# INF8215 - Projet Avalam
# Rapport

***Abstract*—Ce rapport a pour but d'expliquer et de détailler l'implémentation d'un agent intelligent capable de remporter un jeu en exploitant le principe de recherche adversarielle. Le jeu de société choisi, Avalam, peut être caractérisé comme un jeu déterministe à 2 joueurs, tour par tour, à somme nulle et à information parfaite. Plusieurs stratégies ont été testées au cours de ce projet et seront développées ci-dessous.**

## I. Classement sur Challonge

Nous avons été jusqu'au deuxième tour de la phase éliminatoire du concours regroupant les 32 meilleures équipes. Notre agent est identifiable en tant que et a été soumis par l'utilisateur

## II. Formalisation

L'essentiel du problème ici est de déterminer une politique optimale $\pi^*_s(s) = a$ qui fournit une action $a$ pour un état $s$ donné. Pour ce faire, il est nécessaire de considérer les différentes actions possibles tout en prenant en compte les actions potentielles de l'adversaire.

On pose $S(s)$ le score brut du jeu enregistré à un état $s$. Cela correspond au nombre de tours ayant à leur sommet un jeton de la couleur appartenant au joueur concerné.

## III. Observations sur le jeu

### A. Constatation générales

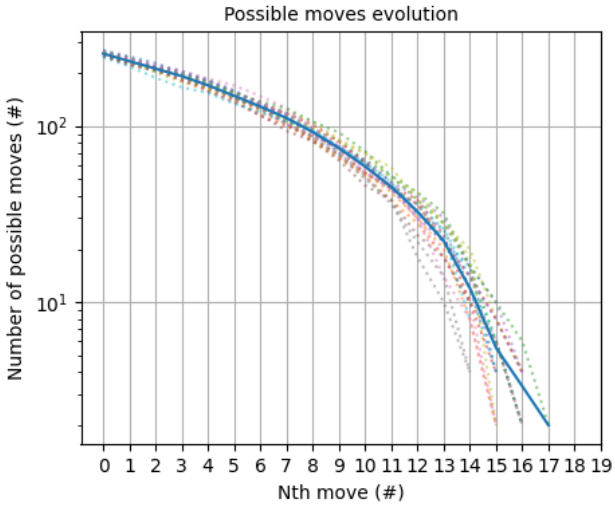


Fig. 1: Nombre de coups possibles vus par un joueur.

En vue de développer un agent le plus efficace possible, nous avons commencé par observer le déroulement d'une partie et avons tenté d'estimer un maximum de caractéristiques de l'arbre de recherche. Bien que certains aspects soient difficiles à obtenir de manière absolue, les expériences suivantes ont pour but de simplement "donner une idée" pour orienter la conception de notre agent.

Pour ce faire nous avons simulé plusieurs parties entre deux **agents aléatoires**.

*1) Estimation du facteur de branchement:* ou le nombre de coups possibles à chaque tour du jeu.

On remarque sur la Fig.1. que le facteur de branchement est extrêment élevé au début de la partie pour se réduire petit à petit. On en déduit facilement que le nombre de coups total d'une partie possède donc forcément une borne supérieure, autour de 34 coups, et que **toutes les séquences d'états sont finies**. Un agent explorant l'espace de recherche (Alpha-beta)

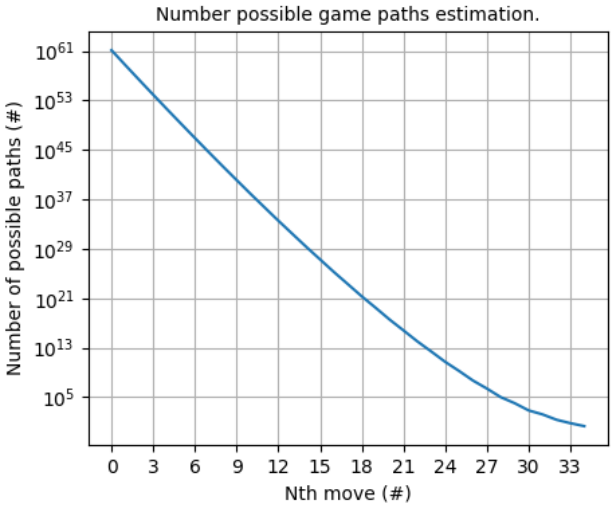


Fig. 2: Nombre de parties possibles à partir du n-ième coup.

sera donc très gourmand en ressource dès le début de la partie et simuler une proportion significative des parties entières (Monte-carlo) semble irréaliste comme le montre le graphe en Fig.2. [1]

[1]Tracé sous l'hypothèse d'un facteur de branchement constant à chaque profondeur de l'arbre. Cette hypothèse, bien qu'incorrecte, permet de donner une idée sur le nombre de parties possibles à partir d'un n-ième coup.

