Intelligence Artificielle : Projet Awélé
Le Crocodile Crétin

FUNCK Hervé & HOCHLANDER Matthieu

7 avril 2019

Table des matières

1 Introduction

Dans le cadre du cours d'Intelligence Artificielle, nous avions à programmer un algorithme permettant de jouer au jeu de l'Awélé. Cet algorithme doit prendre la place d'un joueur et tenter de gagner face à un autre robot adverse.

L'Awélé est un des jeux les plus répandus de la famille mancala. Le plateau de jeu comporte douze trous, six pour un joueur et six pour un autre. Dans chaque trou peuvent être placées des graines (quatre au départ). L'objectif du jeu est de récolter plus de graines que son adversaire. Le détail des règles peut être trouvé sur Wikipédia [1].

Notre objectif est donc de créer un robot capable de jouer le plus intelligemment possible à l'Awélé, et par la même occasion de battre les robots des autres étudiants.

1. https://fr.wikipedia.org/wiki/Awalé

2 Recherches

Pour créer un robot final capable de jouer à l'Awélé comme un as, nous avons dû explorer plusieurs algorithmes permettant d'évaluer la justesse d'un coup par rapport aux autres. Dans cette partie, nous présenterons les premiers robots que nous avons créé afin d'explorer toutes les possibilités. Une bonne partie de ces variantes porte un nom particulier, un animal d'Afrique suivi d'un qualitificatif en général plutôt péjoratif.

2.1 MinMax

L'algorithme **MinMax** est l'algorithme le plus adapté pour un jeu combinatoire comme l'Awélé : si on en avait la capacité de calcul, cet algorithme serait capable de déterminer des coups parfaits, que seul un autre MinMax pourrait contrer, puisqu'il évalue toutes les possibilités de jeu pouvant découler d'une situation donnée.

Il nous est donc apparu évident d'utiliser un MinMax comme base pour notre robot, avec une profondeur maximale. Nous avons implémenté une version du MinMax avec un élagage Alpha-Bêta. Son objectif est de trouver une chaîne de décisions maximisant la différence entre les gains du joueur et de son adversaire. Sur un noeud *max*, l'algorithme gardera la décision renvoyant le plus grand écart entre le joueur et l'adversaire. Sur un noeud *min*, l'algorithme gardera la décision renvoyant le plus petit écart (pouvant être négatif) entre le joueur et l'adversaire.

Cet algorithme a donné d'excellents résultats (voir figure 1) tant sur le robot Random que sur les algorithmes KNN-1 ou KNN-2. Après quelques essais, nous avons conservé une profondeur d'exploration de 6 pour avoir une évaluation rapide lors des tests.

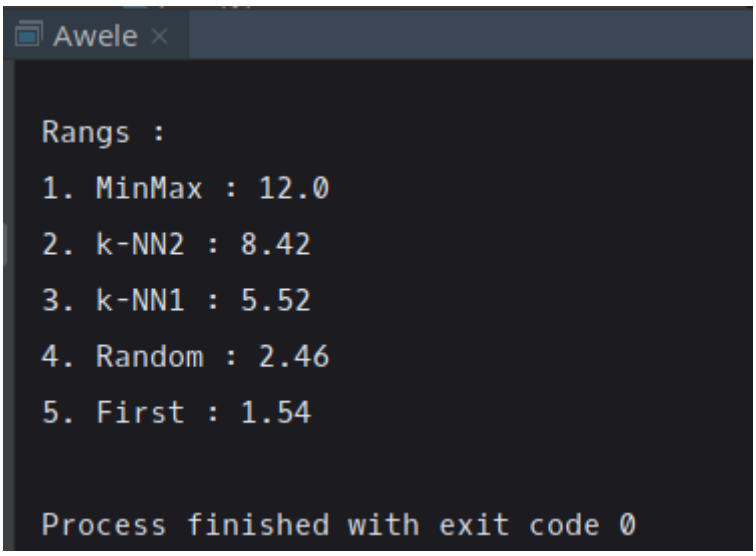


FIGURE 1 – Résultats du MinMax.

