AWELE : RABBI JACBOT

RAPPORT DE PROJET – INTELLIGENCE ARTIFICIELLE

Clément LAUER – Bastien PAZZAGLIA

Table des matières

1. Introduction

L’objet de ce projet est de créer une intelligence artificielle capable de jouer à un jeu de stratégie connu sous le nom de l’Awélé, c’est-à-dire de savoir jouer le meilleur coup possible pour chaque situation donnée.

2. Le jeu de l’Awélé

L'Awélé est un jeu de société originaire d'Afrique de l'Ouest, qui se joue généralement à deux joueurs. Il nécessite un plateau de jeu avec deux rangées de six trous (pour un total de 12 trous) et deux grandes réserves de chaque côté (appelées "maisons"), ainsi que de 48 graines (4x12) qui sont placés dans les trous au début du jeu.

Le but du jeu est de capturer le plus grand nombre de graines possible. Le joueur qui capture le plus grand nombre de graines à la fin de la partie est déclaré vainqueur.

Au début de chaque tour, le joueur choisit l'un des six trous de sa rangée et prend toutes les graines de ce trou. Il distribue ensuite ces graines dans les trous de sa rangée et dans la rangée de l'adversaire dans le sens inverse des aiguilles d'une montre. Il saute le trou de départ.

Si le dernier grain semé se trouve dans une case de l’adversaire qui contenait 1 ou 2 graines, alors il ramasse ses graines et les ajoute à son score, ainsi que les graines de chaque trou précédent consécutif de l’adversaire de 2 ou 3 graines.

Il est interdit d’affamer son adversaire, c’est-à-dire de jouer un coup qui ne laisserait aucune graine à l’adversaire, s’il est possible d’en jouer un autre.

Enfin, le jeu s’arrête si le plateau contient 6 graines ou moins.

3. Méthodologie

Pour mieux appréhender ce projet, nous nous sommes renseignés sur le jeu, ses règles et son fonctionnement (Simon, s.d.) (Bennalal, s.d.). Par la suite, nous avons exploré les différentes stratégies. Pour ce faire nous avons commencé par jouer ensemble sur des jeux d'awélé en ligne (Awélé en ligne, s.d.), puis nous avons consulté différents sites qui nous ont donnés des stratégies basiques (Retschitzki, s.d.), notamment en ce qui concerne les stratégies liées aux Krous (ou greniers). Un Krou est une case contenant un grand nombre de graines, afin de réaliser une semence en le jouant (cela permet de déposer des graines dans un maximum de graines afin de les récupérer au tour suivant, si possible avec un autre Krou).

Nous avons aussi consulté les travaux de professeurs en algorithmie qui se sont intéressés au jeu de l'awélé dans leurs cours (Simon, s.d.) (Richer, s.d.).

Nous en avons conclu plusieurs principes que nous décrirons par la suite.

Pour ce qui est du développement des bots, à chaque modification ou ajout majeur dans le code, nous avons créé un bot différent contenant ces modifications tout en gardant le travail précédent, afin de vérifier la pertinence de la piste suivie.

Lorsque nous avions assez de bots, nous lancions un tournoi et nous ne gardions que les meilleurs bots.

4. Pistes explorées

4.1. Constats de départ

N'ayant qu'une heure d'apprentissage, ce qui nous semblait insuffisant pour baser notre IA dessus tout en sachant qu'il n'était pas autorisé d'importer des données de parties précédentes lors du tournoi d'évaluation. Nous pensions qu'une IA entièrement basée sur l'apprentissage ne serait pas assez efficace.

Nous nous sommes donc orientés vers un algorithme MinMax qui peut être efficace dès le départ. De plus dans les rapports qui nous ont été fournis, la majorité d'entre eux utilisait un algorithme de type MinMax. Nous avons donc expérimenté plusieurs versions de MinMax c'est-à-dire un MinMax simple, un MinMax avec élagage Alpha-Bêta et un NegaMax (simplification du MinMax Alpha-Bêta) que nous avons fait s'affronter lors d'un tournoi avec le MinMax Alpha Bêta fourni avec le projet. A la suite de ce tournoi c'est algorithme fourni qui semblait le plus efficace et de très loin. Nous sommes donc partis sur cette base pour notre IA, il ne nous restait maintenant qu'à trouver un moyen de l'améliorer.