## IV. Alpha-beta et dérivés

Étant donné les observations précédentes, une implémentation d'un minimax/alpha-beta semblait la plus appropriée.

Pour pouvoir écrire un tel algorithme il est nécessaire de choisir une bonne fonction d'utilité $U(s)$ pour un état $s$. Différentes solutions ont été testées et sont décrites plus en détail ci-dessous.

### A. Tower-isolation player

Pour cet agent on choisit:

$U(s) = S(s) + \sum_i G_i(s) - \sum_j B_j(s)$

où $G_i(s)$ vaut 1 si la $i^{eme}$ tour appartenant à notre agent est isolée, 0 sinon et $B_j(s)$ vaut 1 si la $j^{eme}$ tour appartenant à l'adversaire est isolée, 0 sinon, et ce pour un état $s$.

L'heuristique développée ici consiste donc à essayer de maximiser le nombre de tours isolées qui nous sont favorables afin d'empêcher l'adversaire de s'en emparer tout en essayant de minimiser le nombre de tours isolées de l'adversaire.

Il est maintenant nécessaire de fixer la profondeur d'exploration de l'algorithme alpha-beta. Après de nombreuses simulations, nous avons constaté que les premiers coups joués influencent peu la suite de la partie. De plus le nombre d'actions possibles est assez élevé. Le cutoff est donc fixé à 2 dans un premier temps. Ensuite, lorsque l'on dépasse l'étape 24, on l'augmente à 4. Cela nous permet d'obtenir un bon compromis entre efficacité et temps d'exécution de notre agent. Cependant, les 15 minutes allouées ne sont pas totalement exploitées car chaque action est choisie de manière quasi instantanée.

Au niveau des performances, cet agent bat l'agent *Greedy* et l'agent *Alpha-beta* implémenté avec l'algorithme alpha-beta de base, et ce peu importe la configuration du match. En effet, nous avons remarqué que le joueur désigné pour commencer la partie disposait d'un léger avantage. De ce fait, lors d'un match entre deux agents *Tower-isolation*, celui jouant en second termine la partie par une défaite.

Cependant, cet algorithme va nous servir de base pour évaluer les suivants.

### B. Timed tower-isolation player

Pour exploiter de manière plus optimale les 15 minutes allouées, nous avons implémenté une autre version de notre agent *Tower-isolation* appelée *Timed tower-isolation*. Dans celle-ci, nous fixons la valeur du cutoff à 3, puis à 4 lorsque l'on dépasse l'étape 10 et finalement à 5 après l'étape 22. L'exécution de cet agent prend alors en moyenne 13 minutes 30 secondes sur nos ordinateurs mais peu varier d'une machine à l'autre et demander plus que 15 minutes. Grâce à l'augmentation de la valeur du cutoff, l'agent *Timed tower-isolation* bat *Tower-isolation*, et ce même en étant second joueur. On peut donc affirmer que la valeur du cutoff influence de manière non négligeable l'efficacité de l'algorithme alpha-beta. La figure 3 montre le nombre d'états évalués en fonction de la valeur du cutoff pour l'agent *Tower-isolation* et nous a aidé à déterminer les valeurs ci-dessus.

Il s'agit ici de notre meilleur agent. Cependant, étant donné la variabilité de son temps d'exécution, nous n'avons pas osé soumettre ce dernier pour le concours de peur d'obtenir un time out. Nous avons préféré soumettre l'agent *Gold* présenté ci-dessous afin de nous garantir une bonne place dans le concours.