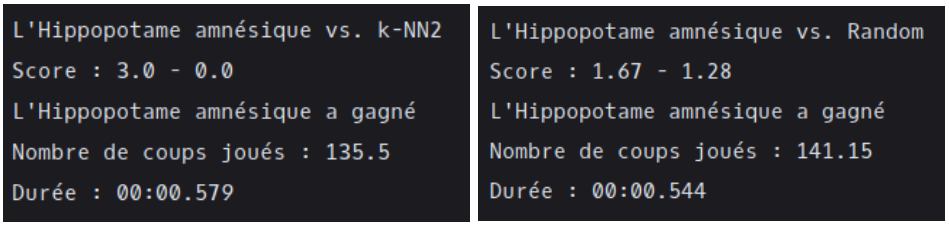
2.2 L'Hippopotame amnésique : Réseau de neurones

2.2.1 Principe

Une fois l'algorithme MinMax implémenté, nous avons cherché à mettre en place un algorithme de classification pour décider des coups à jouer. Ici, nous avons d'abord essayé un réseau de neurones (Multi-Layer Perceptron) à 6 neurones de sorties : les valeurs sur ces six neurones permettraient de déterminer quel trou choisir pour jouer. En entrée, 12 neurones,

un neurone pour chaque trou (ou chaque quantité de graines). Pour avoir des performances acceptables (augmenter le nombre de couches cachées / le nombre de neurones par couche cachée fait rapidement augmenter le temps de calcul), nous avons utilisé 3 couches cachées de 24 neurones.

Les résultats furent fort décevant ! Ils variaient beaucoup, mais pouvaient passer d'une défaite contre Random et First, à une victoire contre KNN-1 et KNN-2 (voir figure 2). Nous pourrons d'ailleurs observer ces fluctuations au fil du rapport dans les résultats des autres robots. Il était également battu à plate couture dans toutes les situations face à MinMax. Il est peut-être nécessaire de disposer d'une suffisamment grande puissance de calcul pour pouvoir faire varier le nombre de neurones et de couches cachées afin de trouver un système fonctionnel dans plus de cas et de manière plus stable, car malgré toutes nos tentatives, nous ne sommes pas parvenu à trouver de bons résultats.



(a) Résultats contre KNN-2. (b) Résultats contre Random.

FIGURE 2 – Résultats de l'Hippopotame amnésique.

2.2.2 Apprentissage

L'apprentissage du réseau de neurones de l'Hippopotame est initié par les coups joués donnés dans le sujet. Ensuite, on s'entraine pendant un temps donné en faisant jouer le robot contre lui-même (dans le résultat donné, 60 secondes, bien qu'augmenter grandement le temps d'entrainement ne changeait pas grand chose à la qualité des prédictions).

L'algorithme de l'entrainement est expliqué en Annexe 1.

2.3 L'Éléphant malpoli : MinMax + Réseau de neurones

2.3.1 Principe

Malgré la tentative infructueuse présentée précédemment, nous n'avons pas perdu espoir quant aux réseaux de neurones. Cette fois, nous avons eu une approche un peu différente : nous utiliserons un MinMax comme base, ayant prouvé sa capacité précédemment, mais nous y ajoutons cette fois une évaluation de la situation à la profondeur maximale.

L'idée est la suivante : en récupérant l'idée des étudiants du NaiveBonobo [1], nous n'avons pas voulu déterminer si un coup est fonctionnel, mais plutôt si l'état de jeu dans lequel on se retrouve au bout de l'exploration du MinMax est positif ou non. Le réseau de neurones (MLP) devra donc prendre les 12 quantités de graines en entrée et ne sortir, cette fois qu'une seule valeur : l'évaluation de la situation. L'Éléphant malpoli dipose de 2 couches cachées de 12 neurones. Augmenter le nombre de neurones rend rapidement les performances difficilement tolérables.

Les résultats n'ont pas permis de rendre le MinMax meilleur (voire figure 3). L'évaluation de la situation restait assez similaire peu importe la situation présentée, donc cela n'influait pas vraiment les décisions.

