AGENT INTELLIGENT POUR LE JEU ABALONE

W. KEDDACHE & S. LABASSI

E–mail: *samy.labassi@gmail.com, wassim.t.keddache@gmail.com*

Résumé: Ce rapport explore la conception d'un agent intelligent spécialisé dans une version adaptée du jeu Abalone. Joué sur un plateau hexagonal de 61 cellules, ce jeu exige que les joueurs éjectent un maximum de pièces adverses ou se rapprochent du centre pour remporter la victoire. Pour cette tâche, nous avons implémenté un algorithme de type Minimax, permettant à l'agent de prendre des décisions stratégiques. Les exigences du projet imposent des contraintes rigoureuses d'implémentation en Python avec la bibliothèque SeaHorse, incluant un temps de jeu limité à 15 minutes par partie et une utilisation mémoire ne dépassant pas 4 Go.

1. MÉTHODOLOGIE

Les principes de recherche adversariale de base sont des outils simples mais efficaces afin de concevoir des agents intelligents pouvant parfois rivaliser avec des joueurs humains. L'agent implémenté dans le cadre de ce projet fait usage de quelques notions afin de prendre des décisions rapides et judicieuses dans le contexte du jeu de table Abalone.

1.1. Algorithme MinMax

La composante principale dans la prise de décision de l'agent est l'implémentation d'un algorithme MinMax pouvant déterminer quelle action est la plus avantageuse. Le principe de base est simple: dans un situation où deux joueurs essaient d'obtenir un avantage et éventuellement gagner la partie, chaque joueur tente de maximiser son score (joueur Max) et de minimiser le score de l'adversaire (joueur Min). Ainsi, en se faisant présenter un état de jeu, l'agent détermine en premier lieu toutes les actions possibles. Dans notre cas il s'agissait des états après que le joueur ait effectué un action, le tour de ce dernier étant à l'adversaire qui essaie de minimiser le score. Ceci peut être représenté par un arbre de décision d'au moins trois niveaux afin d'anticiper au moins un coup de l'adversaire. Arrivé à un état terminal ou à la profondeur de recherche maximale, l'agent calcule le score de chaque état en fonction de certaines heuristiques. Ensuite il va donc sélectionner l'état ayant le plus haut score parmi tous les états minimum retourné par chaque état où le tour est à l'adversaire. La complexité temporelle de cet algorithme est $O(b\hat{m})$ et présente sa plus flagrante limitation. En effet, il est évidemment plus avantageux d'explorer le plus d'état possible en augmentant la profondeur de recherche, mais en raison des contraintes de temps, nous l'avons limité à deux ou quatre dans le meilleur des cas.

1.2. Alpha-Beta Pruning

L'algorithme MinMax à son plus simple ne tient pas compte de ce qui advient lorsqu'il rencontre un état minimum qui mène à un état maximum ayant un score plus petit qu'une des valeurs retournée par les états minimums déjà visités. En effet, en supposant que l'adversaire est rationnel, il est donc inutile de poursuivre la visite de cet état du joueur Min étant donné que nous avons déjà trouvé une meilleure action à prendre. On peut donc procéder à l'élagage de plusieurs états réduisant ainsi le nombre d'opérations nécessaires et permettant d'utiliser une plus grande profondeur de calcul.

1.3. Heuristiques

Le score de chaque état terminal dans l'arbre représentant l'algorithme MinMax est calculé en fonction des informations tirées de l'état du jeu. Ainsi, il est important de déterminer les situations avantageuses et les actions les plus favorables. Après avoir expérimenter en jouant plusieurs partie et avoir effectué des recherches sur les agents existants, voici les concepts que nous avons retenus:

1. **Distance des pièces par rapport au centre**

   Une stratégie très commune en début de partie est l'attaque rapide du centre du plateau. En effet, il s'agit d'une position stratégique permettant d'éviter de se faire attaquer et de menacer l'adversaire de plusieurs directions. Il y a plusieurs façons de modéliser cette distance pour une disposition de pièce donnée. Nous avons en premier lieu tenté de calculer le centre de gravité discret des pièces, mais ceci représentait les états où les pièces étaient équidistantes au centre comme avantage, et ce, même si elles en sont éloignées. Ainsi, nous avons plutôt opté pour le calcul de la somme des carrés des distances de chaque pièce. Ce modèle permet de punir davantage les pièces les plus éloignées permettant ainsi de se diriger rapidement vers le centre. (Özcan, 2004) (Friedman, n.d.)