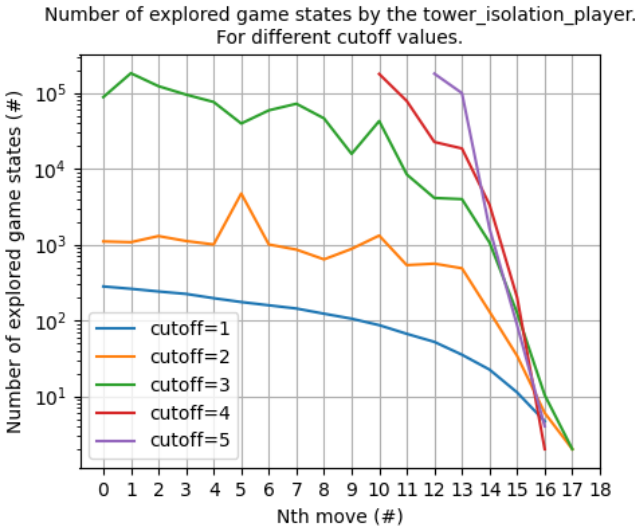


Fig. 3: Comparaison du nombre d'états explorés au n-ième coup par l'agent Tower isolation en fonction du cutoff.

### C. Gold player

Cet agent est une évolution de l'agent *Tower-isolation*. C'est celui qui a été soumis pour le concours. Il est dans le fichier *my_player.py*.

Au vu des performances de *tower_isolation*, il est possible que par chance nous ayons approximé assez correctement la fonction d'utilité réelle du plateau. Bien que cela soit difficilement vérifiable, supposons cette prémisse comme vraie.

Si l'utilité réelle du plateau est effectivement

$U(s) = S(s) + \sum_i G_i(s) - \sum_j B_j(s)$

lors d'une partie optimale, le second joueur n'a aucune chance de gagner et, comme au morpion, il ne peut espérer que faire un match nul. En cherchant un peu, on trouve une heuristique légèrement différente de la précédente qui permet d'obtenir un match nul lorsqu'on joue en second.

On la définit:

$U(s) = S(s) + \sum_i G_i^5(s) - \sum_j B_j^5(s)$

où $G_i^5(s)$ vaut 1 si la $i^{eme}$ tour appartenant à notre agent est isolée et a une hauteur de 5, 0 sinon et $B_j^5(s)$ vaut 1 si la $j^{eme}$ tour appartenant à l'adversaire est isolée et a une hauteur de 5, 0 sinon, et ce pour un état $s$.

En appliquant l'heuristique développée ici qui consiste à maximiser les tailles des tours de l'agent et minimiser celles de l'adversaire, l'agent *Gold* bat l'agent *Tower-isolation* s'il joue en premier et obtient une égalité sinon.

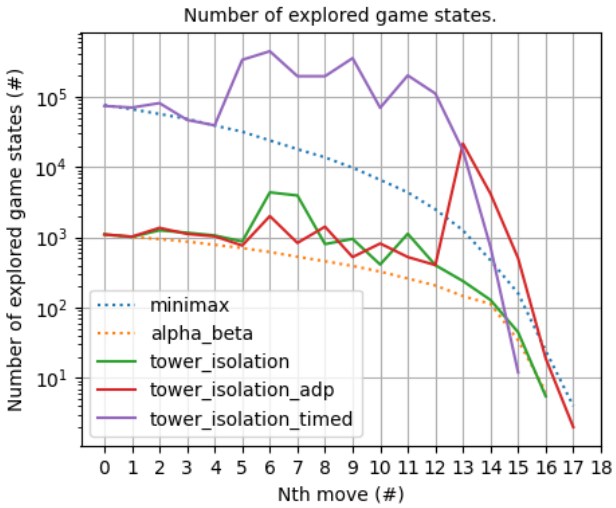


Fig. 4: Comparaison du nombre d'états explorés au n-ième coup.

### D. Comparaison des différents agents

La figure 4 nous permet de comparer le nombre d'états explorés à chaque étape pour les différents agents. Ces derniers ont tous été exécutés en tant que deuxième joueur. Tous les algorithmes ont une profondeur constante de 2 exception faite de *tower_isolation_timed* ainsi que *tower_isolation_adp* qui ont une profondeur qui s'adapte telle que décrite dans les sections ci-dessus. On remarque assez facilement l'efficacité du pruning alpha-beta par rapport à l'algorithme minimax. Il n'est pas étonnant que notre agent *Timed tower-isolation* est le plus performant étant donné qu'il explore et évalue un plus grand nombre d'états tôt dans la partie. Les figures 6 et 5 montrent l'évolution des scores lors des matchs entre nos différents agents.

## V. Pistes d'améliorations

### A. Un peu d'apprentissage machine ?

Il est clair que jusqu'ici, nous n'avons tenté d'approximer la fonction utilité d'un état que de manière empirique, à force d'observations et de parties jouées. Sans limitations de ressources tant temporelles que matérielles, il aurait été intéressant d'obtenir une approximation de cette fonction utilité à l'aide de modèles d'apprentissage machine.

