8INF846 – Intelligence Artificielle

Travail No. 2
Compétition d'agents intelligents pour le jeu d'échecs

15 décembre 2017

Esmé James JAME15539504
Wilfried Pouchous POUW04069501

Sommaire

I. Préambule **3**
A. Méthode choisie 3
B. Architecture des fichiers 3

II. Pre-exploration MiniMax **4**
A. Couleur de l'IA 4
B. Tableaux 2 dimensions 4
C. Inversion échiquier (si IA noir) 5

III. Exploration MiniMax **6**
A. Tous les coups possibles 6
B. Fonction d'évaluation simple 7
C. Algorithme MiniMax 8

IV. Post-exploration MiniMax **11**
A. Reset de la couleur de l'IA 11
B. Inversion du Best Move (si IA noir) 11
C. Best Move 11

V. Améliorations **12**
A. Élagage Alpha-beta 12
B. Nouvelle fonction d'évaluation 12
C. Promotions 13

Conclusion **14**

I. Préambule

A. Méthode choisie

Pour cet agent intelligent qui joue aux échecs, nous avons choisi d'implémenter l'algorithme Minimax couplé à l'élagage alpha-béta. La raison pour laquelle nous avons choisi cette exploration est parce qu'il s'agit d'une des techniques les plus utilisées dans le domaine des intelligences artificielles jouant aux échecs.

Cette technique étant plutôt bien documentée sur internet, nous nous sommes aidés de cours d'intelligence artificielle disponibles sur le le web et de codes ou pseudocodes déjà écrits afin de mener à bien ce projet.

B. Architecture des fichiers

Le ProcessAI1 qui contient notre agent jouant aux échec est divisé en trois fichiers :

- ***Program.cs*** : Contient le code de base permettant de communiquer avec le plateau. Il contient également la création d'un objet *IAEchec*, chargé de l'exploration et de renvoyer le meilleur coup trouvé. Cette classe contient la boucle de vie de l'agent.

- ***Echiquier.cs*** : Contient toutes les connaissances liées au plateau d'échec. C'est cette classe qui contient les fonctions de vérification de la couleur de l'IA, de l'affectation des poids aux pièces, d'évaluation, de recherche des coups possibles, etc.

- ***IAEchec.cs*** : Classe contenant les fonctions relatives à l'exploration et à la recherche du meilleur coup. L'instanciation de cet objet dans *Program.cs* créer une instance de *Echiquier* dans la classe *IAEchec*.

II. Pre-exploration MiniMax

Lors du développement de notre intelligence artificielle, nous avons décidé de procéder par étapes, des plus simples aux plus complexes. La première partie du développement correspond à la mise en place de certains critères de jeu (tel que la couleur de l'IA), traduite dans le code par la méthode « **InitEchiquier** ».

Cette première partie nous a également permis de rentrer dans le code et de comprendre comment l'existant fonctionnait.

A. Couleur de l'IA

Quand une partie commence, la première chose essentielle que l'IA doit savoir est : quelle couleur m'a été attribuée? En effet, beaucoup d'éléments, comme le sens de mouvement des pièces, vont découler de l'affectation de cette couleur.

Pour deviner la couleur de l'IA, nous effectuons un test simple qui vérifie si les quatres lignes du milieu de l'échiquier (lignes 3 à 6) sont vides au premier tour :

- si elles le sont, alors c'est à notre IA de jouer en premier, nous sommes donc blanc.

- si une case est occupée, un coup a déjà été procédé, nous sommes donc noir.

Nous avons développé sur l'IA1 fourni qui est toujours blanc par défaut. Néanmoins, notre IA pouvant se faire attribuer le rôle des noirs durant le tournoi, nous avons fait en sorte qu'il puisse jouer les deux rôles.

B. Tableaux 2 dimensions

Pour faciliter notre développement par la suite, nous préférions travailler avec des tableaux deux dimensions pour représenter le plateau de jeu. Nous avons donc créer deux fonctions pour transformer les tableaux une dimension existants en tableaux deux dimensions :

- la méthode « **TabEvaluation** » : transforme le tableau tabVal, contenant les pièces et leurs valeurs, en tabEval2D contenant les pièces et des nouvelles valeurs associées. Nous avons changé la valeur des pièces car nous trouvions que celles utilisées par défaut complexifiaient l'identification des pièces (par exemple, différentes valeurs pour un cavalier de la même couleur).

