Equipe Lightning Strike – TD F
Morgane – Alban – Stéphane – Dimitri

Rapport Projet IA : Puissance 4

Fonctions :

Nous avons codé les fonctions nécessaires au bon fonctionnement du puissance 4. Ainsi, nous avons codé des fonctions :

- D'affichage de grille (Affichage) ;
- Permettant le changement de joueur (ChangementJoueur) ;
- Vérifiant la disponibilité de la colonne (PositionCorrecte) ;
- Donnant la liste d'actions possibles (Actions) ;
- Appliquant l'action donnée (Result) ;
- Clonant la grille (Clone : qui renvoie une copie de notre grille actuelle. Elle est utilisée dans Result, afin de ne pas altérer la grille actuelle.) ;
- Vérifiant si un joueur a gagné (JoueurGagnant) ;
- Mettant fin à la partie s'il y a victoire d'un des joueurs ou match nul (TerminalTest) ;
- Renvoyant la valeur d'une grille terminale (Utility).

Minimax Alpha-Beta :

Pour notre fonction Minimax Alpha-Beta (AlphaBetaDecision), nous avons adapté le pseudo-code donné en TD au langage python et à nos fonctions. Pour chaque action possible dans la grille, nous appelons notre fonction minValue2 (qui cherche la min-value avec l'élagage alpha-beta). Dans les fonctions minValue2 et maxValue2, on vérifie d'abord si la grille est terminale. Si elle l'est, alors on renvoie l'utility pondérée en fonction de la profondeur actuelle de l'algorithme : cela nous permet de valoriser les actions qui nous font gagner plus vite ou perdre moins vite. Si la grille n'est pas terminale mais que la profondeur dépasse notre profondeur maximale, alors on renvoie 0. Dans ce cas, on fera appel ultérieurement à notre heuristique ScoreQuadruplet. Sinon, l'algorithme continue normalement. Pour la sélection finale du coup, on renvoie l'action pour laquelle l'utility est maximale.

Choix d'heuristique(s) :

1- Premières idées

Pendant notre recherche d'heuristiques, nous avons pensé à un premier principe : une heuristique visant à éliminer dès le début des actions considérées comme « inutiles », dans notre cas quand le pion posé n'était pas à proximité d'autres pions. Cette heuristique était peu efficace, car elle réduisait trop l'espace de recherche et n'avais d'intérêt qu'au début de la partie. Mais surtout, elle était difficile à mettre en place au premier tour, lorsque la grille était vide : toutes les actions étaient alors considérées comme inutiles.

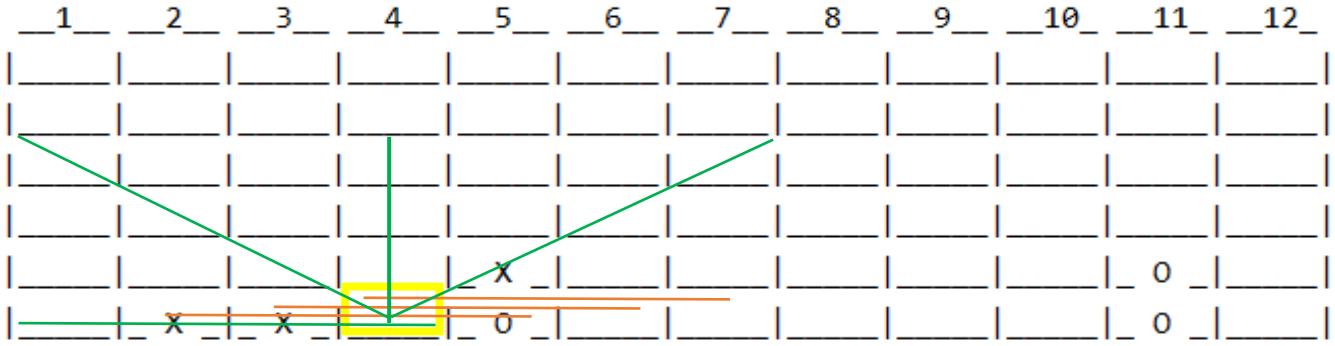
Par la suite, nous avons pensé au calcul de score grâce aux bitboards : l'idée était de superposer un tableau de poids à la grille du puissance 4, en faisant évoluer ces poids au fur et à mesure de l'avancement de la partie. Lorsque l'algorithme atteindrait la profondeur limite, on retournerait la somme de ces poids. Nous nous sommes vite rendu compte que transmettre et recalculer les tableaux de poids en parallèle du minValue2/maxValue2 allait beaucoup augmenter notre temps de calcul. Afin de résoudre ce problème nous avons pensé à calculer un score uniquement lorsque la profondeur maximale serait atteinte, sans faire de mises à jour successives. Cela nous a orienté vers un calcul avec des quadruplets. Cette heuristique nous semblait déjà plus adaptée, cependant la taille non conventionnelle de la grille rendait le temps de calcul assez long. Nous avons finalement choisi de calculer les scores dans la grille actuelle, à partir des actions immédiates.

2- Notre heuristique : ScoreQuadruplet

Lorsque la profondeur est dépassée, nous faisons appel à une heuristique nommée ScoreQuadruplet, qui prend en paramètres notre grille, un numéro de colonne (une action), ainsi que le joueur. Cette fonction a pour but de calculer le score d'une action, dans notre grille actuelle.