### B. Pourquoi pas du Monte-Carlo ?

Comme expliqué dans la section III de ce rapport, la topologie de l'espace de recherche présageait un usage *efficace* de cette technique laborieux et éventuellement difficile à accomplir[2]. Il faut toutefois reconnaitre une certaine similarité entre Avalam et d'autres jeux de plateaux à facteur de branchement élevé (comme le Go) pour lesquels le Monte-Carlo se

[2]Quand simuler (certainement pas au début) et quoi simuler (une partie de plateau? le plateau entier?) sont déjà deux questions auxquelles trouver la réponse n'est pas forcément évident.

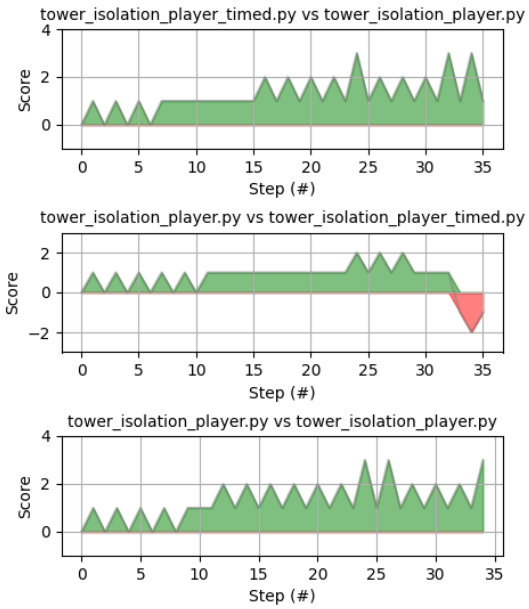


Fig. 5: Evolution du score lors de différents scénarios

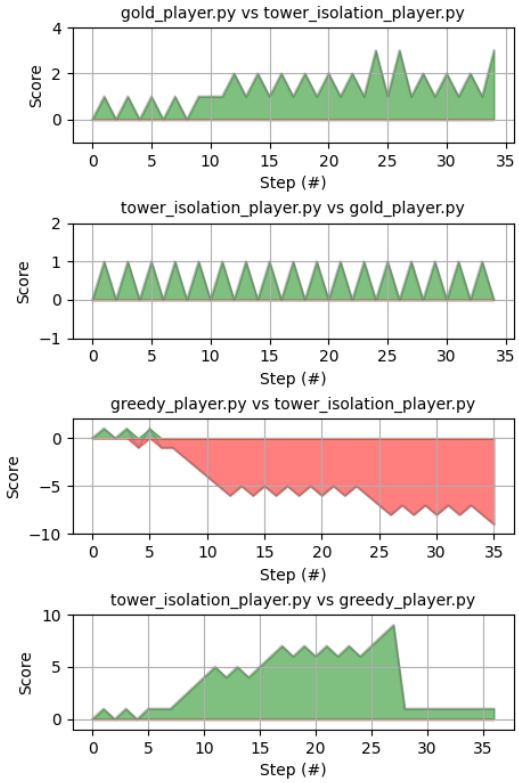


Fig. 6: Evolution du score lors de différents scénarios

révèle performant. C'est pourquoi cette piste d'amélioration mériterait d'être explorée.

## VI. Conclusion

Pour terminer, nous pouvons dire que, après plusieurs simulations afin de comprendre la mécanique du jeu, nous avons produit un agent *Timed tower-isolation* très efficace basé sur l'alpha-beta pruning ainsi qu'une heuristique qui s'appuie sur l'isolation de certaines tours. Cependant, étant donné les spécificités du matériel sur lequel il est exécuté, il est possible qu'il dépasse les 15 minutes allouées. C'est pourquoi nous avons créé un autre agent *Gold* qui lui s'appuie sur une heuristique se basant sur l'isolation des tours de taille maximale mais qui a une profondeur d'exploration et donc un temps de calcul moindre que l'agent précédent. Cela implique que ses performances, bien que satisfaisantes, sont légèrement en dessous de *Timed tower-isolation* car il n'obtient qu'une égalité contre *Tower-isolation* en étant second joueur.

## References

[1] Artificial Intelligence: A Modern Approach, 4th Edition. Retrieved 2020-05-1