Rapport Puissance 4

Librairies ajoutées :

* *Numpy* : nous avons préféré créer la grille du jeu avec Numpy plutôt qu’avec une liste de listes pour faciliter son traitement. Nous avons notamment utilisé la fonction np.diag(grille,k) pour extraire effciacement les diagonales de la grille.
* *Time* : pour calculer le temps que l’IA prend pour retourner un numero de colonne.
* *Math* : pour avoir les valeurs infinies grâce à math.inf
* *Copy* : pour réaliser des copies rpofondes de la grille grâce à copy.deepcopy()

Fonctions (dans l’ordre d’apparition)

* **Fonctions de vérification de victoire** :
  + *VerifH(grille, lignecoup, colonnecoup)* : retourne True si le dernier jeton joué apporte une victoire horizontale (en ligne), et une liste contenant les emplacements « testés » par Verif. C’est-à-dire que, si l’on n’a pas de victoire horizontale pour le pion qui vient d’être posé, il n’y en aura pas non plus en partant de chacun des pions de même couleurs et collés à ce dernier de façon horizontale. Cela sera utile dans la fonction TerminalTest(s)
  + *VerifV(grille, lignecoup, colonnecoup)* : idem en vertical (en colonne)
  + *VerifD1(grille, lignecoup, colonnecoup)* : idem en diagonale montante
  + *VerifD2(grille, lignecoup, colonnecoup)* : idem en diagonale descendante
  + *Verif(grille, lignecoup, colonnecoup) :* renvoie True si au moins l’une des vérification précédente retourne True
* **Fonctions de base qu’on a repris du morpion :**
  + *Actions(grille)* : retourne la liste des positions qu’un jeton peut prendre
  + *Result(grille, action\_choisie, joueur)* : met à jour la grille
  + *Terminal\_test(grille)* : retourne True si la grille correspond à une victoire ou à un match nul.
* **Fonctions heuristiques :**

Nous avons choisi de travailler avec des strings afin d’exploiter au mieux l’efficacité des fonctions built-in de python tout en permettant de spécifier une infinité de motifs complexes. Ainsi notre fonction Utility a l’avantage considérable d’être très facile à lire et à modifier.

* + *ValMotifs(chainedecaractères, m, l)*: valorise et dévalorise certains motifs d’agencement de jetons pour les lignes et les diagonales. Elle pren den entréer une chaine de caractères qui correspond soit à une ligne, soit à une diagonale et teste si des motifs exitent dedans.
  + *ValMotifsCol(chainedecaractères, m, l)*: même chose mais adapté aux motifs en colonne qui sont contraints par la gravité.
  + *Utility(grille) :* retourne un entier correspondant à la valeur de la grille (plus elle est élevée, plus cette grille apporte(ra) une victoire). Elle appelle ValMotif et ValmotifCol en ayant converti au préalable une ligne, colonne ou diagonale en chaine de caractères.
* **Fonctions d’affichage :**
  + *remplirGrille(joueur, jeu) :* fonction de M.Chendeb
  + *printGrille()* : idem
* **Fonction IA :**
* **Minimax(position, profondeur, alpha, beta, booléen\_tour\_max) :** réalise l’algorithme minmax avec élagage alpha-bêta, retourne l’action considérée optimale par l’IA. Elle intègre une limitation en profondeur (definie par l’argument éponyme). Les variables sont nommées en référence à la position dans l’arbre de recherche. Ainsi *position* est la grille en l’état actuel et *enfant* correspond à un des noeuds fils de cette position.
* **TerminalTest(s) :**

Fonction permettant de vérifier si une grille est « terminée ». Dans le cas ou 36 tours ont été joués. Ou bien dans le cas où 4 pions identiques sont alignés, horizontalement, diagonalement, ou encore verticalement. Pour ce faire, elle appelle les différentes fonctions de vérification (VerifD1, VerifD2, VerifH, VerifV) et les appliques sur chacune des cases non-vides. Mais, de manière a optimiser ce processus, elle utilise les listes renvoyées par chacune des fonctions de vérification des pions « plus à vérifier » pour éviter de vérifier inutilement des pions.

