**Projet IN104 : Implémentation d’une IA pour Dames, Echecs, Puissance 4 et Abalone**

Tom CHOLLET  
Florian LATESTERE  
20/05/2020

**Introduction :**

L’objectif de ce projet est de créer et d'implémenter une Intelligence Artificielle (IA) en langage Python qui puisse jouer aux Échecs, Dames, Puissance 4, Abalone. A travers le package Aiarena qui gère l’interface et nous permet d’essayer facilement notre IA.

Les étapes de la réalisation sont les suivantes :

* Installation du package
* Création d’une IA qui joue aléatoirement
* Implémenter un algorithme minimax
* Créer des fonctions d’évaluations de position pour chaque jeu.
* Augmenter l'efficacité du programme minimax
* Adapter l’IA pour qu’elle puisse jouer en un temps donné
* Créer la fonction d’évaluation la meilleur possible

**Trouver la bonne estimation de Trecherche :**

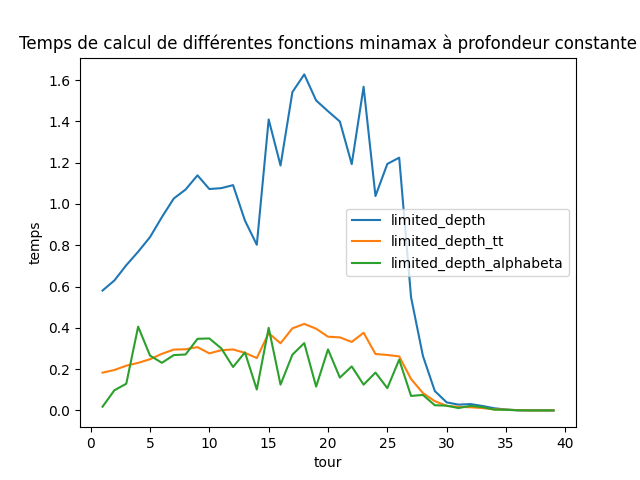
Pour que l’algorithme fonctionne correctement, il faut commencer avec une bonne estimation de Trecherche. Correspondant ou temps que met l’ordinateur à rechercher les états suivant possibles et à évaluer une position.

Notre première idée était de faire une moyenne de plusieurs séries de recherche et évaluation. Mais finalement nous avons opté pour un maximum, effectivement, il est mieux de majorer le temps de recherche puisque l’algorithme redistribue le temps non utilisé.

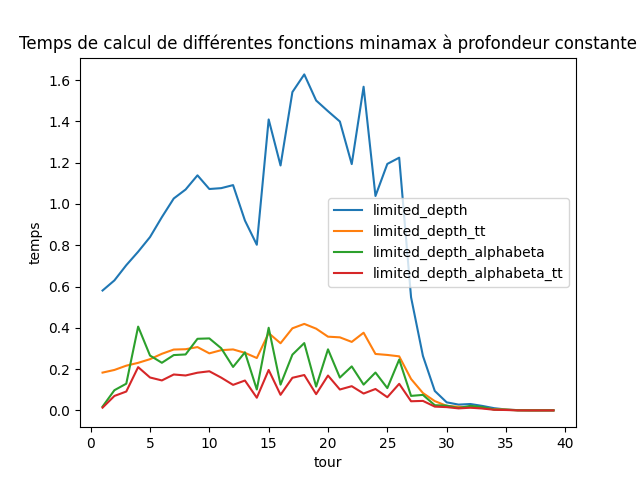
**Optimisation de minamax :**

L’un des objectifs principaux du projet est de réaliser des intelligences artificielles les plus fortes possible. Pour cela il y a une forte contrainte d’efficacité en temps donné : les matchs se disputent en effet en temps limité et notre IA doit être capable de vite trouver le meilleur coup possible. Pour rendre notre IA la plus forte possible il faut donc trouver un moyen de la faire explorer le plus profondément possible les branches de l’arbre des coups possibles.

Trois codes de fonction minamax ont été proposés et pour évaluer leurs performances, nous avons fait en sorte que deux IA jouent la même partie avec une profondeur de 5 fixée et nous avons recueilli le temps qu’elles mettaient à chaque coup pour parcourir la même branche de l’arbre. La *figure 1* détaille les efficacités des trois algorithmes proposés et comme on peut le remarquer, l’élagage alpha-beta est la méthode la plus efficace.