2. **Proximité des pièces entre elles**

   La manière dont les pièces sont réparties sur le plateau a aussi une grande influence sur

l'évaluation de l'état de jeu. Plus les pièces sont regroupées, plus il est facile de pousser les pièces de l'adversaire et de protéger les siennes en les empêchant de se déplacer vers les extrémités du plateau. Une façon simple et efficace de modéliser ce concept est la moyenne de la distance de chaque pièce avec toutes les autres pièces de la même couleur. (Friedman, n.d.)

3. **Sous-groupe de pièces**

   Pour complémenter l'adjacence des pièces, il est aussi intéressant de connaître le nombre de sous-groupes. Ceci permet de consolider la proximité des pièces en évitant certaines pièces se détachent même si cela améliore le résultat de la proximité des pièces. Pour ce faire, il est nécessaire de modéliser les pièces en graphes non dirigées et simplement attribuer le nombre de graphes au score de l'heuristique. (Friedman, n.d.)

4. **Pièces menacées**

   Les premières heuristiques sont intrinsèquement défensives et ne permettent pas de marquer des points nécessaires à cette situation de jeu à somme nulle. Comme dans le cas d'autres jeux de stratégie tels les échecs il est nécessaire d'avoir conscience des pièces adverses en danger d'être perdues. Ainsi, les états de jeux présentant le plus de pièces menacées possibles sont avantagés ce qui empêche de se rediriger vers le centre du plateau alors que l'agent aura l'opportunité de prendre un pièce dans un coup subséquent. Dans le cas du jeu abalone il s'agit des pièces sur les bords du plateau étant en désavantage numérique dans un direction opposée à un coin vers lequel elles pourraient être poussées. (Lee, n.d.)

5. **Pointage**

   Une partie d'Abalone se remporte en éliminant des pièces adverses dans le temps alloué à chaque joueur. Ainsi, le pointage de la partie est le meilleur indicateur de la performance de l'agent ainsi que des états avantageant le plus. Il faut donc comparer la différence de score entre l'état courant de jeu et les états obtenus en évaluant toutes les actions possibles. L'agent produit présente un comportement glouton dans l'implémentation de cette heuristique. En effet, dès qu'il trouve un nœud final dans l'arbre de recherche où le delta de score est non-null, il lui attribue un score dépassant la borne supérieure de la normalisation. Ainsi, les mouvements capturant une pièce sont toujours sélectionnés. Cependant, le résultat favorise les plus grands delta afin d'accommoder les différentes profondeurs de recherche. (Friedman, n.d.)

1.4. Calcul de score

Chaque heuristique est ensuite utilisée afin de calculer le score total représentant l'état à évaluer. Afin de pouvoir les comparer sur la même base, leurs résultats sont normalisés entre zéro et un en fonction des extremums de valeurs. Ces derniers ont été obtenus empiriquement à travers quelques cycles de parties entre les agents. Ces extremums sont plus représentatif de la réalité du jeu que les valeurs théoriques de maximum et de minimum et permettent ainsi de normaliser plus précisément les résultats des heuristiques. À travers nos tests, nous avons remarqué que ces valeurs ont un énorme impact sur les décisions prises. En effet, un maximum empirique trop bas coupe la borne supérieure de l'intervalle de résultats heuristiques valides, limitant ainsi les états où, par exemple, toutes les pièces sont éloignées du centre dans le cas de l'heuristique évaluant cette distance. de plus, puisque le mouvements ne changent pas significativement les états du plateau, le calcul des heuristique est davantage sensible aux petits changement. Ainsi, si des états voisins dépassent la valeur du maximum fixée pour une heuristique données, il seront évalués à la même valeur de 1, et ce même si un des états est légèrement plus avantageux. Par conséquent, un enjeux d'équilibre entre les maximum empiriques et théoriques entre en jeu.

