Projet IA: Implémentation du Morpion 4x4 en utilisant plusieurs techniques d'intelligence artificielle et évaluation de leur performance.

Binôme: Pauline TEOULLE et Gabriel BARNEAUD, M1 WIC

1. Introduction

Pour répondre aux demandes du projet, nous avons décidé d'implémenter le jeu du Morpion sur une grille de 4 lignes et 4 colonnes. Ce jeu s'inspire donc du Morpion classique qui se joue sur une grille 3 lignes et 3 colonnes. C'est un jeu de stratégie entre deux joueurs qui sont représentés par des pions de couleurs (la couleur rouge correspond au joueur 1 et la couleur bleu correspond au joueur 2). A chaque tour, le joueur doit placer un pion et son but sera d'en aligner quatre. La partie est terminée si un des joueurs gagne ou s'il y a égalité, c'est-à-dire que la grille est remplie et qu'il n'y a aucun alignement de quatre pions.

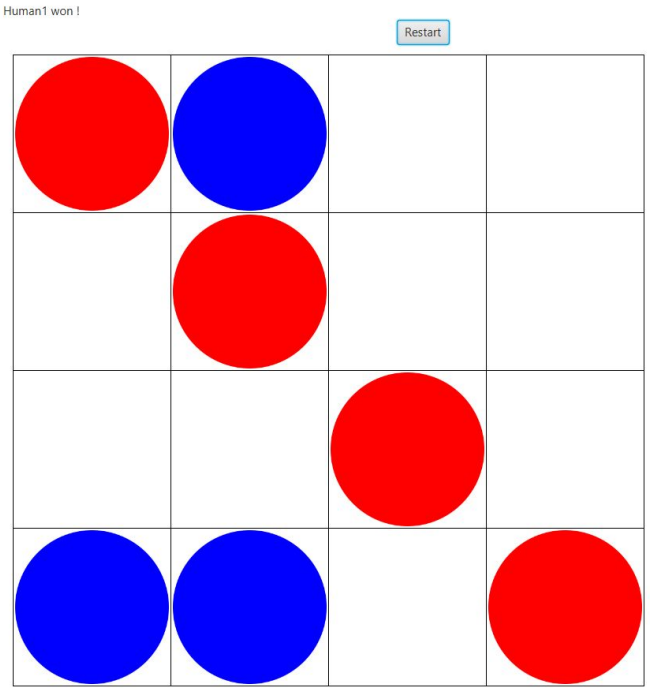

Figure 1: Interface du Morpion 4x4 et victoire du joueur 1 contre un autre joueur.

Le projet est entièrement codé en Java et la partie graphique est programmée en JavaFX. Le code aura donc une partie concernant l'affichage graphique et une autre partie qui concerne tous les algorithmes utilisés pour l'implémentation de l'IA.

2. Implémentation et évaluation

2.1 Humain vs Humain

Ce mode de jeu est la version classique d'un Morpion 4x4 entre deux joueurs humains.
Il permet la mise en place du gameplay général comme la génération de la grille, la récupération des coordonnées de la souris du joueur et la vérification de celles-ci (vérifier si les coordonnées sont bien dans la grille et qu'on ne joue pas sur une case déjà occupée), l'affichage du pion sur la grille lors d'un tour de jeu, le changement du tour de jeu, et la fin de la partie.

La gameplay est le suivant: les joueurs posent leur pion sur une case disponible dans la grille chacun leur tour. Si un joueur arrive à aligner quatre pions, il gagne. Si la grille est remplie et que personne n'a réussi à aligner quatre pions, il y a un égalité.

2.2 Humain vs IA Random

Nous avons réalisé ce mode de jeu de façon à ce que l'intelligence artificielle choisisse une case jouable aléatoirement parmi la liste des cases vides.
A la fin du tour de l'humain, l'IA crée une liste des cases disponibles pour placer son pion. Elle choisit aléatoirement une case de la liste et joue sur celle-ci.

L'heuristique de cette IA donne le même poids pour chaque case disponible (car toutes les cases possibles sont ajoutées à une liste), donc aucune n'est privilégiée.

L'IA Random est imprévisible. Son évaluation du jeu n'est pas satisfaisante car le fait de contrer l'adversaire ou gagner contre lui est relié à un facteur aléatoire (et donc un facteur de chance).

Le tour de jeu est très rapide car l'IA ne calcule rien pour choisir une bonne case.
Lors de son tour, l'IA met environ 0.002 seconde pour jouer.

2.3 Humain vs IA MinMax

Nous avons réalisé ce mode de jeu de façon à ce que le joueur puisse jouer contre une intelligence artificielle plus intelligente. En effet, celle-ci choisit de poser son pion sur une case jouable qui bloque l'adversaire quand il est sur le point de gagner avec l'algorithme MinMax ou alors d'aligner quatre pions.