- la méthode **«TabCoord2D** » : transforme le tableau tabCoord, contenant les coordonnées des cases de l'échiquier, en tabCoord2D contenant ces coordonnées.

C. Inversion échiquier (si IA noir)

Dans le cas où l'IA joue en noir, on retourne le plateau de jeu (tabEval2D) pour faire croire qu'il joue en blanc et donc pour simplifier l'exploration suivante. Il faudra juste penser à la fin de l'exploration, à inverser le déplacement choisi afin qu'il soit cohérent avec la couleur noir (cf. partie IV. B.)

III. Exploration MiniMax

A. Tous les coups possibles

Avant d'entamer l'exploration, il faut que notre agent soit capable de récupérer l'ensemble des coups possibles dans un état donné, pour un joueur donné. On a donc créé une fonction « **GetAllMoves** » permettant de faire cela.

Dans cette fonction, une simple boucle parcourant l'ensemble des pièces du joueur permet de récupérer tous les coups possibles pour chacune d'entre elles.

Pour chaque type de pièce, nous avons calculé les coups possibles en les schématisant d'abord à l'aide de tableaux représentants le plateau.

Par exemple, la figure ci-dessous représente un plateau avec tous les coups possibles pour une reine positionnée en (4,4). La calcul des coups possibles récupère les huits directions dans laquelle la reine peut se déplacer (cases rouges) et incrémente chaque direction (cases mauves) jusqu'à rencontrer un des éléments suivants :

- la fin du plateau
- une pièce alliée (on s'arrête juste avant la case contenant la pièce)
- une pièce ennemie (on s'arrête sur la case pour la « manger »)

Reine

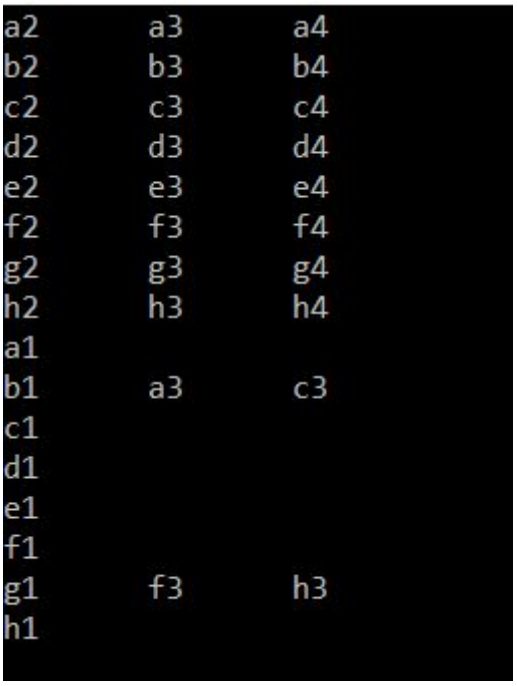
i=0 , j=0	0,1	0,2	0,3	0,4	0,5	0,6	0,7
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7

movePossible = { (-1,-1), (0,-1), (1,-1), (1,0), (1,1), (0,1), (-1,1), (-1,0) }

L'ensemble des coups possibles des pièces ont été calculés d'une manière semblable.

Chaque mouvement d'une pièce est stockée dans une liste, où le premier élément de cette liste est la pièce à l'origine de ces mouvements. Puis, chaque liste est stockée dans la liste parente, cette dernière contenant toutes les listes des coups de chaque pièce du joueur. Par exemple, l'image ci-dessous illustre ces propos :

- chaque ligne étant une liste de coups.
- chaque premier élément de la liste étant l'origine des coups suivants.



L'image ci-dessus montre l'ensemble des coups possibles pour le premier tour du joueur blanc. On voit bien que la pièce qui est présentement en a2 (un pion), peut aller en a3 ou a4. La pièce en b2, peut aller en b3 ou b4 et ainsi de suite, pour chaque pièce. Toutes les pièces n'ont pas forcément de coup possible pour le tour, comme les pièces situées en a1 ou c1 par exemple.

B. Fonction d'évaluation simple

Nous avons ensuite attribué un poids à chaque pièce afin d'en mesurer sa valeur dans le jeu. Ci-dessous deux tableaux représentant les valeurs que nous avons utilisées.

