruning* a été conçu ainsi que quatre différentes heuristiques pour cet agent. Tous les agents Minimax utilisaient une profondeur limite pour que l'agent puisse jouer tous ses coups dans le délai imparti de 15 minutes. La profondeur de recherche était décidée en fonction du nombre de tours joués. Plus la partie était à un stade avancé, plus la profondeur était grande.

a) Agent Monte Carlo contre Agent Minimax avec heuristique score réel
   L'heuristique score réel retournait simplement le score actuel de la partie. Donc, lorsque la profondeur limite était atteinte, l'heuristique retournait le score de la partie selon l'état du plateau de jeu.
   Sur les 8 parties entre l'agent Monte Carlo et l'agent Minimax avec heuristique score réel, l'agent Monte Carlo a remporté 6 parties. Pour ce qui est des 2 parties perdues, le score était de 7 à 7 où l'agent Minimax possédait plus de tour de 5 que l'agent Monte Carlo. Le nombre total de points marqués par l'agent Monte Carlo lors de ces 8 parties était de 62 contre 53 points pour l'agent Minimax.

b) Agent Monte Carlo contre Agent Minimax avec heuristique score garanti
   L'heuristique score garanti retournait le nombre minimal de tours qui allaient rester de la couleur de notre agent jusqu'à la fin de la partie. En d'autres mots, l'heuristique retournait le nombre de tours de la couleur de notre agent qu'il n'était plus possible de déplacer.
   Sur les 6 parties entre l'agent Monte Carlo et l'agent Minimax avec heuristique score garanti, l'agent Monte Carlo a remporté toutes les parties avec en moyenne 3 points de plus que l'agent Minimax.

c) Agent Monte Carlo contre Agent Minimax avec heuristique #1
   Le principe de l'heuristique #1 était d'évaluer à quel point les tours de notre agent étaient "bien positionnées". On estime le score de cette façon : on évalue chacune des tours de la couleur de notre agent. Si la tour a une hauteur de 5, on ajoute 1.2 point au score estimé. Si la tour a une hauteur inférieure à 3 et ne peut pas être déplacée, on ajoute 1.7 point au score estimé. Dans les autres cas, on regarde parmi toutes les actions impliquant la tour traitée s'il y a plus d'actions qui font perdre une tour à notre agent (tour adverse par dessus la tour de notre agent ou deux tours de notre agent une par dessus l'autre) ou plus d'actions qui font perdre une tour à l'adversaire (la tour de notre agent qui écrase une tour de l'adversaire). S'il y a plus d'actions qui font perdre une tour à l'adversaire, on ajoute 1.4 point au score estimé. Sinon, on ajoute 0.8 point au score estimé.
   Sur les 8 parties entre l'agent Monte Carlo et l'agent Minimax avec heuristique #1, l'agent Monte Carlo a remporté 7 parties. Pour ce qui est de la partie perdue, le score était de 6 à 6 où l'agent Minimax possédait plus de tour de 5 que l'agent Monte Carlo. Le nombre total de points marqués par l'agent Monte Carlo lors de ces 8 parties était de 65 contre 50 points pour l'agent Minimax.

d) Agent Monte Carlo contre Agent Minimax avec heuristique #2

L'heuristique #2 était une version améliorée de l'heuristique #1. Le principe de l'heuristique #2 est, tout comme l'heuristique #1, d'évaluer à quel point les tours de notre agent étaient "bien positionnés". On estime le score de cette façon : on évalue chacune des tours de la couleur de notre agent. Si la tour est de la hauteur maximale, un nombre différent de points sera ajouté au score estimé selon l'état de la partie. Si l'écart entre les points de notre agent et de l'agent adverse est de 2 points ou moins et que la fin de partie est proche, on additionne 1.5 point au score estimé puisqu'on estime qu'il y a une bonne probabilité d'égaliser. Donc, pour briser cette égalité nous devons avoir plus de tour de 5. Toujours pour cette même tour, si la fin de partie n'est pas proche et qu'on est plutôt en début de partie, on additionne 1.2 point au score estimé. Les tours de hauteur maximale ne faisant pas partie de ces catégories auront pour effet d'ajouter 1 point au score estimé. Dans les autres cas, on regarde parmi toutes les actions impliquant la tour traitée s'il y a plus d'actions qui font perdre une tour à notre agent ou plus d'actions qui font perdre une tour à l'adversaire. On calcule le nombre de points à ajouter au score estimé de à l'aide de cette formule :
pts à ajouter = 1
+ (nbr actions entourant un tour adverse - nbr actions enterrant une tour de notre agent) / 10.
Sur les 8 parties entre l'agent Monte Carlo et l'agent Minimax avec heuristique #1, l'agent Monte Carlo a remporté 7 parties. Pour ce qui est de la partie perdue, le score était de 7 à 7 où l'agent Minimax possédait plus de tour de 5 que l'agent Monte Carlo. Le nombre total de points marqués par l'agent Monte Carlo lors de ces 8 parties était de 61 contre 50 points pour l'agent Minimax. Donc, l'agent Minimax avec l'heuristique #2 représentait un adversaire un peu plus robuste pour notre agent Monte Carlo final, mais ce dernier était tout de même considérablement supérieur.

Tous ces tests ont permis de conclure que l'agent Monte Carlo était meilleur que nos agents Minimax. Il ne restait qu'à faire s'affronter notre agent Monte Carlo final contre l'agent Monte Carlo RAVE, un autre agent Monte Carlo que nous avions conçu. La principale différence entre les deux agents Monte Carlo était que Monte Carlo RAVE utilisait une formule différente pour estimer la qualité des nœuds. Sur les 6 parties entre l'agent Monte Carlo final et l'agent Monte Carlo RAVE, l'agent Monte Carlo final a remporté les 6 parties avec en moyenne 8 points de plus que l'agent Monte Carlo RAVE.

5. Discussion

Utiliser l'algorithme de Monte Carlo Tree Search pour un jeu comme Avalam procurait quelques avantages. Premièrement, le facteur de branchement du jeu Avalam est très grand, surtout en début de partie et Monte Carlo tend à mieux performer dans ces conditions. Deuxièmement, les heuristiques que nous avons écrites pour l'agent Minimax n'étaient pas des heuristiques qui estiment précisément la qualité d'un état du jeu, car elles n'étaient que de simples approximations de cette qualité basées sur nos connaissances personnelles d'Avalam. Elles n'étaient donc pas basées sur un avis d'expert, mais seulement sur notre compréhension limitée du jeu. Cela rend l'utilisation de l'algorithme Minimax moins intéressante. Ces deux raisons appuient notre décision de choisir l'algorithme de Monte Carlo Tree Search en plus des statistiques des parties jouées qui penchent également vers l'agent Monte Carlo. Un désavantage de notre agent est qu'il doit utiliser presque la totalité du temps imparti pour être réellement performant. Comme nous avons pu le constater, lorsque notre agent Monte Carlo disposait d'un nombre de temps de recherche statique et limité, il n'était pas forcément meilleur que les agents Minimax avec heuristique #1 et heuristique #2. Cependant, dans tous les cas, il est à notre avantage d'utiliser le plus de temps possible. C'est pourquoi notre choix s'est arrêté sur cet agent.