MinMax est un algorithme qui consiste à jouer le meilleur coup en fonction des coups possibles de l'adversaire.
L'IA parcourt tous les coups pour une profondeur donnée dans un arbre des possibilités et donne un poids à chacune d'elles en fonction du nombre de possibilités de gains pour le joueur et pour l'adversaire.
Plus le poids est élevé, meilleure est la case. L'IA cherche à choisir la case avec le poids le plus élevé en considérant que l'adversaire veut minimiser les poids de l'IA.

L’heuristique que nous avions implémentée au début pour donner les poids était mauvaise. En effet, elle ne prenait pas en compte le nombre de pions joués participants aux gains potentiels mais uniquement le nombre de gains potentiels sans distinction.

Nous avons donc modifié cela de façon à avoir une IA plus efficace et plus intelligente.

L'heuristique finale est la suivante:

Soient IA et Adversaire deux entiers qui correspondent au poids des cases possibles tels que

IA = □ (gain possible pour l’IA) * (nombre de coups joués participant à ce gain possible pour l’IA)

Adversaire = □ (gain possible pour l’adversaire) * (nombre de coups joués participant à ce gain possible pour l’adversaire)

Heuristique = IA - Adversaire

Evaluation de l’IA MinMax:

- Plus la profondeur de l’arbre des possibilités est élevée, plus l’algorithme devra donner des poids pour un nombre de coups élevés. La profondeur influence le temps de calcul des poids: plus elle est élevée, plus le temps de calcul est long.
- Plus la partie avance, moins il y a de poids à calculer, plus le temps de calcul se réduit.

Par exemple:

- Pour une profondeur de 3, le temps de calcul est d’environ 0.021 secondes pour le premier tour de l’IA MinMax et d’environ 0.007 secondes pour le deuxième.
- Pour une profondeur de 7, le temps de calcul est d’environ 52.133 secondes pour le premier tour de l’IA MinMax et d’environ 9.038 secondes pour le deuxième.

2.4 Humain vs IA MinMax AlphaBeta

Nous avons réalisé ce mode de jeu de façon à ce que le joueur puisse jouer contre une intelligence artificielle se basant sur l’IA MinMax mais jouant beaucoup plus rapidement. En effet, celle-ci choisit de poser son pion sur une case jouable qui bloque l’adversaire quand il est sur le point de gagner avec l’algorithme MinMax AlphaBeta ou alors d’aligner quatre pions.

L’algorithme MinMax AlphaBeta se base sur l’algorithme MinMax vu précédemment mais utilise l’élagage alpha-bêta en plus. C’est un moyen de réduire le nombre de nœuds évalués.
Il permet d’optimiser le calcul de poids en explorant l’arbre de possibilités de façon partielle et en évitant les calculs de poids inutiles.

En effet, il arrête d’évaluer un mouvement lorsqu’il trouve une possibilité qui est pire qu’un mouvement précédemment exploré.

Cet algorithme renvoie le même mouvement que MinMax mais ignore les branches ne pouvant pas influencer le résultat final. Il est donc beaucoup plus rapide que MinMax sans élagage.

Evaluation de l'IA MinMax AlphaBeta:

- Plus la profondeur de l'arbre des possibilités est élevée, plus l'algorithme devra donner des poids pour un nombre de coups élevé mais dans une moindre mesure comparé à l'algorithme MinMax. La profondeur influence le temps de calcul des poids: plus elle est élevée, plus le temps de calcul est long.
- Plus la partie avance, moins il y a de poids à calculer, plus le temps de calcul se réduit.

Par exemple:

- Pour une profondeur de 3, le temps de calcul est d'environ 0.007 secondes pour le premier tour de l'IA MinMax AlphaBeta et d'environ 0.001 secondes pour le deuxième.
- Pour une profondeur de 7, le temps de calcul est d'environ 1.258 secondes pour le premier tour de l'IA MinMax AlphaBeta et d'environ 0.025 secondes pour le deuxième.

2.5 IA MinMax vs IA MinMaxAlphaBeta

Nous avons réalisé ce mode de jeu de façon à ce que deux IA qui ont la même heuristique puissent jouer l'une contre l'autre. La différence majeure entre les IA est l'algorithme utilisé pour placer un coup. Une IA utilise MinMax de façon simple alors que l'autre utilise MinMax AlphaBeta qui permet de réduire considérablement le temps de calcul grâce au système d'élagage.