L'évaluation du résultat de chaque heuristique dépend aussi de leurs objectifs. En effet, les trois premières heuristiques sont défensives; leur valeurs sont à minimiser et seront négatives, tandis que le les deux autres sont offensives et à maximiser. Le calcul du score fait aussi l'usage de poids attribué à chaque heuristique pour déterminer le comportement de l'agent. Ainsi, la valeur obtenue pour chaque état est entre -1 et 1.

1.5. Optimisations

Afin de tirer un maximum de performance de l'agent et palier à la complexité temporelle de l'algorithme de recherche ainsi que le calcul des heuristiques qui est parfois lourd, quelques optimisations ont été mises en place. Tout d'abord, le générateur de coups fourni avec le jeu renvoi même les états où un des joueurs pousse sa propre pièce hors du plateau. Il faut donc passer par une étape de sélection des états valide puisqu'on suppose que les agents sont rationnels et cherchent tous deux un comportement optimal. Il serait aussi intéressant d'ordonner les états en fonction du nombre de pièces déplacées mais il s'agit d'une opération coûteuse qui ne vaut probablement pas son temps d'opération. Ensuite, l'implémentation d'une table de transposition est un bon ajout afin de réduire la complexité temporelle de la recherche au profit de la mémoire disponible.

1.6. Contraintes

L'agent doit respecter une contrainte temporelle lors de son fonctionnement et à un nombre de mouvements limité. Ainsi, la profondeur de la recherche ne peut pas être trop élevée pour ne pas consommer tout le temps alloué que pour les quelques premières actions. Considérant les contraintes et distribuant le temps également entre les coups, chaque action doit prendre au maximum 15 secondes. De plus, la profondeur doit être paire afin de tomber dans un état du joueur Max et que le calcul de maximisation de score soit cohérent. Avec une profondeur de deux niveaux, on constate que le temps de calcul ne dépasse jamais le temps alloué par tour.

1.7. Entrainement

Les poids sont cruciaux au comportement de l'agent. De ce fait, un entraînement rigoureux est nécessaire afin de trouver la meilleure combinaison. Ce problème se porte très bien aux concepts de recherche locale et d'optimisation. Cependant, pour s'assurer de la diversité des candidats proposés par la fonction de voisinage une méta-heuristique implémentant un algorithme génétique s'est avérée être satisfaisante. Pour ce faire, une première génération aléatoire respectant la contrainte stipulant que la somme des poids doit être exactement 1 est générée et m agents sont produits. Ensuite, ces agents jouent entre eux à raison de 3 parties par paire afin de récolter assez d'information pour leur évaluation. Les résultats sont ordonnés par pourcentage de victoire et les n meilleurs agents sont sélectionnés. Ensuite, une prochaine génération est produite en suivant 4 étapes. (MathWorks, 2023)

1. **Sélection des meilleurs agents** Parmi les agents de la génération précédente, les n meilleurs agents sont sélectionnés et seront utilisés pour les prochaines étapes de génération.

2. **Combinaison de 2 agents** Ces agents sont ensuite combinés pour obtenir de nouvelles collections de poids. prenant pour point de départ les agents sélectionnés précédemment le poids des heuristiques de n nouveaux agents a une probabilité de 50% d'être remplacé par le poids provenant d'un autre agents. Il est ainsi possible de diversifier les générations et déterminer quelle valeur de poids est la meilleure pour une certaine heuristique. Il est important de noter que même après une combinaison la contrainte de somme des poids doit toujours être respectée et un étape de normalisation est nécessaire.

3. **Mutation des agents** Afin de s'assurer de ne pas tourner en rond avec les mêmes poids que ceux obtenus après la première génération, il est important de créer n agents dont les valeurs de poids sont différentes. Il faut donc aléatoirement modifier les agents obtenus à l'étape 1 afin de diversifier le pool génétique.

4. **Combinaison de 3 agents** Cette étape n'est pas typique d'un algorithme génétique, mais il s'agit d'une façon de davantage diversifier les agents en répétant l'étape deux avec trois agents au lieu de deux.