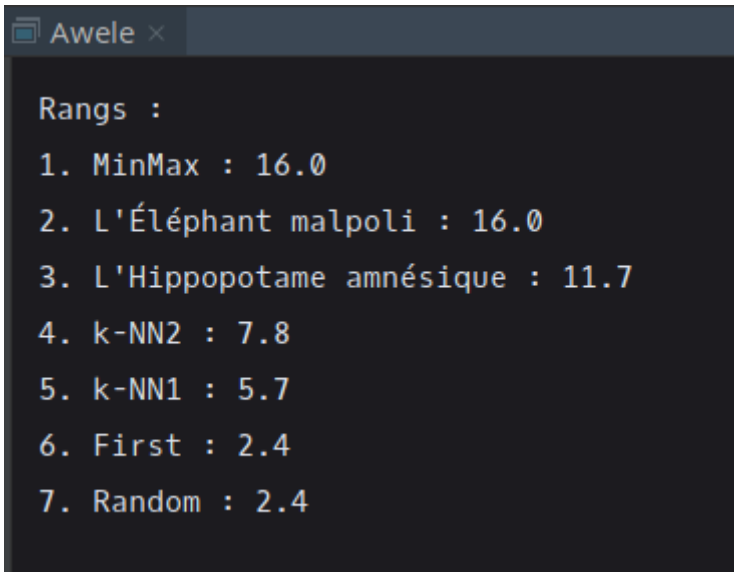


FIGURE 3 – Résultats de l'Éléphant malpoli.

2.3.2 Apprentissage

L'apprentissage du réseau de neurones de l'Éléphant est initié par les coups joués donnés dans le sujet, comme pour l'Hippopotame. Ensuite, on s'entraine pendant un temps donné en faisant jouer le robot contre lui-même. Le temps d'entrainement n'a pas eu de grand impact sur l'apprentissage de l'Éléphant. L'algorithme est le même que pour l'Hippopotame, en Annexe 1.

2.4 La Gazelle vociférante : MinMax + NaiveBayes

2.4.1 Principe

Nous reprenons la même idée que précédemment. De nouveau, nous nous servons de MinMax comme base, avec une évaluation de la situation à l'atteinte de la profondeur maximale par une classification naïve bayesienne. Grâce à la bibliothèque Java-Naive-Bayes-Classifier [2], nous avons rapidement mis en place cette variante.

La Gazelle ne donne pas d'excellents résultats (voir figure 4) contre une bonne partie des robots. Elle perd par exemple contre MinMax. Néanmoins, nous l'avons gardée dans la liste car elle parvenait régulièrement à battre certains robots basés sur d'autres méthodes de classification.

2.4.2 Apprentissage

La base de coups donnée dans le sujet était suffisante pour donner des résultats acceptables de temps en temps. Le problème principal vient probablement du fait que NaiveBayes n'est pas suffisamment précis pour pouvoir détecter les situations dans un jeu aussi complexe que l'Awélé. Ajouter des coups et des nouvelles situations générées depuis un autre apprentissage (avec un algorithme similaire que l'algorithme de l'Hippopotame en Annexe 1) n'a pas aidé à améliorer les résultats, et a même plutôt produit l'effet inverse.

2. https://github.com/ptnplanet/Java-Naive-Bayes-Classifier

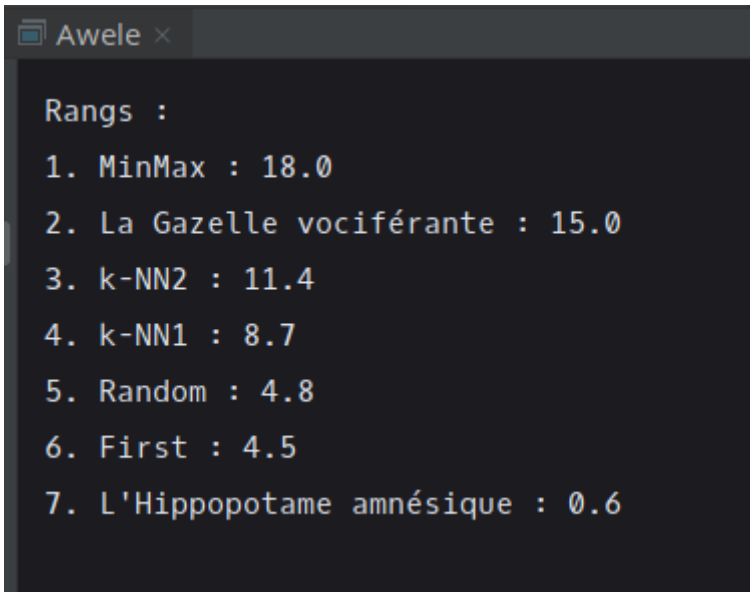


FIGURE 4 – Résultats de la Gazelle vociférante.

2.5 Le Pangolin pleurnichard : MinMax + CART

2.5.1 Principe

Cette fois, l'évaluation de la situation est effectuée à partir d'un système de forêts d'**arbres de décision**. De nouvelles situations de jeu sont générées en faisant jouer le robot contre lui-même. On ajoute ensuite les résultats des parties générées à l'ensemble d'apprentissage. Une nouvelle forêt est générée après chaque partie d'Awélé d'entrainement.

