M1 - MIASHS IC Novembre 2021

Rapport - Intelligence Artificielle

Projet de Puissance 4

RUFFIEUX Mathis, GIRARDIER Nicolas, NCUBE Nozithelo

Sommaire

I) Le jeu puissance 4

1) Règles du jeu

Puissance 4 est un jeu de stratégie conçu pour deux joueurs, sur un plateau comprenant 6 lignes et 7 colonnes. Chaque joueur dépose successivement un jeton de sa couleur dans une des colonnes du plateau et le jeton tombe dans la case vide la plus basse de cette colonne.

Le premier joueur qui réalise un alignement de quatre jetons de sa couleur gagne :
- verticalement,
- horizontalement,
- ou en diagonale.

Si le plateau est entièrement rempli sans alignement alors c'est un match nul.

2) **Lancement du jeu** (classes GameFrame et MenuPanel)

La fonction “Main” du programme se trouve dans la classe du jeu Puissance4. Elle crée une instance du jeu GameFrame en héritant de la classe JFrame, afin d’afficher une interface graphique.
Depuis cette classe, on instancie dans un premier temps une classe MenuPanel, qui va afficher les options du jeu en y ajoutant des éléments graphiques. *(Cf. Annexe 1)*

Ici trois options se présentent à nous :
- Player VS Player
- Player VS AI
- AI VS AI

Lorsqu’on choisit de jouer contre une IA, deux nouveaux boutons de type JComboBox sont créés. *(Cf. Annexe 2)*

Le premier bouton permet de choisir une implémentation de l’IA :
- soit en utilisant Minimax (correspondant à la classe AI1.java)
- soit utilisant Minimax + la coupe AlphaBeta (correspondant à la classe AI2.java)

Le second bouton permet de choisir la profondeur (Depth) de l’algorithme Minimax.

Une fois les choix du jeu effectués, on instancie le jeu depuis la classe GameFrame avec une nouvelle classe GamePanel.
Cette classe va générer un terrain ainsi que toutes les règles du jeu.

3) **Le terrain et l'interface graphique** (classes GamePanel et Terrain)

Depuis la classe GamePanel, on récupère toutes les variables initialisées précédemment afin de définir le type de partie (Player VS Player, Player VS AI, AI VS AI) et d'autres paramètres, tel que les dimensions du terrain (6 lignes pour 7 colonnes), la taille de la fenêtre (Height, Width) ou encore la taille d'une case en pixels (Unit_size).

Par ailleurs, on initialise les IAs appelées "Tron" (en référence au film Tron l'héritage). Tron2 implémente Minimax et Tron3 implémente Minimax + la coupe Alphabeta.

La fonction *void setButtons* permet d'afficher des boutons au-dessus de chaque colonne, afin que le joueur puisse cliquer, en appelant la fonction *actionPerformed*.

Lorsque le jeu est terminé (si un joueur est victorieux), ces boutons seront grisés par la fonction *void greyButtons* pour pas qu'on ne puisse y ajouter des jetons. *(Cf. Annexe 3)*

La fonction *draw* permet d'afficher graphiquement l'état le terrain actuel à chaque coup des joueurs.

La fonction *void écrit* permet d'écrire le tour du joueur (Player 1, Player 2, AI turn) en dessous du terrain. Lorsqu'il y a une victoire, cette fonction écrit le nom du joueur victorieux (ou de l'IA).

Dans la classe Terrain sont implémentées les règles du jeu.
Le constructeur de classe permet de créer un terrain vide rempli de '0' de taille 6 par 7.

Ce terrain est lu à chaque tour par la fonction *void dessine* pour en interpréter les jetons.

- la case 0 correspond à une case vide
- la case 1 correspond à un jeton orange
- la case 2 correspond à un jeton rouge
- la case 3 correspond à un jeton bleu

La fonction *ajoutPiece* permet d'ajouter un jeton dans une colonne du tableau.
Attention ! On vérifie toujours au préalable que la colonne n'est pas pleine avant d'ajouter un jeton. En effet, si on essaye d'ajouter un jeton sur une colonne pleine, il ne se passe rien.

