Intelligence Artificielle

I – Validation

1) Niveaux de difficultés

Nous avons 4 niveaux de difficulté implémentés : facile, défensif, agressif et difficile, le temps de calcul augmentant de manière croissante.

1.a) Cohérence

La version du jeu utilisée à été celle qui nous paraissais la plus cohérente : le roi doit s'échapper par les coins, peut manger, n'est pas limité dans son déplacement et les moscovites commencent. C'est une des versions qui nous est apparue comme étant la plus équilibré après de nombreux tests entre nous.

Ratio de victoire suédoise				
Suédois	Moscovites			
	Difficile	Agressif	Défensif	Facile
Difficile	100%	100%	100%	97%
Agressif	66%	72%	51%	80%
Défensif	20%	85%	57%	77%
Facile	81%	92%	94%	85%

Les données en gris clair sont issues de tests à seulement 5 parties. Celles en noir sont issues de tests à 100 parties et sont donc bien plus représentatives.
On peut constater que les IA agressives et défensives sont selon la situation légèrement en-deçà de l'IA facile au niveau de l'affrontement d'IA mais contre des vrais joueurs ce n'est plus le cas car les IA agressives et défensives déstabilisent le joueur.

1.b) Pertinence

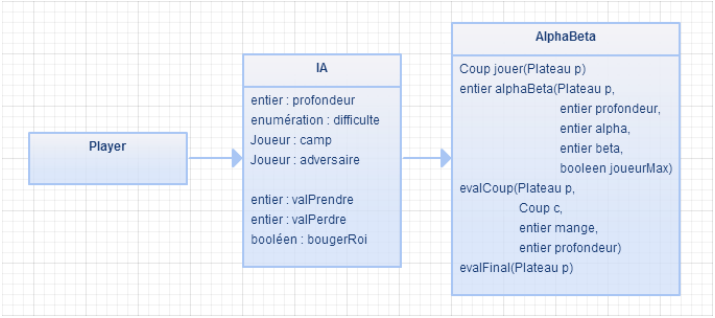
Après de nombreux tests il apparait que la complexité du jeu fait que l'IA facile tend à battre les utilisateurs débutants qui n'ont pas encore bien saisi les règles. Mais elle se fait bien plus battre par des utilisateurs plus expérimentés. L'IA agressive semble clairement sacrifier des pions pour en prendre d'autres tandis que l'IA défensive à plus tendance à rester grouper. Enfin l'IA difficile est un compromis entre ces différents niveaux.

2) Pistes explorées mais abandonnées

Des IA plus difficiles ont été crées, mais de par leur temps de jeu (de l'ordre de plusieurs minutes voir dizaines de minutes), elles ont été abandonnées.

II – Spécifications techniques

1) Structure de l'IA



1.a) Classe Player

Player est une classe issue de la LibGdx, son utilisation permet une intégration plus simple de l'IA.

1.b) Classe IA

IA est une classe générique d'IA qui peut être utilisée par plusieurs algorithmes différents (à l'origine nous avions une classe MiniMax).
Plusieurs données y sont stockées, notamment la profondeur de l'algorithme, quelle est l'IA utilisée : Facile, Défensive, Agressive, Difficile, ainsi que d'autres servant à mieux personnaliser l'alpha-bêta.

1.c) Classe AlphaBeta

AlphaBeta est la seule classe finalement utilisée car la plus performante.
Elle dispose de plusieurs fonctions, notamment les fonctions d'évaluation qui doivent de manière optimisée évaluer la branche étudiée :

- evalCoup permet d'évaluer chaque coup selon plusieurs paramètres : ajouter à la valeur d'évaluation de l'algorithme les éventuelles prises qui ont été faites, ainsi que selon les niveaux de difficultés, une autre valeur selon le déplacement du roi : il cherche à se rapprocher des bords.
- evalFinal permet d'évaluer l'état final du plateau en fonction principalement de l'emplacement du roi : s'il est entouré par des ennemis, c'est mauvais pour le suédois.

2) Les différentes difficultés

Comme précisé précédemment, il y a quatre niveaux de difficultés utilisant un algorithme alpha-bêta :

- Facile : correspond à un jeu équilibré de profondeur quatre.
- Défensif : correspond à un jeu défensif de profondeur cinq : il évitera à tout prix de sacrifier des pions pour en capturer d'autres grâce à des valeurs d'évaluation différentes.
- Agressif : correspond à un jeu agressif de profondeur cinq : il tentera à tout prix de capturer des pions adverses, même si cela lui en fait perdre. S'il est suédois, il cherchera aussi à amener son roi vers les côtés puis les coins.
- Difficile : correspond à un jeu équilibré de profondeur six. S'il est suédois, il cherchera à gagner en déplaçant le roi vers les côtés.

3) Pistes abandonnées

Nous avions pensé à ajouter un algorithme A* dans la fonction evalCoup pour évaluer en combien de coup le roi pouvais quitter le plateau en prenant en compte la position des pions sur le plateau contrairement à notre fonction actuelle. Mais après quelques tests nous nous sommes rendus compte que la fonction evalFinal était beaucoup trop appelée pour cela : à l'époque, elle était appelée plus de 20000 fois dans un algorithme minimax de profondeur 5.
Nous avons aussi essayé d'évaluer la protection conférée par la mise côte-a-côte de plusieurs pions, mais nous n'avons pas réussis à avoir un résultat satisfaisant donc nous l'avons enlevé.