Pour l'implémentation, nous avons utilisé la bibliothèque QuickML [3] avec les valeurs par défaut [4] pour les paramètres des arbres de décision. Nous avons choisi de générer une forêt de 4 arbres car les résultats étaient satisfaisants (nous n'avons pas remarqué de grande différence en fonction du nombre d'arbres concernant la qualité des prédictions).

Étant donné que la génération des forêts est partiellement aléatoire, une fois que l'entrainement a été effectué, on recherche parmi un ensemble de forêts générées la meilleure d'entre elles. On génère donc 10 forêts qui s'affronteront pendant quelques minutes pour déterminer la quelle d'entre elles mérite le plus de représenter le Pangolin pleurnichard (d'une manière similaire aux championnats entre robots).

Les résultats du Pangolin pleurnichard sont souvent très bons ! Si nous n'avons pas été trop malchanceux dans la génération, il est capable de gagner contre un MinMax, même d'une profondeur de 1 supérieure. Il se débrouille aussi souvent très bien contre la Gazelle vociférante, bien qu'elle réussit de temps en temps à s'en sortir contre lui.

2.5.2 Apprentissage

Du fait de l'utilisation de *RandomForests*, l'algorithme d'apprentissage pour le Pangolin pleurnichard diffère un peu par rapport aux robots précédents. Ici, nous ne cherchons pas à générer de nouvelles situations, différentes, en utilisant le deuxième meilleur coup (voir Annexe 1). La partie aléatoire des forêts est peut-être un problème lorsqu'il faut jouer à un jeu où il n'y a pas de place pour l'aléatoire, mais est en revanche un gros point fort pour la

3. https://github.com/sanity/quickml
4. https://javadoc.jitpack.io/com/github/sanity/quickml/10.16/javadoc/constant-values.html#quickml.supervised.tree.decisionTree.DecisionTreeBuilder.DEFAULT_MAX_DEPTH

génération de nouvelles parties. En effet, à chaque partie jouée, on ajoute les situations vues dans cette partie et on génère une nouvelle forêt. Cette nouvelle forêt sera différente de la précédente et pourra beaucoup influencer les choix du robot.

Selon nous, cette variété nous permet de générer facilement et rapidement des coups différents et donc intéressants à traiter pour la génération d'arbres de décision prenant en compte les situations de jeu.

L'algorithme complet est détaillé en Annexe 2.

2.6 Stratège : MinMax + Stratégies

Le stratège est un robot qui se base sur un MinMax, qui, une fois la profondeur maximale atteinte, évalue si une stratégie permettant d'avoir une bonne situation de jeu est en place. Les stratégies utilisées ont été trouvées sur le site myriad-online.com [5].

Le stratège utilise actuellement 3 stratégies :

— En début de partie, quand il y a plus de 40 graines sur le plateau de jeu (choix arbitraire), il essaye de ne pas jouer de cases consécutives. Cette stratégie vise à éviter les prises multiples pour l'adversaire.

— En milieu de partie, quand il y a plus de 22 graines sur le plateau de jeu (choix arbitraire), le stratège essaye de créer un *Krou*, une stratégie classique de l'Awélé consistant à rassembler un nombre important de graines dans un trou. Ce *Krou* pourra être joué par la suite pour une prise importante de points.

— En fin de partie, quand il y a moins de 22 graines sur le plateau de jeu (choix arbitraire), le stratège essaye de voir si jouer un coup donne accès à une *fourchette*. Comme aux jeu d'échecs, une fourchette consiste à pouvoir attaquer 2 cases en même temps, le tour suivant garantit alors une prise de graines.

Ces trois stratégies ont été implémentées et testées une par une contre le MinMax classique, et ont chacune donné des résultats globalement positifs. Ensuite, les trois ont été utilisées en même temps, donnant ensemble de bons résultats.

2.7 Le Crocodile crétin : MinMax + Stratégies + CART

Stratège et le Pangolin pleurnichard ont tous les deux obtenus de bons résultats, et Stratège parvient à battre le Pangolin pleurnichard environ une fois sur deux. Nous avons donc décidé de fusionner leurs capacités ! Ce qui a donné le Crocodile crétin. Il a toujours le même inconvénient que le Pangolin pleurnichard, étant donné qu'on peut toujours ne pas avoir de chance et avoir une forêt faisant perdre contre certains autres robots.