La première idée qui nous est venue est d'augmenter la profondeur du MinMax ce qui nous permettrait d'avoir un meilleur horizon sur l'état de la partie. Deuxièmement, nous voulions trouver une meilleure heuristique ce qui nous permettrait d'obtenir un meilleur élagage, d'augmenter encore la profondeur et d'affiner le tri des meilleurs coups possibles.

4.2. Travail sur l'optimisation

Lorsque nous avons commencé à modifier la profondeur nous nous sommes confrontés aux problèmes de temps de décision et de gestion de la mémoire. Il était donc essentiel d'optimiser au maximum notre code pour pouvoir augmenter la profondeur.

4.2.1. Typage des données

Pour commencer nous avons cherché à alléger le typage notamment en mettant des bytes lorsque cela était possible. Puis nous avons eu comme seconde idée de sauvegarder les coups rencontrés et leur évaluation pour ne pas avoir à les calculer de nouveau et donc à baisser le temps de décision en dépit de la mémoire. Ceci a été uniquement possible grâce à la simplification du code, des classes et de l'allègement du typage effectuée auparavant.

En ce qui concerne le typage il y avait certaines limitations comme le score qui devait être compris entre -127 et 127, ainsi qu'une impossibilité d'utiliser des coefficients flottants pour définir notre heuristique.

4.2.2. Stockage des coups en mémoire

Pour la sauvegarde, nous avons eu quelques difficultés. En effet, le nombre de coups étant exponentiel à chaque profondeur parcourue, les sauvegardes auraient rapidement surchargé la mémoire. Nous avons donc choisi de limiter le nombre de coups sauvegardés à 70, autrement nous dépassions largement le temps de décision autorisé. Mais malgré cela, cela ne fonctionnait pas car l'évaluation de chaque coup sauvegardé ainsi que celle de ses descendants devait être recalculée. Nous avons fini par trouver une solution simple, qui consiste à aussi sauvegarder la profondeur associée au coup. Cela nous permet de prendre en considération une éventuelle réévaluation du coup.

4.2.3. Améliorations possibles

Ainsi, si notre IA est fonctionnelle, il subsiste des pistes d'améliorations que nous n'aurons pas eu le temps de suivre, comme réduire encore la taille des données sauvegardées pour augmenter le nombre de coups en mémoire, ou encore ajouter un apprentissage avant le tournoi, afin de commencer les parties avec des coups déjà en mémoire.

Cela nous permettrait d'accroître encore la profondeur de l'algorithme, car au lieu d'augmenter le temps de décision, on augmenterait l'espace mémoire.

4.3. Travail sur la profondeur

4.3.1. Observations

Comme dit précédemment, la profondeur a été l'axe principal de nos recherches, puisque pour nous, la meilleure façon d'améliorer un algorithme MinMax, quel que soit l'heuristique, est d'augmenter la profondeur, pour qu'il puisse voir plus loin que son adversaire et prendre le match à son avantage.

D'après nos observations, augmenter la profondeur demande beaucoup de ressources (mémoire, temps de décision). Après plusieurs tests, nous nous sommes rendus compte qu'avoir une certaine profondeur selon l'état de la partie n'a pas le même impact. En effet, nous avons observé que la profondeur en début de partie est plus négligeable, alors que plus on avance dans la partie, plus les possibilités sont nombreuses, et plus la taille de la profondeur a d'impact.

4.3.2. Profondeur dynamique

Nous nous sommes ainsi orientés vers une profondeur dynamique, qui augmente au fur et à mesure de la partie, c'est-à-dire selon le nombre de graines encore en jeu.

Profondeur = Profondeur_de_base + 48 / nombre_de_graines_encore_en_jeu

La profondeur de base correspond à la profondeur en début de partie, dans notre cas 8, 48 correspond au nombre de graines en début de partie.

Cette formule nous a été inspiré par le rapport de Mathis Saillot et de son IA « J'ai Awélé de Travers » (Saillot, 2020).

4.4. Travail sur l'heuristique

Etant donné que notre travail repose sur le MinMax fourni dans le projet, l'objectif ici est de trouver une heuristique plus efficace que celle de base. Elle permettrait d'effectuer un meilleur élagage, de trouver les meilleurs coups le plus rapidement possible.

Nous nous sommes inspirés des rapports des années précédentes, ainsi que de nos recherches Internet et de nos parties en ligne.