Après l'ajout de chaque jeton, on vérifie s' il y a une victoire avec la fonction *int testVictoire*, qui renvoie 1 si le premier joueur gagne, 2 si le second joueur gagne et 0 sinon.
Pour vérifier une victoire, on parcourt l'ensemble du tableau et on vérifie pour chaque case, si ses trois cases adjacentes sont toutes égales entre elles.

On vérifie donc une potentielle adjacence en :

- ligne
- colonne
- diagonale (de haut à gauche en bas à droite)
- diagonale (de haut à droite en bas à gauche)

Lorsqu'un joueur est victorieux, on affiche un message approprié et on grise les boutons. *(Cf. Annexe 3)*
De plus, les jetons "**victorieux**" prennent une valeur négative dans le tableau Terrain.

Par exemple les jetons 3 (color blue) prennent la valeure -3 (color cyan)
les jetons 2 (red) -> -2 (magenta)
et les jetons 1 (orange) -> -1 (yellow)

II) L'implémentation des IAs

1) La classe tools partagée par les IAs

La classe *AI_tools* est une classe partagée entre les différentes IAs. Elle sera appelée en tant qu'objet pour pouvoir utiliser ses méthodes.

La fonction *boolean isPair* permet de savoir si un entier est pair ou impair. Elle renvoie TRUE si pair et FALSE sinon. Cela permet dans Minimax de savoir si on est au tour de l'IA ou du joueur en comptant le nombre de coups joués, car ils jouent l'un après l'autre.

La fonction *int[][] copyTerrain* permet de copier un tableau représentant le terrain dans un nouveau tableau, afin de ne pas écrire dans un même tableau à chaque appel récursif de l'algorithme Minimax.

La fonction *boolean colonneIsNotFull* permet de savoir si une colonne du terrain est pleine ou non. Pour cela, on va vérifier que le premier indice de la colonne est vide (donc égal à '0').

La fonction *boolean estVictorieux* permet de savoir si un coup est victorieux. Pour cela la fonction teste si en ajoutant un jeton dans une colonne donnée, il y a une victoire en largeur, en hauteur ou en diagonale.
Elle renvoie TRUE si une victoire est détectée et FALSE sinon.

La fonction *int[][] addJeton* permet d'ajouter un jeton dans le terrain dans une colonne donnée. Elle renvoie un terrain avec le jeton ajouté.
Attention : On ne peut pas ajouter un jeton dans une colonne déjà pleine, auquel cas un message d'erreur est affiché.

La fonction *void afficheTable* permet d'afficher un tableau représentant le terrain dans la console, afin d'observer les étapes de Minimax, elle peut être appelée n'importe où dans l'algorithme.

La fonction *int[] changeCoup* permet de changer l'indice de la colonne dans laquelle une IA souhaite jouer si cette colonne est pleine, en choisissant une autre colonne non pleine. Cette fonction est une mesure de prévention.

2) **Implémentation de Minimax** (classe AI2)

Dans notre programme, nous avons trois classes pour trois IAs différentes :

- La classe *AI0* est une première implémentation de Minimax, qui fonctionne mais nous ne l'utilisons pas ici car *AI1* est plus optimale.
- La classe *AI1* est une implémentation de Minimax.
- La classe *AI2* implémente Minimax et une coupe AlphaBeta.

Les classes *AI1* et *AI2* utilisent toutes deux des fonctions partagées de la classe *AI_Tools*.

L'algorithme de Minimax est un algorithme qui s'applique à la théorie des jeux consistant à minimiser les pertes pour un jeu à deux joueurs jouant l'un contre l'autre.
Il peut être représenté par un arbre avec comme racine l'état actuel du jeu. Pour le jeu puissance 4, à chaque nœud de l'arbre un joueur dispose de 7 possibilités car il y a 7 colonnes (en excluant les colonnes déjà pleines).

Chaque nœud de l'arbre représente un état du terrain. L'algorithme va chercher à jouer le meilleur coup possible parmi les 7 possibilités afin de maximiser son score défini par un heuristique.
Dans notre cas, l'heuristique est définie par une victoire du joueur. Lorsqu'une victoire sera détectée par notre algorithme Minimax, le score maximal du joueur prendra la valeur de la hauteur du terrain multipliée par sa largeur (6*7), moins le nombre de coups joués et le tout divisé par deux.