Pour autant, le Crocodile crétin a obtenu de très bons résultats contre les autres robots, battant très souvent le Stratège et le Pangolin pleurnichard.

3 Version finale

La version finale est basée sur le Crocodile crétin, puisque c'est le robot qui a donné les meilleurs résultats dans la phase de recherche. Afin de résoudre le problème de l'aléatoire qui donne souvent des forêts différentes qui n'orientent pas toujours bien le robot par rapport à l'algorithme adverse, nous avons ajouté un système supplémentaire.

3.1 Adaptation au robot adverse

À chaque coup, on détecte en fonction du nombre de graines dans le jeu si on se trouve toujours dans la même partie ou dans une nouvelle partie. Si on se trouve dans une nouvelle

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

partie, on regarde si la précédente a été gagnée. Ce n'est pas difficile à déduire puisqu'on a enregistré chaque étape de jeu de la partie précédente, et que pour chaque étape, on a sauvegardé qui devrait être le gagnant d'après MinMax et le nombre de graines possédées par chaque joueur.

Dans le cas d'une défaite, il apparaît comme évident que la forêt d'arbres de décision n'est pas adaptée au robot adverse. On en régénère donc un nouveau, et on réessaie ! En règle générale, au bout de 5 tentatives au maximum, l'algorithme arrive à trouver une forêt adaptée à l'adversaire (sauf si la profondeur de MinMax n'est pas assez grande, à ce moment là les décisions ne sont pas assez réfléchies et les prédictions s'avèrent alors plutôt mauvaises).

Cela nous a également permis de vérifier la supériorité du Crocodile crétin face au Pangolin pleurnichard, puisqu'il a gagné contre lui en appliquant ce système aux deux robots.

FIGURE 5 – Résultats finaux.

4 Conclusion

Ce projet a été pour nous une opportunité d'approcher diverses techniques d'intelligence artificielle et par conséquent de comprendre mieux ces techniques au travers d'une application concrète. Nous avons principalement cherché à évaluer les situations de jeu pour renforcer notre bot final et sommes confiants de ses résultats.

Une des améliorations possibles pourrait être de chercher à implémenter des stratégies défensives et de les privilégier lors d'une situation de jeu difficile. On pourrait également obtenir une évaluation de la qualité de la victoire ou la gravité de la défaite. En effet, avec notre méthode d'évaluation actuelle, il est difficile de savoir si une victoire est une grande ou une petite victoire. Gagner avec 20 graines d'avance est beaucoup mieux que de gagner avec 3 graines d'avance, et notre évaluation actuelle n'en tient pas compte. Cette évaluation de la qualité d'une situation pourrait peut-être être approchée grâce à une méthode de régression.

Références

[1] D. Kaced & M. Simon. *NaiveBonobo*. 2016.

5 Annexes

5.1 Annexe 1 : Apprentissage des réseaux de neurones

```
pour chaque élément de AweleData faire
    Mettre à jour les poids par rétropropagation du gradient.
fin

Ajouter une nouvelle partie vierge à ProchainesParties.
tant que le temps passé à s'entrainer est inférieur à N secondes faire
    tant que la partie n'est pas terminée faire
        Utiliser le robot pour déterminer le classement des décisions.
        Jouer un jour en se servant de la meilleure décision et enregistrer le coup joué.
        Ajouter la partie alternative issue de la deuxième meilleure décision à
         ProchainesParties.          // Ceci permet de créer des situations
         différentes plutôt que de générer toujours les mêmes si on
         faisait se battre le bot contre lui-même plusieurs fois de
         suite.
    fin

    Déterminer qui est le gagnant.
    Déterminer quels coups mènent à la victoire / la défaite.
    pour chaque coup joué faire
        Mettre à jour les poids par rétropropagation du gradient.
    fin
fin
```

5.2 Annexe 2 : Apprentissage des forêts aléatoires

```
pour chaque élément de AweleData faire
    Ajouter chaque situation à Training (l'ensemble d'entrainement).
fin
Générer une forêt.

tant que le temps passé à s'entrainer est inférieur à N secondes faire
    Jouer une nouvelle partie avec les décisions du robot et enregistrer chaque
     situation rencontrée dans Situations.

    Associer les coups de Situations à un état Victoire / Défaite.
    pour chaque situation dans Situations faire
        Ajouter la situation à l'ensemble d'entrainement.
    fin
    Générer une forêt.
fin
```