UNIVERSITÉ
DE LORRAINE

Rapport Projet IA

Le jeu de l'Awélé

Teluob Bagarre

Réalisé par :
LALLEMENT Théo & MASKIO Alexis

2020-2021

Sommaire

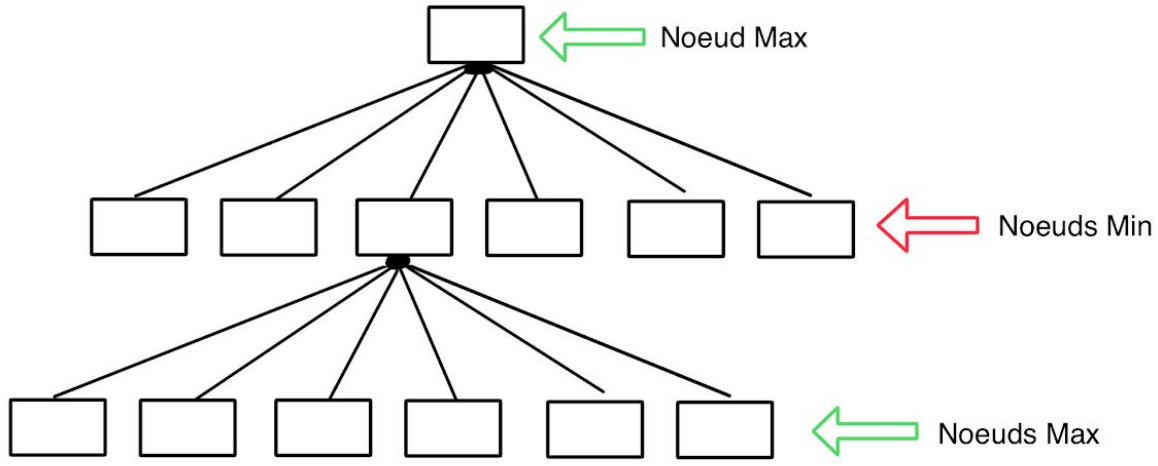
I - Introduction

Dans le cadre du cours d'intelligence artificielle du Master 1 Informatique, il nous a été demandé de réaliser un algorithme le plus performant possible pour simuler un joueur d'Awélé. Ce bot devra participer à un tournoi qui affrontera tous les bots des autres groupes. Il y a des contraintes à respecter dans la création de ce bot, comme par exemple celles liées au temps de prise de décision, ou de mémoire (la décision prise par le bot ne doit pas dépasser X ms, ou Y Mb).
Dans ce jeu, deux joueurs s'affrontent sur un plateau avec 12 trous (6 appartenant à chaque joueur) contenant des graines. Le but est de ramasser le plus de graines possibles. Le joueur ayant le plus de graines à la fin de la partie gagne.
Ce qui est intéressant dans ce jeu, c'est qu'il y a énormément de combinaisons, soit 889 063 398 406 coups possibles. Il nous est donc impossible de répertorier toutes les combinaisons possibles, d'une part à cause des contraintes de temps liées au projet, et d'autre part car nous n'avons pas le matériel nécessaire.

II - MinMax

Principe :

Pour la réalisation de ce bot, nous avons choisi de nous baser sur l'algorithme MinMax fourni avec le projet. Il s'applique à des jeux combinatoires à informations complètes. Cet algorithme est une très bonne solution pour ce type de jeu, si ce n'est le meilleur puisqu'il permet de parcourir toutes les possibilités de jeu à partir d'une situation donnée.
Le but de cet algorithme appliqué à notre bot, est de maximiser les gains tout en minimisant les gains de l'adversaire. Au moment où le botdevra jouer, celui-ci va parcourir ses 6 coups en profondeur en maximisant la différence des gains du joueur et celui de son adversaire.
Cet algorithme peut être représenté par un arbre contenant plusieurs nœuds. Le premier nœud sera un nœud Max, car il 'agit du coup que l'on doit jouer. On alternera entre les nœuds Max et les nœuds Min chaque fois qu'on descendra en profondeur.



III - AlphaBeta

Principe :

L'algorithme AlphaBeta est basé sur l'algorithme MinMax. Celui-ci permet de réduire le nombre de nœuds à parcourir en réalisant des coupes.
Dans cet algorithme, deux valeurs sont implémentées, soit les valeurs α et β. α représente le score minimum que peut obtenir le joueur qui souhaite maximiser ses gains (c'est à dire notre bot), et β représente le score maximum que peut obtenir le joueur qui souhaite minimiser ses gains (c'est-à-dire le joueur adverse). Dès lors que le score maximum (soit β) du joueur Min devient supérieur au score minimum (soit α) du joueur Max, alors il n'est plus nécessaire de parcourir le nœud en question puisqu'on ne pourra pas améliorer le score du joueur Max.