*Figure 1 :*

 Seulement, nous avons pensé que dans le but d’être encore plus efficace, nous pouvions additionner le principe de la table transposition et de l’élagage alpha-beta. Ce nouvel algorithme nous avons comparé ses performances avec le même principe de fonctionnement ce qui nous donne la *figure 2*. Les performances étant supérieure comme on pouvait s’y attendre, nous avons décidé de par la suite implémenter une fonction limited\_time\_alphabeta\_tt.py qui combinait les deux principes afin de gagner un peu de temps et donc de parcourir un peu plus l’arbre par la suite.

*Figure 2*

**Fonction evaluate du connect4 :**

* Première tentative

La fonction evaluate est en grande partie ce qui va rendre notre IA intelligente, il faut donc réussir à la rendre la plus précise possible. Pour réaliser cet objectif nous avons commencé par réfléchir à tous les coups et les différentes sommes de points qu’ils pouvaient rapporter.

Le bût étant d’aligner 4 pions, privilégier les longues séries de jetons semblait être l’idée principale, mais il faut aussi remarquer que certaines séries de pions sont plus inutiles que d’autre et nous avons jugé bon de les différentier. On peut en effet voir dans les *figures 3 & 4* que même si aligner trois pions est important, cela doit pouvoir mener sur une possible série de quatre plutôt que de mener sur un emprisonnement de trois pions.

*Figure 4 : alignement qui n’ouvre sur rien*

*Figure 3 : alignement pour ouvrir sur un puissance 4*

Pour prendre en compte ce problème, notre fonction evaluate ne se contente pas de trouver s’il existe des suites de pions mais elle devait aussi prendre en compte si cette suite de pions permettait d’ouvrir sur un puissance 4 d’un côté, voire même des deux côtés s’il n’y a aucun pion ni aucun mur de part et d’autre des séries de pions. Ainsi, evaluate va parcourir tout le plateau et dès qu’elle tombe sur un pion, elle regarde la série maximale de pions qui découle du pion actuel dans chacune des 4 directions possibles et va ajouter des points en fonction du nombre de pions alignés et des ouvertures de chaque côté de la série.

P

Si on prend le pion P pour exemple, horizontalement il y a une série de 3 pions qui part de lui mais elle est bloquée de chaque côté donc cela n’apporte pas de points. Verticalement il n’y a une série que d’un pion, mais elle est ouverte d’un côté donc cela rapporte quelques points. Sur la diagonale qui part du bas gauche pour aller vers le haut droit on a une suite de trois pions qui est ouverte d’un côté cela rapporte donc beaucoup de points. Finalement sur la dernière diagonale on à une suite de deux pions qui est aussi ouverte d’un côté donc cela rapporte aussi pas mal de points.

Finalement, faire un puissance 4 rapporte une infinité de points. Comme nous regardons tous les pions et tous les alignements, il faut prendre en compte que des alignements de pions jaunes donnent des points positifs mais des alignements de points rouges donnent des points négatifs.

Nous avons donc implémenté cet algorithme pour le puissance 4 qui semblait assez complet avant de se rendre compte de plusieurs problèmes.

* Premièrement, en donnant une infinité de points à un puissance quatre où à un contre de puissance quatre on ne permettait pas de faire la distinction entre un puissance 4 que l’on pouvait faire au prochain tour où un puissance 4 que l’on faisait dans 3 tours. Ce problème menait à des situations désagréables où l’IA ne gagnait pas en 1 coup alors qu’elle le pouvait car elle voyait un moyen de gagner en 3 coups et donc perdait.
* Deuxièmement, cet algorithme était en fait plutôt fait pour trouver la meilleure solution pour le coup suivant plutôt que pour évaluer des situations loin dans le futur ce qui donc ne concordait pas trop avec notre objectif dans ce projet avec notre fonction minamax, son élagage alphabeta et sa table de transposition qui essayent d’aller toujours plus loin.

Cet algorithme n’est pas un échec en soit, mais nous voulions trouver mieux. Nous avons dans implémenté un autre algorithme.

* Seconde tentative

L’aspect principal sur lequel nous voulions jouer était de privilégier les puissances 4 ou les contres qui se faisaient en 1 coup devant ceux qui se faisaient en 2, 3, 4 ou plus de coups. Pour cela, il nous est venu l’idée de donner une très grande valeur de points à un puissance 4 mais de la diviser par le nombre de pions sur le plateau. Ainsi un puissance 4 fait en un tour rapportera plus qu’un puissance quatre fait en 4 tours car il y aura moins de pions sur le plateau.

