Groupe :

Eden

Chérifa

Thibault

Rapport SAE Dev 2

IA Morpion

Notre projet Morpion est un jeu à deux joueurs où l'un utilise des croix (X) et l'autre des ronds (O) pour aligner 5 pions de son propre symbole sur une grille de 15x15 cases. L'objectif était de créer deux intelligences artificielles pour les croix et les ronds, le but des ronds est d'aligner 5 pions pour gagner, les croix doivent empêcher les ronds de le faire, s'ils y arrivent, ils gagnent.

Après avoir discuté de nos idées sur la manière de crée et d'implémenter les algorithmes, nous avons réalisé que la proposition de Thibault était la plus fonctionnelle. Nous avons donc travaillé ensemble pendant les heures de TP pour élaborer les IA.

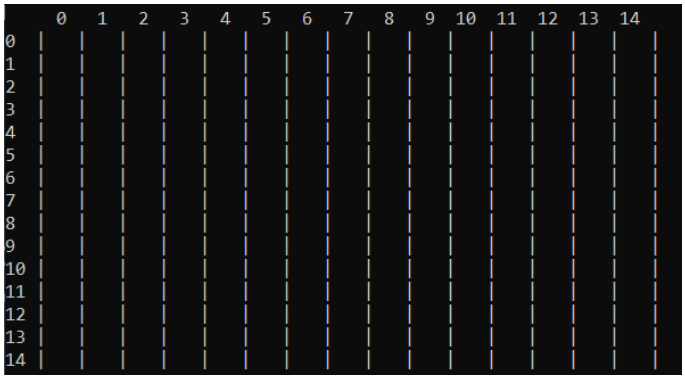
Explication des Algorithmes :

1. Affichage de la Grille (morpion.cpp):

La fonction ***affichage_morpion*** est responsable de l'affichage de la grille de jeu dans la console. Il parcourt chaque case du tableau et les imprime, ainsi que les indices des lignes et colonnes. Il utilise une boucle double pour parcourir le tableau bidimensionnel.

2. Initialisation du Morpion (morpion.cpp):

La fonction ***init_morpion*** alloue dynamiquement un tableau 2D de caractères pour représenter la grille de jeu, initialisant chaque case à un espace (' '). Cela crée la base pour le jeu.



3. Suppression du Morpion (morpion.cpp):

La fonction ***delete_morpion*** libère la mémoire allouée dynamiquement pour le tableau 2D, évitant les fuites de mémoire.

4. Placement d'un Pion (morpion.cpp):

La fonction ***placer_morpion*** vérifie d'abord si les coordonnées fournies sont valides et si la case est libre, puis place le pion approprié.

5. Vérification de la Case Libre (morpion.cpp):

La fonction ***estLibre_morpion*** vérifie si une case est libre et se trouve à l'intérieur des limites de la grille.

6. Vérification de la Victoire (morpion.cpp):

La fonction ***victoire_morpion*** parcourt la grille à la recherche d'alignements de pions du même symbole (horizontalement, verticalement et en diagonale) et détermine s'il y a une victoire en fonction du nombre de pions requis.

7. Copie de la Grille (morpion.cpp):

La fonction ***copie_morpion*** crée une copie du tableau 2D, utile pour la simulation des coups lors de l'implémentation des IA.

8. Algorithmes pour les IA (ia.h):

→Les fonctions dans le fichier "ia.h" définissent les algorithmes pour les ia des croix (***jouerCroix***) et des ronds (***jouerRond***).

Struct coord :

Cette structure coord est utilisée pour représenter les coordonnées (x, y) d'une case dans la grille du Morpion. Elle facilite la manipulation et le stockage des positions dans le code

initTabOuPlacer:

Cette fonction initialise un tableau 2D de taille 15x15. Elle remplit le tableau avec des zéros, ce sera l'espace qui permettra d'évaluer les opportunités de placement des pions

afficherTabOuPlacer:

Cette fonction affiche le tableau 2D ***tab*** dans la console, en tenant compte des valeurs négatives pour représenter les cases déjà occupées par des pions. Elle est utilisée pour comprendre les déplacements des pions.