*Score de victoire = (hauteur * largeur - nbPlays) / 2*

Pour notre cas, nous avons implémenté dans la classe *AI1* du programme une variante de Minimax appelée **Negamax**. Cette variante est basée sur les propriétés de jeu à somme nulle avec deux joueurs.
Ainsi, la valeur du score du premier joueur (c'est à dire notre IA) est l'inverse de la valeur du score du joueur adverse. De cette manière nous pouvons effectuer un seul calcul pour trouver le Min et le Max dans l'arbre qui représente Minimax, en prenant son inverse de manière récursive :

- *scoreChoix = minMax(newTab, nbPlays-1, profondeur-1);*
- *scoreChoix[0]= -scoreChoix[0];*

Ici le tableau scoreChoix possède deux valeurs. En position *scoreChoix[0]* se trouve le score représenté par l'heuristique de Minimax et en position *scoreChoix[1]* se trouve son choix, c'est-à-dire l'indice de la colonne dans laquelle l'IA devra jouer son coup.

Notre IA, représentée par la classe *AI1*, peut jouer un coup avec sa fonction *void play()*. Cette fonction appelle donc Minimax qui va s'appeler récursivement jusqu'à atteindre la profondeur souhaitée.

Lors de l'appelle de Minimax, on vérifie d'abord si le terrain est rempli, auquel cas on renvoie un score nul.

Ensuite, on parcourt l'ensemble des colonnes en y ajoutant un jeton de manière virtuelle afin d'observer si une victoire est détectée.

Si aucune victoire n'est détectée et à condition que la profondeur souhaitée ne soit pas déjà atteinte, on poursuit l'algorithme en appelant de manière récursive notre fonction *int[] minMax()* sur chacune des colonnes qui ne sont pas déjà pleines.
Ensuite, en utilisant le variant Negamax on prend l'inverse du score, afin de ne pas maximiser mais de minimiser son score. On retourne ensuite le meilleur choix trouvé parmi les 7 appels récursifs de la fonction *int[] minMax()*.

Une fois cette fonction terminée, dans la fonction *void play()*, l'ia joue son jeton en appelant la fonction *void ajoutPiece()* de la classe terrain et renseigne l'indice de la colonne dans laquelle elle souhaite jouer.

3) **Coupe AlphaBeta** (classe AI3)

La classe *AI2* reprend le même algorithme de minimax que dans la classe *AI1* en y ajoutant une coupe AlphaBeta.
Cette coupe sert essentiellement à optimiser l'algorithme, lui permettant d'ignorer les branches de l'arbre qui ne sont pas considérées comme utiles, c'est-à-dire celles qui de toute façon ne seront pas remontées jusqu'à la racine.

Dans notre cas, nous initialisons Alpha avec une très petite valeur (-2^10) et Beta avec une très grande valeur (2^10).

De la même manière que précédemment, nous allons d'abord vérifier que le terrain n'est pas rempli et qu'une victoire n'est pas détectée avant d'appeler récursivement la fonction *int[] minMax()* sur chacune des colonnes non-remplies.

Dans chaque appel récursif, on initialise une variable Max qui prendra la valeur de notre heuristique en cas de victoire :

- *Max = (hauteur * largeur - nbPlays) / 2;*

Si cette valeur est plus petite que Beta alors Beta prendra la valeur de Max afin d'être initialisée à la borne appropriée au nœud dans lequel on se trouve.
Si Alpha (qui est la borne maximale) est plus grand ou égal à Beta (qui est la borne minimale) alors on ne fera pas d'appel récursif car l'ensemble des solutions est nul, **on coupe la branche**.

Par ailleurs, dans notre variante appelée Negamax, les valeurs d'Alpha et de Beta sont inversées d'une récursivité à l'autre afin de conserver les bornes supérieures et inférieures.

C'est-à-dire que le paramètre -Beta devient Alpha et -Alpha devient Beta :

- *scoreChoix = minMax(...**,-beta, -alpha**);*

Est un appel récursif de :

- *minMax(..., **int alpha, int beta**){...}*

Pour calculer le score d'une fonction récursive, nous avons trois possibilités :

- Si le score perçu est compris entre l'intervalle de la borne Min et Max, on retourne le score exact.
- Si le score est inférieur à Alpha (borne maximale), on peut retourner un score supérieur.
- Si le score est supérieur à Beta (borne minimale), on peut retourner un score inférieur.