Evaluation de l'IA MinMax AlphaBeta:

- Plus la profondeur de l'arbre des possibilités est élevée, plus l'algorithme devra donner des poids pour un nombre de coups élevé mais dans une moindre mesure comparé à l'algorithme MinMax. La profondeur influence le temps de calcul des poids: plus elle est élevée, plus le temps de calcul est long.
- Plus la partie avance, moins il y a de poids à calculer, plus le temps de calcul se réduit, car il y a moins de cases disponibles.
- Le résultat de la partie entre deux IA ayant la même heuristique pour le Morpion 4x4 donne toujours une égalité (match nul).

Exemple du 1er tour: temps de calcul en secondes en fonction de l'algorithme et de la profondeur

Profondeur/IA	MinMax	MinMax AlphaBeta
3	0.028	0.002
7	118.628	1.204

Exemple du 2ème tour: temps de calcul en secondes en fonction de l'algorithme et de la profondeur

Profondeur/IA	MinMax	MinMax AlphaBeta
3	0.006	0.003
7	23.676	0.107

3. Conclusion

Les algorithmes implémentés ayant des heuristiques plus poussées que l'aléatoire sont intéressants à différents niveaux. En effet, les algorithmes étant assez généraux, ils peuvent s'appliquer à de nombreux jeux à 2 joueurs. Cependant, chaque jeu à une stratégie différente pour gagner donc les heuristiques peuvent changer d'une stratégie à l'autre. Nous avons vu que les heuristiques plus recherchées permettent d'augmenter le niveau de l'IA. Cela nous a permis de nous rendre compte que l'heuristique n'est pas le seul critère qui permet un jeu rapide et intelligent. En effet, le choix de l'algorithme utilisé (MinMax ou MinMax AlphaBeta) peut également permettre de modifier le temps de calcul et donc de rendre le temps de jeu plus ou moins long. Avec l'IA MinMax AlphaBeta, le temps de jeu est beaucoup plus court, rendant l'expérience utilisateur plus agréable.

Annexes :

Lien vers le code source sur le dépôt GitHub:
https://github.com/PaulineTeoulle/IA-Project-Morpion-4x4.git

Heuristique pour évaluer la valeur des cases :

```
public int evaluateTerminalTest(int joueur, int profondeur, int symbol, Grid grid) {
    int vainqueur = checkWinner(grid);

    if ((vainqueur == symbol)) {
        return 10000 - profondeur;
    } else if ((vainqueur == 0)) {
        int adversaire;
        if (joueur == 2) adversaire = 1;
        else adversaire = 2;
        return giveValue(joueur, adversaire, grid);
    } else {
        return -10000 + profondeur;
    }
}
```

```
private int giveValue(int joueur, int adversaire, Grid grid) {
    return evaluate(joueur, grid) - evaluate(adversaire, grid);
}
```

```
public int evaluate(int joueur, Grid grid) {
    int poids = 0;
    for (int i = 0; i < grid.getColumnCount(); i++) {
        if ((grid.grid[i][0] == 0 || grid.grid[i][0] == joueur)
                && (grid.grid[i][1] == 0 || grid.grid[i][1] == joueur)
                && (grid.grid[i][2] == 0 || grid.grid[i][2] == joueur)
                && (grid.grid[i][3] == 0 || grid.grid[i][3] == joueur)) {

            poids += (grid.grid[i][0] + grid.grid[i][1] + grid.grid[i][2] + grid.grid[i][3]) / joueur;
        }

        if ((grid.grid[0][i] == 0 || grid.grid[0][i] == joueur)
                && (grid.grid[1][i] == 0 || grid.grid[1][i] == joueur)
                && (grid.grid[2][i] == 0 || grid.grid[2][i] == joueur)
                && (grid.grid[3][i] == 0 || grid.grid[3][i] == joueur)) {

            poids += (grid.grid[0][i] + grid.grid[1][i] + grid.grid[2][i] + grid.grid[3][i]) / joueur;
        }

    }
    if ((grid.grid[0][0] == joueur || (grid.grid[0][0] == 0)
            && (grid.grid[1][1] == joueur || grid.grid[1][1] == 0)
            && (grid.grid[2][2] == joueur || grid.grid[2][2] == 0)
            && (grid.grid[3][3] == joueur || grid.grid[3][3] == 0))) {
        poids += (grid.grid[0][0] + grid.grid[1][1] + grid.grid[2][2] + grid.grid[3][3]) / joueur;
    }
    if ((grid.grid[3][0] == joueur || (grid.grid[3][0] == 0)
            && (grid.grid[2][1] == joueur || grid.grid[2][1] == 0)
            && (grid.grid[1][2] == joueur || grid.grid[1][2] == 0)
            && (grid.grid[0][3] == joueur || grid.grid[0][3] == 0))) {
        poids += (grid.grid[0][3] + grid.grid[1][2] + grid.grid[2][1] + grid.grid[3][0]) / joueur;
    }
    return poids;
}
```