Nous calculons le score en nous basant sur le nombre de quadruplets dans lequel pourrait être impliqué le pion s'il était joué à cette position. Un quadruplet représente un alignement potentiel de 4 pions appartenant au même joueur. Ainsi nous prenons en compte les cases vides et les pions appartenant au joueur. Nous avons rajouté une pondération du score : si un pion de la même couleur est déjà présent dans le quadruplet, le poids est plus élevé ; si 2 pions de la même couleur sont déjà présents, le score est encore plus élevé…

Voici un exemple de calcul de score :



Ici, nous calculons le score de la case jaune (▭) du point de vue de notre IA (les pions X).

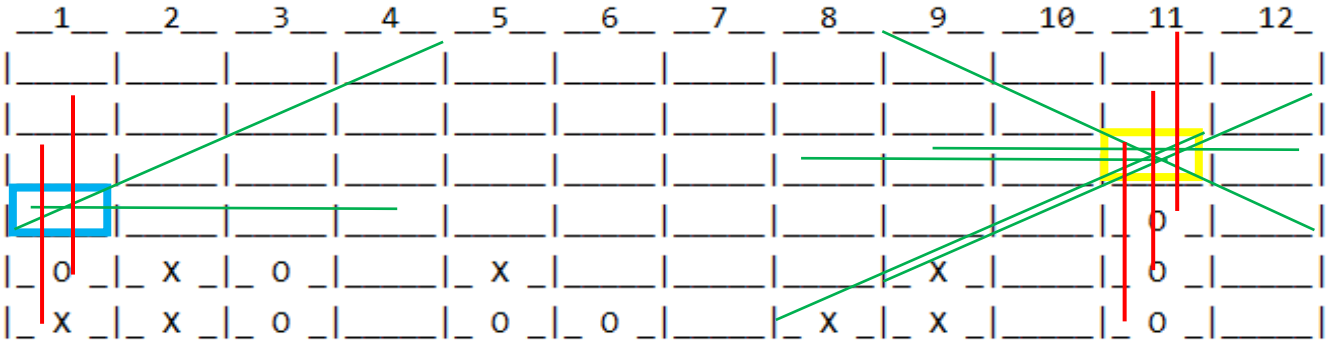
Les 7 quadruplets « possibles » sont représentés par des lignes : vertes si un alignement de 4 pions est possible, rouges si un alignement de 4 pions n'est pas possible (car on rencontre un pion adverse).

scoreQuadruplets = scoreQuadrupletsLigne(L)+ scoreQuadrupletsColonne(C)+ scoreQuadrupletsDiagGauche(DG)+ scoreQuadrupletsDiagDroite(DD)+ scoreNombreQuadruplets(N)

On a : (L) = 4 ((0+1+1+0)*2=4) ; (C) = 0 ((0+0+0+0)*2=0) ; DG = 0 ((0+0+0+0)*2=0) ;
(DD) = 2 ((0+1+0+0)*2=2) et (N) = 4 (les 4 lignes vertes)
D'où : scoreQuadruplets(▭) = 4+0+0+2+4=10

Notre fonction ScoreQuadruplet fait appel à 4 sous-fonctions : ScoreQuadrupletLigne, ScoreQuadrupletColonne, ScoreQuadrupletDiagGauche et ScoreQuadrupletDiagDroite. Ces fonctions prennent en paramètre la grille actuelle, et la ligne/colonne que le pion joué occuperait dans notre grille. Elles retournent un intervalle qui permettra à notre fonction ScoreQuadruplet d'ajuster le nombre et la position des quadruplets à prendre en compte (afin d'éviter les « index out of range »).

Voici un exemple où l'on peut voir que le nombre de quadruplets « possibles » diffère d'une case à l'autre, d'où l'intérêt de nos sous-fonctions.



Pour la case bleue ( ), il y a 4 quadruplets « possibles » (qui ne rentrent pas en conflit avec la taille de la grille): 2 quadruplets dont l'alignement de 4 pions X est possible (2 lignes vertes) et 2 quadruplets dont l'alignement de 4 pions n'est pas possible (2 lignes rouges).

Pour la case jaune ( ), il y a 8 quadruplets « possibles » (qui ne rentrent pas en conflit avec la taille de la grille): 5 quadruplets dont l'alignement de 4 pions X est possible (5 lignes vertes) et 3 quadruplets dont l'alignement de 4 pions n'est pas possible (3 lignes rouges).

Guide d'installation :

Nous n'avons recours à aucun package ou librairie supplémentaire dans notre code. Tout se fait sous python. Il n'y a donc pas de manipulation spéciale à réaliser avant le lancement du programme. Par ailleurs, notre IA jouera toujours avec les pions « X », qui correspondent aux cases « 1 » dans notre grille de puissance 4, tandis que le joueur adverse aura toujours les pions « O » qui correspondent aux cases « -1 ».

Pour lancer notre programme, nous exécutons la fonction Jeu_Du_Puissance4 qui prend en paramètre la profondeur max. Nous avons choisi une taille d'arbre égale à 4 car notre programme était suffisamment rapide pour rester performant à cette profondeur. Si nous avions choisi une profondeur max égale à 5, notre IA mettrait en moyenne 5 secondes de plus pour jouer, le temps d'exécution dépasserait donc parfois les 10 secondes autorisées.