Noirs	
Pièces	**Poids**
Pion	-10
Cavalier	-30
Fou	-40
Tour	-50
Reine	-90
Roi	-900

Blancs	
Pièces	**Poids**
Pion	10
Cavalier	30
Fou	40
Tour	50
Reine	90
Roi	900

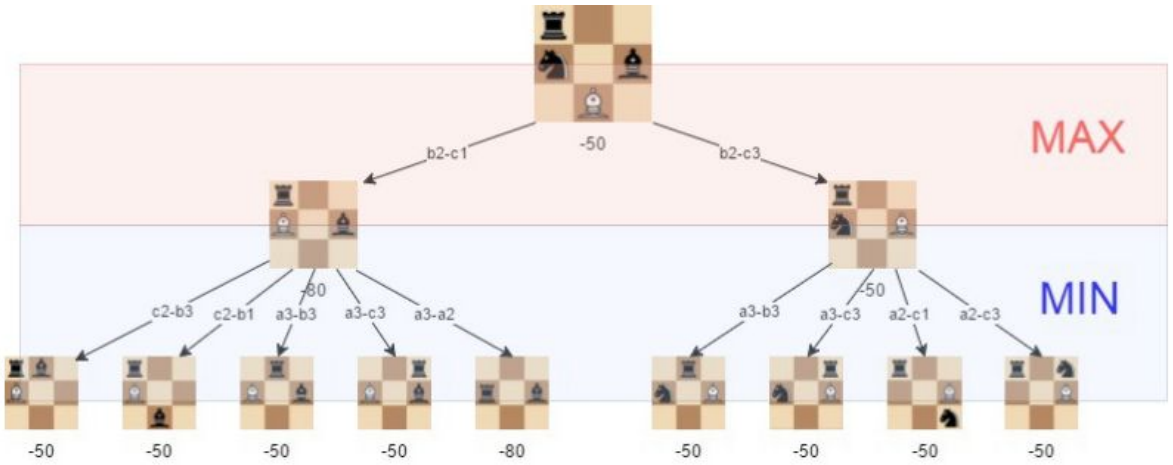
Ces poids sont ensuite utilisés dans une fonction d'évaluation simple permettant de calculer la valeur du plateau d'échec à un instant donné. Le calcul est simple : on somme les poids de toutes les pièces. Si le résultat est positif, c'est un bon coup pour l'agent blanc, sinon c'est un bon coup pour l'agent noir.

En appliquant cette évaluation à la recherche de coups possibles, l'agent effectuera le coup qui lui rapportera le score le plus élevé (s'il est blanc). Ainsi, en trouvant le coup apportant un meilleur score à chaque tour, l'agent, s'il peut, mangera la pièce de l'adversaire. L'évaluation permet donc de créer un agent simple avec une intelligence très basique.

C. Algorithme MiniMax

A travers ce TP, nous avons implémenté l'algorithme Minimax avec une profondeur d'exploration des coups possibles de 3. Cet algorithme est un algorithme récursif qui consiste à explorer toutes les possibilités jusqu'à une profondeur fixée et de retourner une valeur de l'état de l'échiquier lorsque cette profondeur est atteinte (via la fonction d'évaluation mise en place au préalable). Le but de MiniMax est de choisir le meilleur coup possible pour le joueur l'ayant instancié tout en sachant que l'adversaire fera de même après, on dit alors qu'on : minimise la perte maximum.

Le joueur ayant implémenté l'algorithme est désigné comme le joueur MAX tandis que son adversaire est désigné comme le joueur MIN. A chaque appel récursif, l'algorithme calcule les possibilités de mouvements pour un des deux joueurs. Comme l'image le montre ci-dessous, pour tous les coups possibles d'un joueur, on retient soit la valeur maximum renvoyée (cas du joueur MAX), soit la valeur minimum renvoyée (cas du joueur MIN).



De plus, à chaque appel récursif de la fonction MiniMax, on décrémente la valeur de la profondeur et on change la valeur du tour du joueur. Changer la valeur du tour du joueur sert à s'intéresser alternativement aux mouvements possibles des deux joueurs. Décrémenter la profondeur sert à atteindre la profondeur limite fixée.