IV - Tri des coups

But de l'ajout de cette fonctionnalité :

Pour ce qui est de l'idée de trier les coups, nous nous sommes basés sur les rapports des années précédentes et notamment sur celui de l'année dernière.
Nous avons donc ajouté cette fonctionnalité de tri des coups à l'algorithme d'AlphaBeta.
Le but est de trier les coups dans un certain ordre afin de permettre à l'algorithme AlphaBeta d'éviter un maximum de nœuds à parcourir ce qui le rendra plus rapide. Si la prise de décision est moins longue, il nous sera donc possible d'augmenter la profondeur à parcourir pour l'AlphaBeta ce qui nous permettra ainsi d'avoir de meilleurs résultats tout en respectant les contraintes de temps imposées pour le tournoi.

Fonctionnement :

Pour trier les coups, nous avons décidé d'affecter un score à chaque type de coup. Nous sommes partis du principe qu'un coup était associé au trou duquel on part, à son nombre de graines et au nombre de graines sur le trou auquel on arrive.
Chaque coup possédera donc un id unique qui nous permettra de l'identifier, que l'on calcule à l'aide de la formule suivante :

Coup = CaseDeDépart + 6 x NombreDeGraines(CaseDeDépart) + 6 x 48 x NombreDeGraines(CaseArrivée)

Exemple : Un joueur joue la deuxième case de son plateau, qui contient 3 graines, et tombe sur la cinquième case de son plateau qui contient 3 graines. L'id du coup sera donc 884.
Le but d'identifier des coups est de voir s'il s'agit d'un bon ou mauvais coup. Dans le cas où il s'agit d'un bon coup, on ajoute un score à ce type de coup, dans l'autre cas on lui retire du score.
De base l'algorithme AlphaBeta, parcours les coups en partant de la gauche vers la droite. Il peut donc potentiellement parcourir tous les coups et ne pas faire de coupes, ce qui revient à faire un algorithme MinMax classique. En triant les coups en fonction de leur score, l'AlphaBeta parcourra les coups ayant le plus de chances d'être de bons coups, ce qui augmentera les chances de faire une coupe, et ainsi de ne pas parcourir tous les coups.

Résultats :

Sur l'image n°1, on peut voir le temps de décision pour un algorithme classique AlphaBeta en profondeur 10. On constate que cette valeur est supérieure au temps de 100ms maximum imposé par le sujet.
Sur l'image n°2, on peut avoir un algorithme AlphaBeta avec tri des coups en profondeur, qui ne dépasse pas la limite maximum des 100 ms.
Nous avons donc un algorithme plus rapide qui permet ainsi de le rendre plus performant en augmentant la profondeur à parcourir.

```
Bot 8/8
Classe : awele.bot.AlphaBeta.AlphaBetaBot
Instanciation du bot "AlphaBeta sans Tri"
Nom du bot : AlphaBeta sans Tri
Auteur(s) : Théo Lallement
Temps d'apprentissage : 00:00:00.000
Durée d'une prise de décision : 00:00:00.236
Bot "AlphaBeta sans Tri" disqualifié : durée d'une prise de décision trop long
```

```
Bot 5/8
Classe : awele.bot.AlphaBeta.AlphaBetaTriBot
Instanciation du bot "AlphaBeta avec Tri"
Nom du bot : AlphaBeta avec Tri
Auteur(s) : Théo Lallement
Instanciation du bot "AlphaBeta avec Tri"
Nombre de parties jouées
0
Temps d'apprentissage : 00:00:00.002
Durée d'une prise de décision : 00:00:00.069
Usage mémoire : 81,9 KiB
Usage mémoire maximum : 138,5 MiB
```

V - Heuristiques

Principe :

Dans le développement de ce bot, les heuristiques sont un ensemble de variables (coefficients) permettant au bot d'évaluer plus précisément la situation du plateau de jeu. La première étape pour mettre en place des heuristiques est de trouver lesquelles pourront servir au bot.

Heuristique 1 :

La première heuristique que nous avons décidé d'ajouter est tout simplement l'ajout de coefficient a la fonction DiffScore de fournie avec le projet. L'implémentation de cette heuristique n'a pas permis d'améliorer la puissance du bot. C'est pourquoi nous avons fixé les deux coefficients à 10 pour le score du joueur et -10 pour le score de l'adversaire.

Heuristique 2 :

La seconde heuristique ajoutée est deux coefficients (un pour chaque côté du plateau portant sur le nombre de Krou : "Le krou est l'accumulation dans un trou d'un nombre de graines suffisant pour faire un tour complet, soit au moins 12 graines". En effet, un krou permet souvent de marquer beaucoup de points en un coup.