4) Comparaison des temps d'exécutions avec un timer

Afin d'observer le temps d'exécution des IAs, nous avons implémenté un Timer dans la classe GamePanel.
Le timer est initialisé avec la fonction *Go_Chrono()* avant l'appel de la méthode Play des IAs, puis stoppé avec la fonction *Stop_Chrono()* une fois que l'IA à joué son coup. Le résultat est affiché dans la console.

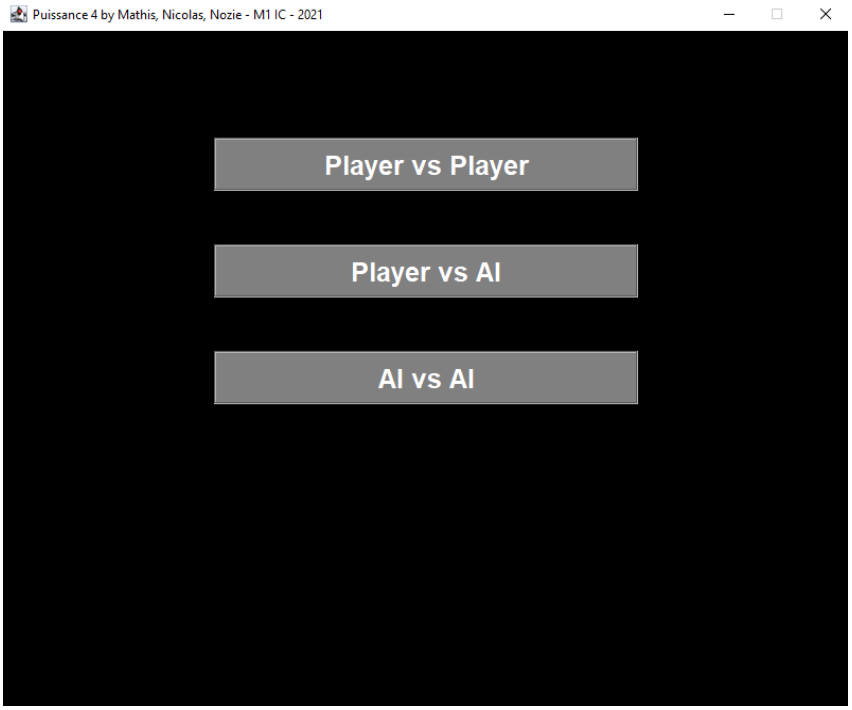
Ainsi, nous avons pu comparer le temps d'exécutions entre l'IA numéro 1 et l'IA numéro 2 en fonction de la profondeur de Minimax. *(Cf. Annexe 4 et 5)*

Ces données nous montrent une grande différence d'efficacité de l'IA implémentant seulement Minimax par rapport à celle implémentant aussi une coupe Alpha beta.
En effet à partir d'une profondeur de 9, l'IA numéro 1 demande déjà plus d'une minute pour s'exécuter, tandis que l'IA numéro 2 ne met que 80 millisecondes pour effectuer les calculs de la même profondeur. La coupe Alpha Beta est donc très efficace.