Notre algorithme MiniMax est construit de la façon suivante :

- récupérer tous les coups possibles du joueur (double liste)
- en fonction du joueur, parcourir chaque liste de coups :
    - conserve la première valeur comme origine du coup
    - pour toutes les autres valeurs de la liste, qui sont des mouvement possibles :
        - on effectue le coup sur l'échiquier
        - on appelle la fonction MiniMax (tout en décrémentant la profondeur et en changeant le tour du joueur)
        - on conserve la meilleure (MAX ou MIN) valeur entre l'actuelle et celle retournée
        - on défait le coup précédent sur l'échiquier
        - on recommence jusqu'à que la liste de coups soit fini
    - on recommence jusqu'à que la double liste de coups soit fini

Dans notre code, nous avons séparé deux cas dans la partie du joueur MAX : la profondeur initiale et les autres profondeurs. Cela s'explique du fait que dans la profondeur initiale, contrairement aux autres profondeurs du joueur MAX, on met à jour le « Best Move » lorsque la valeur max change.

Enfin, nous avons choisi une profondeur de 3 car, après plusieurs tests, il s'agit de la profondeur maximum auquel notre IA peut aspirer s'il veut respecter la contrainte de temps imposée.

Sur la page suivante, un exemple du déroulement de l'algorithme MiniMax : il s'agit du début de l'exploration du premier tour de l'IA blanc.

On voit bien que pour le déplacement du pion blanc de a2 vers a3 et ensuite, pour le déplacement du cavalier noir de b8 à a6, l'IA vérifie tous les mouvements possibles de ses pièces (blanches).

```
Explo (depth/isIA) : 3/True
Piece : a2 / Move : a3

<!-- TOUR adversaire -->
Explo (depth/isIA) : 2/False
Piece : a8 / Move :
Piece : b8 / Move : a6

<!-- TOUR IA -->
Explo (depth/isIA) : 1/True
Piece : a3 / Move : a4
Piece : b2 / Move : b3 b4
Piece : c2 / Move : c3 c4
Piece : d2 / Move : d3 d4
Piece : e2 / Move : e3 e4
Piece : f2 / Move : f3 f4
Piece : g2 / Move : g3 g4
Piece : h2 / Move : h3 h4
Piece : a1 / Move : a2
Piece : b1 / Move : c3
Piece : c1 / Move :
Piece : d1 / Move :
Piece : e1 / Move :
Piece : f1 / Move :
Piece : g1 / Move : f3 h3
Piece : h1 / Move : <!-- FIN IA !-->
```

IV. Post-exploration MiniMax

A. Reset de la couleur de l'IA

Une fois sorti de l'exploration MiniMax, on réattribue sa couleur à l'IA car celle-ci a été modifiée durant MiniMax (utilisée comme valeur du tour du joueur). On doit absolument faire cela car on doit encore effectuer une action en fonction de la couleur de l'IA.

B. Inversion du Best Move (si IA noir)

Comme expliqué en début de rapport, on doit inverser le Best Move trouvé dans le cas où l'IA joue en noir car ce mouvement correspond à un déplacement du côté blanc. Pour inverser le Best Move, on trouve les nouvelles coordonnées de sa position et de sa destination en soustrayant les anciennes coordonnées à 63 (nombre de cases de l'échiquier).

C. Best Move

On récupère le Best Move via un getter. Le Best Move est contenu dans un tableau une dimension de String :

- bestMove[0] = pièce qui se déplace
- bestMove[1] = destination de la pièce

On attribue ces deux valeurs à la variable chargée d'envoyer les coordonnées de mouvement au plateau de jeu : coord.

V. Améliorations

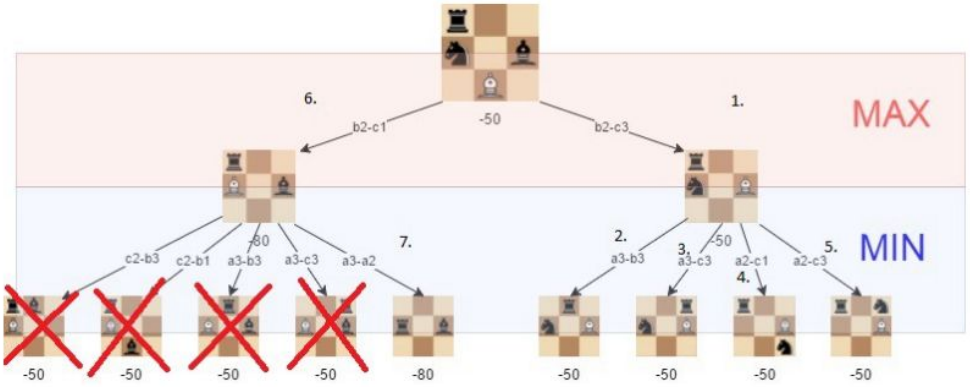
A. Élagage Alpha-beta

L'élagage alpha-bêta est une technique permettant de réduire considérablement le nombre de noeuds explorés lors d'une exploration de type MiniMax.