2. RÉSULTAT ET ÉVOLUTION DE L'AGENT

Après plusieurs entraînements, on a pu remarquer une convergence des scores vers un poids assez commun entre les agents. Tout d'abord, on peut remarquer que l'agent retenu, qui avait l'identifiant 15, avait un poids assez important dans le centre. En effet, le poids de l'heuristique pour le centre était à 44% du score, comme on peut le voir en annexe. Il était le poids le plus élevé de toutes les agents retenus pour cette heuristique. Le score aussi était assez élevé avec un poids de 25% dans le calcul du score. On peut comprendre aussi que l'heuristique cluster_score ainsi que threatened_piece_score ont un poids plus petit mais assez significatif. Cependant, l'heuristique analysant l'adjacence des pièces a un poids très peu significatif, ce qui peut être expliqué par le fait que les autres heuristiques prennent en compte cette aspect.

3. DISCUSSION

L'agent est passé à travers une multitude de changements peaufinant sa prise de décision et de décisions prises afin de personnaliser son comportement.

3.1. Avantages

L'agent retenu offre un bon équilibre entre son comportement offensif et défensif. En effet, on observe que les pièces se dirigent rapidement vers le centre et conservent leur position malgré qu'elles se fassent pousser puisqu'elles ont tendance à rester groupées. De plus, l'agent prend toute opportunité de capturer une pièce adverse mais reste au centre sinon, ce qui lui permet de gagner en cas d'égalité. D'après nous et en observant les résultats, les heuristiques représentent bien les états avantageux tout en pénalisant ceux à éviter.En outre, l'agent retenu est le fruit d'un entraînement permettant de s'assurer de sa qualité. L'algorithme génétique utilisé est donc un grand avantage par rapport à un ensemble de poids sélectionné manuellement.

3.2. Limites

Bien que l'agent soit performant face à ses autres versions et des agents ayant des heuristiques différentes, il pourrait tirer avantage d'une profondeur de recherche plus grande. Cependant, étant donné les

Table 1: Poids des heuristiques pour les meilleurs agents retenus

ID	WINRATE	compactness _score_param	cluster _score_param	threatened _piece_score_param	in_game _score_param	center_mass _score_param
15	75,00%	3,14%	12,64%	15,18%	25,00%	44,03%
8	66,67%	6,86%	13,38%	19,90%	34,16%	25,70%
22	66,67%	2,96%	14,67%	17,62%	30,49%	34,27%
5	64,58%	4,24%	17,05%	20,47%	18,41%	39,83%
18	64,58%	3,96%	15,93%	8,65%	31,50%	39,95%
20	64,58%	7,87%	15,30%	20,12%	18,32%	38,39%
11	62,50%	8,01%	15,57%	18,70%	18,65%	39,06%

contraintes de temps, ceci s'avérait être un défi de conserver la latitude de recherche, c'est-à-dire la nombre de nœuds finaux explorés. Cependant, les tests effectués sont assez satisfaisants pour justifier de conserver cette profondeur. Aussi, l'agent est très adapté à l'environnement dans lequel il à été testé, donc une partie classique d'Abalone. Il est donc peu flexible aux changements de configuration. En effet, ceci est grandement dû aux extremums obtenus de façon empirique. Si l'agent est utilisé pour un plateau ou toutes ses pièces sont séparées, il ne sera pas en mesure de bien évaluer sa distance par rapport au centre ou le nombre de sous-graphes puisque les valeurs des maximums seraient inférieur à celle de l'état courant. Ainsi, tous les états le dépasseraient et auraient une valeur heuristique bornée.

4. RÉFÉRENCES

1 Ö. Özcan, E., & Hulagu, B. (2004). A Simple Intelligent Agent for Playing Abalone Game: ABLA. [Rapport technique]. https://www.researchgate.net/publication/249703658_A_Simple_Intelligent_Agent_for_Playing_Abalone_Game_ABLA

2 S. Friedman, & B. Ibarra. (n.d.). Abalone [Rapport technique]. https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/JET/report_abalone.pdf

3 B. Lee, & H. Joo Noh. (n.d.). Abalone –Final Project Report [Rapport technique]. https://www.cs.cornell.edu/~hn57/pdf/AbaloneFinalReport.pdf

4 MathWorks. (2023). What Is the Genetic Algorithm? MathWorks. https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html