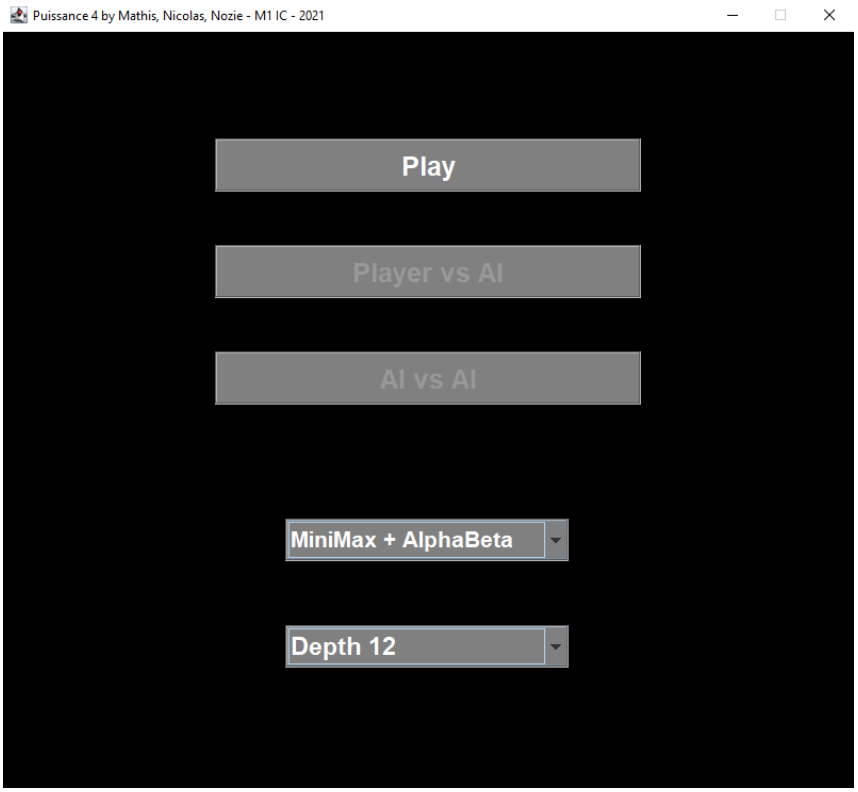
Par ailleurs, une observation intéressante sur l'algorithme implémentant la coupe Alpha Beta est que le temps d'exécution est plus faible lorsque la profondeur est pair que lorsque la profondeur est impair.
Par exemple avec une profondeur de 15, le temps d'exécution est supérieur à 20 secondes, alors qu'avec une profondeur de 16, il n'est que de 8 secondes. Lorsqu'on exécute une profondeur de 17, on retrouve à nouveau un temps d'exécution important.
On peut ainsi tracer deux courbes distinctes sur une échelle logarithmique. *(Cf. Annexe 6)*