Concrètement, il se base toujours sur l'actuelle valeur des profondeurs supérieures (alpha et beta) et en fonction de ces valeurs, il décide d'arrêter ou non l'exploration des noeuds enfants. En cas d'arrêt, c'est que les valeurs de cette partie de l'exploration n'ont pas d'intérêt.

Par exemple, dans l'image ci-dessous, le noeud racine a pour valeur -50 après l'exploration du noeud fils droite. Puis, lorsqu'il explore le noeuds fils gauche et qu'il est au tour du joueur MIN, un des noeuds enfants de ce dernier a pour valeur -80. Étant le joueur MIN, il retiendra cette valeur plutôt que n'importe quelles valeurs plus grandes (-50 par exemple). Or, le noeud racine est le joueur MAX, il prendra donc la valeur maximum entre -50 et celle retournée par le noeuds fils gauche qui vaut à ce moment -80.

Ainsi le noeud racine sait déjà qu'il conservera sa valeur de -50 et qu'il ne retiendra pas la valeur du noeud fils gauche. Donc, continuer l'exploration des noeuds dans cette branche n'a plus d'intérêt et on peut l'arrêter. C'est en suivant cette logique que nous avons codé l'élagage.



B. Nouvelle fonction d'évaluation

Nous avons fait évoluer notre fonction d'évaluation afin que celle-ci soit plus efficace et rende l'agent plus fort aux échecs.

Auparavant, l’agent se basait uniquement sur la valeur des pièces présentes. Maintenant il ajoute à cela une plus-value en fonction de la position de chaque pièce. De ce fait, chaque type de pièce a un tableau comme ceux en exemple ci-dessous (roi à gauche et reine à droite) afin de connaître la plus-value de la pièce en fonction de sa position sur l’échiquier.

```
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -3.0, -4.0, -4.0, -5.0, -5.0, -4.0, -4.0, -3.0],
[ -2.0, -3.0, -3.0, -4.0, -4.0, -3.0, -3.0, -2.0],
[ -1.0, -2.0, -2.0, -2.0, -2.0, -2.0, -2.0, -1.0],
[  2.0,  2.0,  0.0,  0.0,  0.0,  0.0,  2.0,  2.0 ],
[  2.0,  3.0,  1.0,  0.0,  0.0,  1.0,  3.0,  2.0 ]
```

```
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0],
[ -1.0,  0.0,  0.0,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -0.5,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[  0.0,  0.0,  0.5,  0.5,  0.5,  0.5,  0.0, -0.5],
[ -1.0,  0.5,  0.5,  0.5,  0.5,  0.5,  0.0, -1.0],
[ -1.0,  0.0,  0.5,  0.0,  0.0,  0.0,  0.0, -1.0],
[ -2.0, -1.0, -1.0, -0.5, -0.5, -1.0, -1.0, -2.0]
```

Prendre en compte la position des pièces peut être très intéressant du fait que certaines pièces ont un plus grand intérêt dans certaines régions de l'échiquier.

C. Promotions

Le cas de la promotion, c’est-à-dire un pion qui atteint le côté adverse, a été traité. Après l’exploration MiniMax, une méthode est chargée de regarder si le Best Move correspond à une promotion. S’il s’agit d’un cas de promotion, l’agent décide de transformer son pion en reine (il est intelligent !).

Conclusion

L’agent intelligent qui a été développé réussi donc à joueur correctement aux échecs. Il réussit notamment à vaincre l’agent aléatoire fourni sans perdre aucune de ses pièces.

Cependant, lorsque l’on fait jouer l’agent contre lui-même, il arrive qu’ils soient tous deux pris dans un minimum local et bouclent donc sur les deux mêmes coups à l’infini.