De plus, nous voulions simplifier le reste de notre fonction d’évaluation afin de renvoyer des valeurs plus rapidement et donc sans associer des scores différents en fonction de l’ouverture qu’offre les alignements. On veut coder un programme plus simple mais qui profite plus de la profondeur d’exploration. Pour cela, on se contente d’attribuer des points en fonction du nombre de séries de trois pions, deux pions et un pion d’avance sur l’adversaire, les séries de trois pions rapportant plus de points. On va donc toujours parcourir tout le plateau mais juste retourner la longueur maximale de pions alignés dans chacune des directions à partir du pion actuel.

L’idée de diviser la valeur d’un puissance 4 semblait bonne donc nous l’avons aussi implémentée dans l’algorithme précédent afin d’évaluer les performances des deux algorithmes. Pour vérifier si ce nouvel algorithme répondait mieux aux attentes, nous avons donc simulés environ 60 parties de 1 contre 1 entre les algorithmes avec pour temps de réflexion 0.5 secondes puis 2 secondes et enregistré le gagnant de chaque partie ce qui donne la *figure 5*.

Le premier algorithme qui était plus lent car plus complexe s’est donc en effet avéré moins bien que le second, c’est donc pour cette raison que nous avons finalement implémenté le second algorithme, bien qu’il soit plus simple.

**Fonction evaluate du jeu de dames :**

L’idée initiale pour juger une position aux dames est tout naturellement de regarder lequel des deux joueurs possède le plus de pions. Mais une problématique surgit tout de suite. Quelle est la valeur d’une dame par rapport à un pion ? Une dame vaut 2 pions ? 3 pions ? 2,17 pions ?

Il convient donc de faire s’affronter plusieurs IA avec différentes valeurs des dames pour savoir laquelle joue le mieux. On remarque grossièrement que la meilleure valeur se trouve entre 2 et 5 pions car ces IA vont le plus souvent gagner un exemple des résultats est présenté sur la *figure 6*. Il est difficile d’affiner plus cette valeur avec nos outils puisque que les IA font toujours match nul mais il est possible de faire appel à notre bon sens pour comprendre qu'une dame ne vaut pas 5 pions. Nous choisirons donc 1 dame=2 pions.

Si on veut espérer que notre IA se démarque il faut lui donner plus d’information que juste le nombre de pièces sur le plateau.

On espère que notre IA va être assez agressive et va essayer de prendre possession du camp adverse. C’est pourquoi on choisit d’augmenter la valeur d’un pion si celui ci se trouve proche du camp adverse.

Il semble que cette amélioration augmente le niveau de l’IA mais rien n’est flagrant car les parties entre IA sont souvent nulles.

Remarque : l’IA ainsi crée semble très bien jouer en début et milieu de partie mais arrivée en fin de partie, elle est très défensive et ne va pas chercher à attaquer l’adversaire pour gagner même si elle est en supériorité numérique.

**Fonction evaluate de l’abalone :**

Nous avons décidé de réfléchir à une intelligence artificielle pour le jeu d’abalone en codant sa fonction d’évaluation. Connaissant les règles de ce jeu, nous avons décidé de concentrer la répartition des points sur trois éléments importants :

-Le nombres de pièces prises.

-Le fait que nos pions soient situés au niveau du centre du plateau (il est avantageux d’être loin des bords).

- Le fait que les pièces soient groupées.

En effet, avoir des groupements de pièces les rends plus difficile à bouger pour l’adversaire et se situer plus proche du centre permet de ne pas se faire éjecter du plateau trop facilement. Mais en outre, prendre les pièces adverses est vraiment le plus important car il suffit de trois pièces prises pour perdre une partie, il faut donc jouer sur les différentes valeurs des points pour s’assurer que la part prise par le groupement et la position des pions n’est pas trop importante devant celle prise par le simple fait de prendre une pièce adverse. Pour coder cette fonction evaluate, nous avons parcouru le plateau et attribué des points en fonction de la position de chaque pièce et du nombre de voisins direct qu’elle possédait. Malheureusement, nous n’avons pas trop pu tester notre code car nous avons eu des problèmes avec la partie abalone du module aiarena donc nous n’avons pas trop pu jouer sur les différents coefficients pour essayer de déterminer ceux qui rendraient l’IA la plus intelligente possible.

Toutefois nous avons pu observer l’IA jouer les premiers coups et elle se dirigeait bien vers le centre du plateau, en bloc, ce qui ressemble à ce que nous recherchions.