4.4.1. Tests de différentes heuristiques

Nous avons testé plusieurs heuristiques sur plusieurs bots :

- Albatar : Le niveau de danger (nombre de cases à moins de 3 graines).
- HoulaHoula : Le nombre de Krous (cases avec plus de 11 graines) et leur position dans la rangée. Plus le Krou est proche de l'adversaire, plus il est important.
- Saligaud : Un mélange des heuristiques précédentes et de celle de base (score du joueur – score de l'adversaire).

Les heuristiques simples ne battent pas celle de base, seul le Saligaud semble être une piste intéressante, car elle prend plus de paramètres en compte. Néanmoins, chaque partie de cette heuristique a une certaine importance qui doit être déterminée par des coefficients.

Pour déterminer ces coefficients, en ayant suivi le rapport de Mathis Saillot (Saillot, 2020), nous avons tenté de les récupérer via un algorithme génétique. Cette piste a été abandonnée à cause du temps de calcul nécessaire à son fonctionnement. De plus nous avons testé une variante de son heuristique, qui fonctionne mieux que la nôtre. Cependant, pour fonctionner pleinement, celle-ci nécessite un classement de coups dont nous n'avons pas réussi à faire fonctionner de manière optimale, après de nombreux tests.

Enfin, notre optimisation par simplification des données (bytes) ne permet pas d'utiliser des coefficients flottants. Car en supprimant cette optimisation, cela accroît fortement le temps de décision et la mémoire utilisée.

4.4.2. Heuristique finale

Ainsi, nous sommes restés sur l'heuristique Saligaud, en la décomposant selon les différentes heuristiques qu'elle contient, ce qui nous permet de les utiliser au bon moment. Lorsque nous sommes sur un coup fatal, il est inutile de calculer les Krous et le niveau de danger, seul le score suffit.

Concernant le reste des coups, nous prenons le score, auquel nous ajoutons le nombre de cases en danger, et le fait d'avoir des Krous ou non. Contrairement à Saligaud, on a choisi de simplifier l'heuristique, au lieu de compter le nombre de Krous, on ne fait que vérifier leur présence.

5. Choix final : Rabbi Jacbot

Notre IA finale, Rabbi Jacbot, est donc un algorithme de type MinMax avec élagage Alpha-Bêta, qui stocke les coups et leur évaluation en mémoire pour éviter de les réévaluer, et avec une heuristique dynamique, prenant en compte le type de coup (coup fatal, ou autre), la différence de score, le niveau de danger et la présence de Krous.

6. Conclusion

Nous avons beaucoup apprécié ce projet, qui était particulièrement intéressant car il s'approchait plus d'un sujet de recherche avec application qu'autre chose, nous nous nommes donc amusés à explorer de nombreuses pistes avec plus ou moins de succès. Même s'il y avait encore matière à améliorer notre bot, nous sommes assez contents de notre proposition.

Comme améliorations, nous avons pensé, comme dit précédemment, à un apprentissage avec tournoi pour stocker des coups avant le début du tournoi principal, simplifier encore les données et le code pour pouvoir améliorer le temps de décision et la mémoire. La mémoire sauvée pourra être réutilisée pour augmenter le nombre d'états sauvegardés et améliorer encore la profondeur de base. Une dernière amélioration serait de posséder une bibliothèque d'ouvertures pour éviter des calculs inutiles en début de partie.

7. Bibliographie

[1] *Awélé en ligne*. (s.d.). Récupéré sur https://www.playok.com/fr/awale/

[2] Bennalal, M. (s.d.). *Application de l'algorithme MINIMAX sur un jeu du MANCALA*. Récupéré sur https://www.youtube.com/watch?v=wRZf4H7FBRs

[3] Retschitzki, J. (s.d.). *Awélé - Les Stratégies*. Récupéré sur https://www.myriad-online.com/resources/docs/awale/francais/strategy.htm

[4] Richer, J.-M. (s.d.). *IA et Jeux - Méthodes de Résolution*. Récupéré sur https://leria-info.univ-angers.fr/~jeanmichel.richer/ensm1i_ia_jeux.php

[5] Saillot, M. (2020). *J'ai Awélé de Travers.*

[6] Simon, L. (s.d.). *Intelligence Artificielle - Cours sur les Jeux Avancés*. Récupéré sur https://www.electronique-mixte.fr/wp-content/uploads/2018/09/Cours-Intelligence-artificielle-47.pdf