Heuristique 3 :

La troisième et dernière heuristique que nous avons ajouté à la fonction DiffScore est un coefficient mettant en valeur les trous vides et ceux de 1 ou 2 graines. Ils sont assez différents. Quand ceux de 1 ou 2 graines permettent de marquer des points directement, les vides quant à eux donnent une information sur le nombre de points pouvant être marqué dans un futur proche. Il y a donc 4 autres coefficients ajoutés à la fonction, 2 pour le côté du bot et 2 pour son adversaire.

Un bot possédant les coefficients de score à 10 et -10 avec les 6 autres coefficients à 6 pouvait déjà battre un bot portant sur le même algorithme mais sans heuristique.

```
Affrontement 9/21
Bot Tri Heuristique a 1 vs. Bot Tri Allégé
Score : 2.6 - 0.4
Bot Tri Heuristique a 1 a gagné
Nombre de coups joués : 73.8 par match
Durée : 00:00:05.650 par match
Mémoire utilisée : 3,8 MiB
```

Jaugé à la main la puissance de chacun des coefficients est une tâche assez compliquée pour un humain. C'est pourquoi nous devions trouver une solution pour les générer pendant les 1 heure d'apprentissage.

VI - Génétique

Principe :

Un algorithme génétique permet d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. C'est exactement ce que nous recherchons ! Le sujet d'algorithme génétique est assez vaste et complexe. Certains algorithmes génétiques sont plus efficaces que d'autres mais nécessitent plus de temps à mettre en place et aussi lors de l'exécution.

Dans notre cas, sachant que nous avions "que" 6 coefficients à faire varier pendant nous avons beaucoup simplifié notre algorithme. Le principe général de notre algorithme est de comparer deux bots dont un ayant des coefficients générés aléatoirement (à chaque génération). Chaque génération confronte 20 fois le bot référant. Si il gagne 14 parties ou plus, les coefficients sont transférés au bot référant.
Pour éviter de mauvaise surprise (coup de malchance proche de la fin du temps d'apprentissage imparti) un compteur s'incrémentant à chaque victoire consécutive d'une génération. Peu avant la fin du budget temps, la dernière génération "en vie" se confronte contre la génération ayant réussi à faire le plus de victoires consécutives (durant 50 parties). Celle qui sort vainqueur de ce dernier affrontement sera la génération ayant la chance de participer au tournois

```
Bot 1/8
Classe : awele.bot.teluobBagarre.TeluobBagarre
Instanciation du bot "Teluob Bagarre"
Nom du bot : Teluob Bagarre
Auteur(s) : Théo Lallement / Alexis Maskio
Temps d'apprentissage : 00:58:32.154
Durée d'une prise de décision : 00:00:00.091
Usage mémoire : 87,1 KiB
Usage mémoire maximum : 51,7 MiB
```

```
Teluob Bagarre vs. Teluob Tri Heuristique a 1
Score : 2.8 - 0.2
Teluob Bagarre a gagné
Nombre de coups joués : 80.6 par match
Durée : 00:00:05.835 par match
Mémoire utilisée : 5,9 MiB
```

Comme nous pouvons le voir sur ces résultats, notre bot comportant la génétique bat très facilement un bot de profondeur 10 avec ses coefficients fixés à 1 tout en respectant les limites de temps.

VII - Améliorations

Notre bot pourrait être amélioré facilement de quatres façons.
La première, en améliorant l'algorithme AlphaBeta pour augmenter la profondeur de recherche. (par exemple, en lui allouant un budget la place d'une profondeur fixe).
Une autre façon d'améliorer notre bot serait de trouver de nouvelles heuristiques à implémenter pour lui permettre de mieux comprendre l'état du plateau lorsqu'il joue un coup.
Une troisième pourrait être la mise en place de stratégie utilisable par notre bot.
La dernière amélioration serait de modifier complètement notre système de génétique en créant des tournois comportant plusieurs générations se modifiant petit à petit lors de chaque tournois.

VIII - Conclusion

Nous avons trouvé que ce projet a été très enrichissant d'un point de vue programmation et algorithmique. C'était pour nous l'une des premières fois où une partie de notre note portait sur la qualité du projet que nous rendions par rapport aux autres étudiants. Cela nous a motivé car nous avions en tête de battre tous les autres concurrents lors du tournoi. En développant ce bot, avec les contraintes liées au temps, nous avons pu constater l'importance d'optimiser nos algorithmes pour qu'ils soient les plus performants et les plus rapides.

Sources

1. https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

2. https://en.wikipedia.org/wiki/Genetic_algorithm

3. https://www.myriad-online.com/resources/docs/awale/francais/strategy.htm

4. M. Saillot, Projet d'Intelligence Artificielle : J'ai Awalé de Travers, 04 2020

5. H. Funck, M. Hochlander Intelligence Artificielle : Projet Awélé, Le Crocodile Crétin, 04 2019