**Rapport Affrontement IA**

Ce projet a pour but de réaliser une IA imbattable au jeu du morpion 12x12 avec alignement de 4 jetons. L’algorithme d’intelligence artificielle utilisé est le Minimax avec un élagage alpha-bêta.

Remarque : des commentaires sont aussi présents dans le code pour expliquer son fonctionnement

***Détails sur l’implémentation du code :***

Tout d’abord on implémente une classe **Morpion** pour manier plus facilement la grille en définissant la conversion en string pour l’affichage.  La grille est initialisée avec que des None.

Pour l’algorithme Minimax il faut définir la fonction **partie\_terminee** (Terminal\_Test dans le cours) qui renvoie un booléen qui représente l’état de la partie. Cette fonction est décomposée en 4 trois fonctions :

* **test\_ligne** qui vérifie si 4 jetons identiques consécutifs sont présents sur une des lignes
* **test\_colonne** qui vérifie si 4 jetons identiques consécutifs sont présents sur une des colonnes
* **test\_diagonale**qui vérifie si 4 jetons identiques consécutifs sont présents sur les diagonales de gauche à droite dans la grille. Pour vérifier l’autre série de diagonale, les diagonales de droite à gauche, on exécute la même fonction en lui passant la grille qu’on a tourné à 90 degrés
* **grille\_remplie**qui renvoie True si toutes les cases de la grille sont remplies

Dans le cas où l’une de ces fonctions renvoie True, la partie est donc terminée.

L’implémentation de l’algorithme Minimax avec un élagage alpha bêta est classique mais on a ajouté plusieurs heuristiques pour limiter le nombre de cases de la grille testé par l’algorithme.

On commence par déterminer les cases vides avec la fonction **cases\_jouables** puis on fait un tri avec **cases\_jouables\_interessantes** pour ne prendre que les cases dites « intéressantes » parmi les cases jouables. Une case « intéressante » est une case qui possède un voisin (une case est voisine si elle fait partie des 8 cases autour) du type spécifié en paramètre (ordinateur / joueur). Dans le cas où l’IA (l’ordinateur) commence, les cases « intéressantes » sont toutes les cases autour des pions déjà posés de l’IA. Dans le cas où c’est le joueur qui commence, les cases « intéressantes » sont toutes les cases autour des pions de la grille. Cette différence permet d’ajuster le mode de jeu de l’IA si elle commence ou non. Si elle commence, l’IA jouera en mode « attaque », ses coups seront obligatoirement voisins d’un autre de ses pions. Sinon, elle jouera en mode « défense », ses coups peuvent aussi être voisin d’un pion du joueur adverse.

Pour des questions de temps de calcul, les cases jouables intéressantes sélectionnées sont ensuite mélangées aléatoirement. En effet, parcourir les cases dans le sens de parcours de la grille (ligne par ligne) n’est pas forcément optimal étant donné que les pions sont souvent centrés dans la grille.

Ensuite il faut définir la fonction **evaluation\_grille** (Utility dans le cours). Il faut choisir comment noter une grille. L’idée de base était qu’elle renvoie 1 si la grille est gagnante pour l’IA et moins 1 si elle est perdante et 0 sinon. Cependant, en faisant comme cela, la profondeur de calcul du Minimax doit être grande pour assurer un bon coup et l’élagage alpha bêta n’est pas très efficace.    
On opte donc pour une autre solution. On va compter le nombre de coup critique, c’est à dire le nombre de coup amenant à une grille gagnante après un seul coup de l’ordinateur ou du joueur. Si la grille a plusieurs coup critique elle est mis encore plus en avant.   
Pour résumer, la fonction d’évaluation renvoie une note entre -1 et 1 de la grille qu'on lui envoie. Plus la note est proche de 1, plus la grille est à l'avantage de l'IA. Seulement 7 valeurs peuvent être renvoyé par la fonction :

- Les valeurs 1 et -1 indiquent que l’ordinateur / le joueur a gagné la partie

- Les valeurs 0.99 et -0.99 indiquent que l'ordinateur / le joueur possède sur la grille une suite de pions qui font qu'il gagnera forcément au prochain coup

- Les valeurs 0.80 et -0.80 indiquent que l'ordinateur / le joueur aura une occasion de gagner au prochain coup si l'adversaire ne remplit pas la case gagnante

- La valeur 0 indique que l'ordinateur et le joueur n'ont pas d'avantage particulier sur la grille

Pour la fonction **minimax,**on implémente une profondeur pour réduire le nombre de possibilités à évaluer et un élagage alpha-béta vu en cours qui permet de réduire le nombre de coup évalués sans détériorer la solution. Pour finir on implémente une fonction jeu qui regarde le minimax de chaque case intéressante.

***Difficultés rencontrées :***

* Même avec une profondeur faible, comme minimax prenait tout d’abord tous les coups possibles en compte, chaque coup était très long, c’est pour ça que le tri des coups  **cases\_jouables\_interessantes**s'est avéré cruciale pour réduire le temps d’exécution.

* Même avec cela le temps était de temps en temps plus autour de la minute que des 10 secondes donc on a optimisé en forçant l’ordinateur à jouer à un coup s'il est suffisamment intéressant sans calculer les autres coups possibles.