RemplirTabOuPlacer :

Cette fonction remplit le tableau ***TabOuPlacer*** avec des valeurs représentant les opportunités de placement des pions en fonction des alignements possibles.

valMaxDuTab :

Cette fonction trouve la valeur maximale dans le tableau ***tab***, utilisée pour déterminer l'opportunité la plus avantageuse pour placer un pion.

RemplirTabCoord :

Cette fonction remplit le tableau de structures ***tabCoord*** avec les coordonnées des cases ayant la valeur maximale Max dans le tableau ***TabOuPlacer***.

➔ Pour faire simple nous nous sommes inspirées de la sous fonction ***victoire_morpion*** afin que tous les endroits où il y a 4 pions alignés, 3 pions alignés, 2 pions alignés ou 1 pion alignée, soit affiché par un 1 (2 si la case est détectée 2 fois …). Ce tableau qui s'appelle ***TabOuPlacer*** représente donc tous les endroits où « O » aurais le plus de chance de gagner pour ensuite la bloquer en amont.

A ce moment-là nous nous sommes confrontés à divers ***problèmes*** :

- « X » pouvais se placer dans une case où il y avait déjà un pion ce qui faisait boucler le programme à l'infinie.
- La case où « X » était placer était toujours la dernière dans le tableau ***TabOuPlacer***.

-> Afin de régler le premier problème, nous avons simplement ajouter une sous fonction ***CasePrise*** qui rajoute -8 dans ***TabOuPlacer*** quand il y a un « O » ou un « X » dans le morpion afin que même si la case et détecter il n'y aura jamais de pion car la case sera à -7 et donc ce ne seras jamais la valeur max.

-> Enfin, pour régler le second problème, nous avons beaucoup de sous fonctions :

- On commence par connaitre la valeur maximale dans le ***TabOuPlacer*** avec une sous-fonction ***valMaxTab*** qui renvoie cette valeur.
- Ensuite, nous prenons cette valeur et l'on met toutes ses coordonnées dans ***tabCoord*** avec la sous fonction ***RemplirTabCoord***.
- Enfin il ne reste plus qu'à choisir une adresse aléatoire dans ***tabCoord*** afin d'avoir une pose de « X » aléatoire quand il y a plusieurs endroits où il est logique de placer.

Explication de l'algorithme jouerCroix:

- La fonction identifie les meilleures positions pour les croix en analysant les possibilités d'alignement des ronds.
- Elle remplit un tableau (***TabOuPlacer***) avec des valeurs représentant les opportunités de placement des ronds.
- La fonction prend en compte les cases déjà occupées pour se placer (***CasePrise***).
- Enfin, elle choisit une position en fonction des opportunités de jeu définies par ***valMaxDuTab***

```
void jouerCroix(char ** tab, int & x, int & y){
    int TabOuPlacer[15][15];

    int size = 15;
    char pion='O';

    int max = 0;
    coord tabCoord[20];
    int TailleTabCoord = 0;
    int poseRand = 0;

    for(int nbPionAlign = 1; nbPionAlign < 5; nbPionAlign++){
        initTabOuPlacer(TabOuPlacer);
        RemplirTabOuPlacer(tab, size, nbPionAlign,pion, TabOuPlacer);
        CasePrise(tab, size, TabOuPlacer);

        max = valMaxDuTab(TabOuPlacer);
        if(max != 0){
            RemplirTabCoord(tabCoord,TailleTabCoord,max,TabOuPlacer);
            poseRand = rand()% TailleTabCoord;

            x = tabCoord[poseRand].x;
            y = tabCoord[poseRand].y;
        }
    }
}
```