III) Annexes

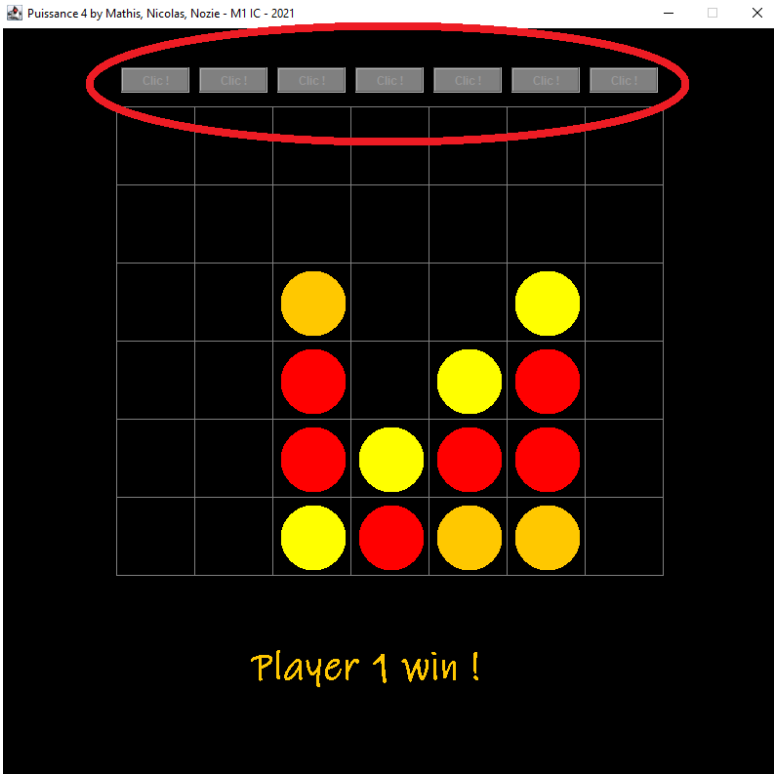
Annexe 1 : Menu du jeu



Annexe 2 : Choix de MiniMax et Depth dans le menu



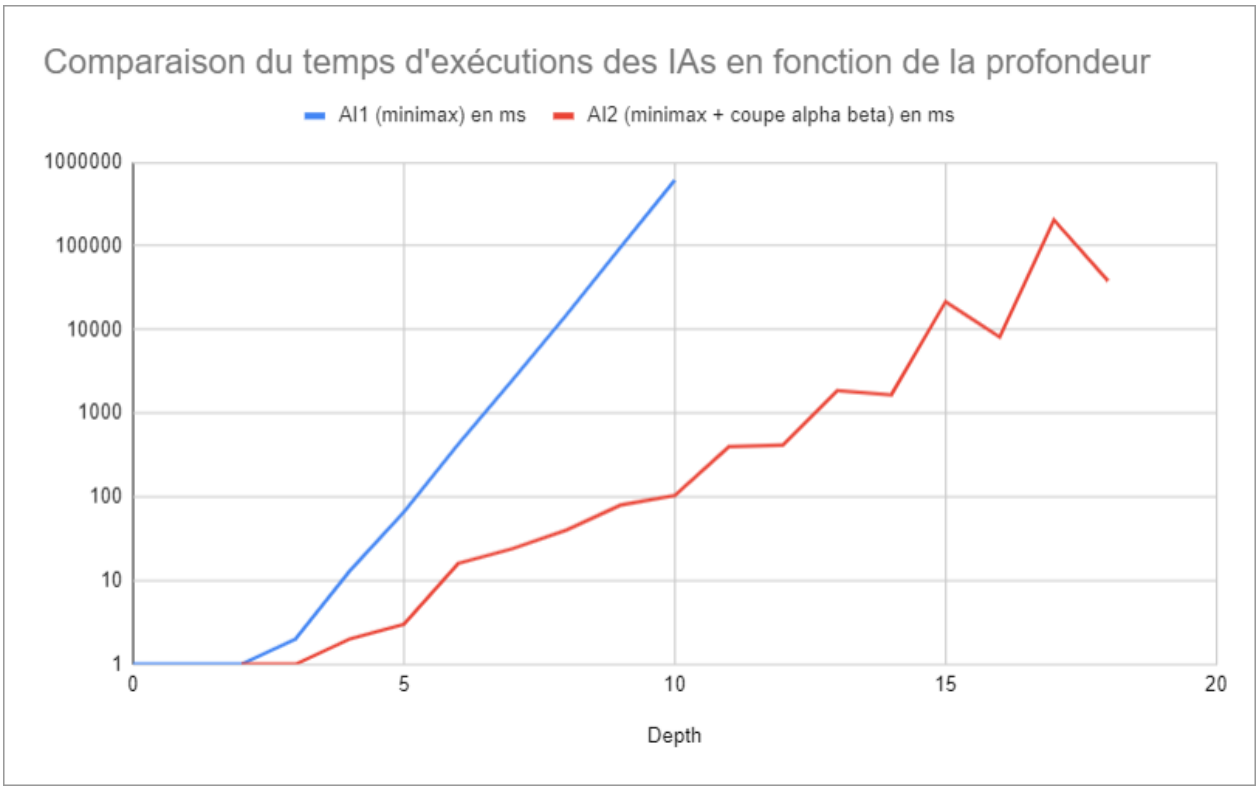
Annexe 3 : Les boutons “Clic !” (grisés)



Annexe 4 : Temps d’exécution des IAs (en millisecondes)

Depth	AI1 (minimax)	AI2 (minimax + coupe alpha beta)
0	1ms	0ms
1	1ms	0ms
2	1ms	1ms
3	2ms	1ms
4	13ms	2ms
5	66ms	3ms
6	425ms	16ms
7	2491ms	24ms
8	14940ms	40ms
9	96073ms	80ms
10	609605ms	104ms
11	-	400ms
12	-	417ms
13	-	1868ms
14	-	1664ms
15	-	21590ms
16	-	8186ms
17	-	204011ms
18	-	38471ms
19	-	-
20	-	-

Annexe 5 : Tableau de comparaison du temps d'exécution des IAs en ms (Échelle logarithmique)



Annexe 6 : Tableau de comparaison du temps d'exécution pour l'IA numéro 2, profondeur pair et impair (Échelle logarithmique)