* **Tentatives d’amélioration :**

De manière à pouvoir utiliser notre minimax à une plus grande profondeur, nous avons tentés d’implémenter le Null Window Pruning, ainsi que le Principal Variation Search, sans succès. En effet, la complexité de la version finale de notre minimax a rendu très difficile l’adaptation de ces algorithmes a notre minimax.

Nous avons cependant réordonné le sens de recherche de notre minimax (du centre vers l’extérieur) de manière a optimiser l’efficacité de l’élagage alpha-beta.

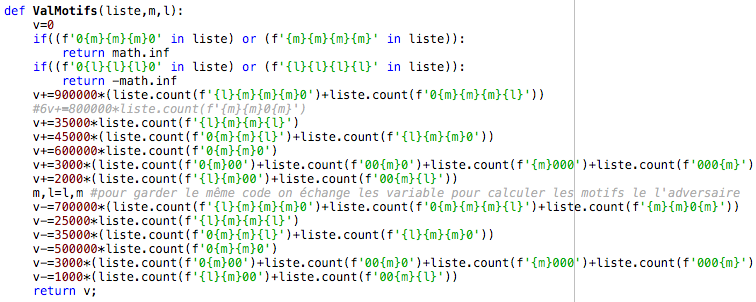
* **Fonctions de jeu** :
  + Monjeu(grille) : demande à l’IA de retourner un numéro de colonne
  + AppliqueJeuAdv(grille,jeu) : adapte le jeu de l’adversaire au format de la grille tel que l’IA puisse la traiter

Heuristiques :

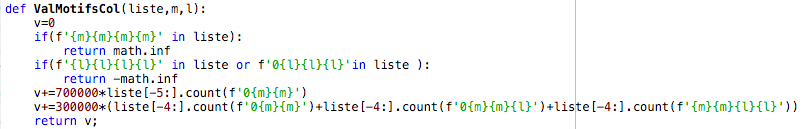
Nous avons été insirés par un papier de recherche auquel nous devons le crédit :

*Kang, Xiyu & Wang, Yiqi & Hu, Yanrui. (2019). Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game. Journal of Intelligent Learning Systems and Applications. 11. 15-31. 10.4236/jilsa.2019.112002.*

Nous avons pris pour base de recherche la bonification qu’ils proposent en évaluant les motifs de la grille. Nous avons décidé de prendre en compte par ailleurs le jeu de l’aversaire mais dans une moindre mesure : à motif équivalent l’IA préfèrera attaquer que défendre ( ce dans l’intérêt du fair-play) .



* Les 4 premiers cas assurent / sont des cas de victoire ou d’échec. Ils sont donc affectés au valeurs les plus extrêmes : ±∞.
* Le bloc suivant correspond aux motifs que l’on souhaite former i.e. des motifs qui mèneront à la victoire. Les motifs se voient attribuer une valeur v plus ou moins grande en fonction de leur proximité avec la victoire. On choisit des motifs de 4 éléments au minimum pour éviter les répétitions
* Le dernier bloc suit la même logique : ce sont cette fois les motifs qu’on ne souhaite pas donc on va faire diminuer la valeur v. Ce sont les mêmes motifs que précédemment car dans ce bloc on se met à la place de l’adversaire (d’où le changement de variables).  
  Notez que les valeurs attribuées sont respectivement moins grandes en valeur absolue que celles du bloc précédent pour forcer l’IA à préférer l’attaque au blocage.



* Lorsque l’on traite les colonnes, la gravité limite grandement les motifs gagnants et perdants. On prendra donc les 4 ou 5 derniers éléments de la colonne (en partant du haut de la grille) et cela suffira à déterminer la valeur du motif.
* On suit la même logique que précédemment.