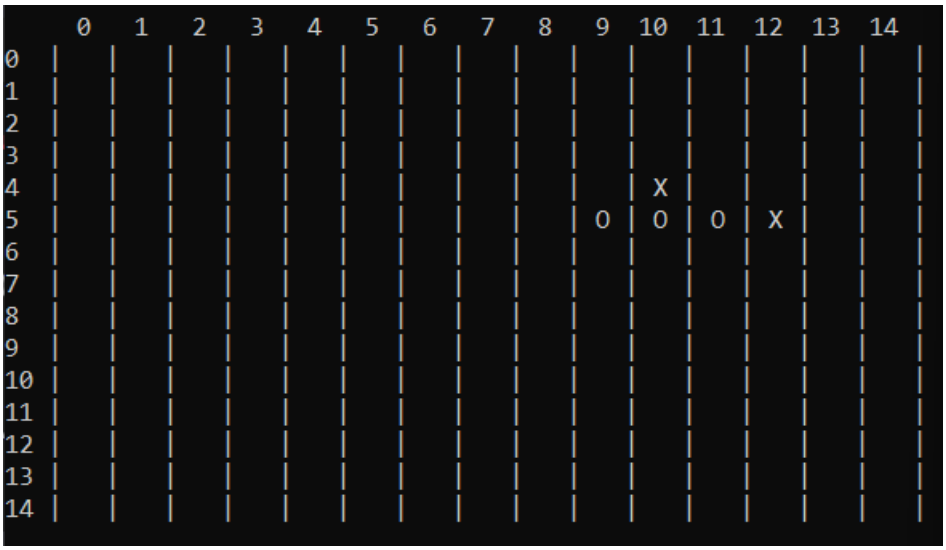
Explication de l'algorithme jouerRond :

- La fonction ***jouerRond*** se place de manière aléatoire pour le premier tour.
- Ensuite, elle applique les mêmes étapes que ***jouerCroix*** en évaluant les opportunités d'alignement pour les ronds.
- Elle choisit également une position définie par ***valMaxDuTab***

```
void jouerRond(char ** tab, int & x, int & y){
    x=rand()%10+2;
    y=rand()%10+2;

    int TabOuPlacer[15][15];

    int size = 15;
    char pion='O';

    int max = 0;
    coord tabCoord[20];
    int TailleTabCoord = 0;
    int poseRand = 0;

    for(int nbPionAlign = 1; nbPionAlign < 5; nbPionAlign++){
        initTabOuPlacer(TabOuPlacer);
        RemplirTabOuPlacer(tab, size, nbPionAlign,pion, TabOuPlacer);
        CasePrise(tab, size, TabOuPlacer);

        max = valMaxDuTab(TabOuPlacer);
        if(max != 0){
            RemplirTabCoord(tabCoord,TailleTabCoord,max,TabOuPlacer);
            poseRand = rand()% TailleTabCoord;

            x = tabCoord[poseRand].x;
            y = tabCoord[poseRand].y;
        }
    }
}
```

Démonstration :

Ici nous avons ses pions placés :



TabOuPlacer des Ronds

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  1  2 -8  2  1  0  0
0  0  0  0  0  0  0  0  1 -8 -8 -8 -8  0  0
0  0  0  0  0  0  0  0  1  2  3  2  1  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

TabOuPlacer des Croix

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0 -8  0  0  0  0
0  0  0  0  0  0  0  0  0  1 -8 -8 -8 -8  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Les -8 correspondent aux cases où sont placés les pions, le rond va se placer sur la plus grande valeur qui est 3 et la croix va donc se placer sur le 1.

```
    0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
0   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
1   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
2   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
3   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
4   |   |   |   |   |   |   |   |   |   |   X   |   |   |   |
5   |   |   |   |   |   |   |   |   X   O   O   O   X   |   |
6   |   |   |   |   |   |   |   |   |   |   O   |   |   |   |
7   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
8   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
9   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
10  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
11  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
12  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
13  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
14  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
```

Conclusion :

Ce projet a été une bonne expérience où nous avons réussi à combiner nos idées pour développer des algorithmes fonctionnels et à trouver des solutions efficaces pour les problèmes posés.