Programmation d'un Morpion

Le Morpion est un jeu simple dans lequel les joueurs jouent chacun leur tour. Deux symboles différents symbolisent chacun des deux joueurs. Un joueur est représenté par la croix (X), l'autre par le cercle (O). Le but pour les deux joueurs est d'aligner trois symboles identiques en ligne en colonne ou en diagonale en premier.

Vous réaliserez en C le programme du jeu de Morpion en suivant les indications suivantes.

En C on peut représenter la grille du Morpion par un tableau d'entiers de 9 cases (par exemple : int gameState[9];). Chaque case correspond à une case de la grille.

Mappage des index du tableau avec la grille de jeu :

0	1	2
3	4	5
6	7	8

Une case non jouée contiendra la valeur 0.

Une case jouée par le joueur 1 contiendra la valeur 1.

Une case jouée par le joueur 2 contiendra la valeur -1.

À chaque fin de tour d'un joueur vous devrez vérifier si la partie est terminée. Les conditions mettant fin à la partie sont les suivantes :

- Un joueur a aligné 3 symboles identiques en ligne, colonne ou diagonale :
    - Cela se traduit par la même valeur présente dans les cases :
        - En ligne :
            - 0, 1, 2
            - 3, 4, 5
            - 6, 7, 8
        - En colonne :
            - 0, 3, 6
            - 1, 4, 7
            - 2, 5, 8
        - En diagonale :
            - 0, 4, 8
            - 2, 4, 6
- Toutes les cases ont été jouées et aucun des deux joueurs n'a gagné (match nul)

Lorsqu'un joueur joue, il ne peut pas jouer sur une case déjà jouée par l'adversaire, si celui-ci choisi une case non vide, son tour se poursuit jusqu'à ce qu'il choisisse une case vide.

Dans un premier temps vous réaliserez ce programme avec une interface type console.

Vous devrez déclarer les fonctions suivantes dans un fichier tictactoehelper.h dont voici les prototypes :

- void initializeGameState(int * gameState);
- int testGameEnd(int * gameState);
- int * copyGameState(int * gameState);
- void destroyGameState(int * gameState);

Vous définirez le comportement de ces fonctions dans un fichier tictactoehelper.c dont voici le comportement attendu :

- initializeGameState : réinitialise le tableau de 9 cases passé en paramètre avec les valeurs 0 dans chaque case.
- testGameEnd : Retourne :
    - 0 si le jeu n'est pas terminé
    - 1 si le joueur 1 a gagné
    - -1 si le joueur 2 a gagné
    - -2 si la partie est nulle (il n'y a plus de case à jouer mais ni le joueur 1, ni le joueur 2 n'a gagné).
- copyGameState : Alloue et retourne un tableau de 9 cases dont le contenu est identique à celui passé en paramètre.
- destroyGameState : Libère la mémoire du tableau passé en paramètre.

Ces fonctions sont nécessaires pour la 3ème étape de développement (ajout d'une IA car l'algorithme min/max utilisera testGameEnd, copyGameState et destroyGameState) et ces contraintes vous aideront à diviser votre code efficacement.

Dans un second temps, vous ajouterez une IHM fenêtré (avec le code complémentaire fournit avec ce sujet).

Le code suivant permet d'afficher la fenêtre IHM en incluant le fichier tictactoewindow.h :

TicTacToeWindow * pWindow = initializeWindow();

initializeWindow() gère l'allocation de la fenêtre ainsi que l'initialisation complète de la structure qui y est associée.

La ligne suivante permet de mettre à jour l'affichage dans la fenêtre avec l'état du jeu passé en paramètre :

drawGameState(pWindow, gameState);

Les paramètres attendus sont :

- Un pointeur sur TicTacToeWindow (retourné par initializeWindow()).
- Un tableau d'entiers de 9 cases avec le mappage des index correspondant à celui décrit précédemment dans le sujet.

La ligne suivante permet de récupérer les coordonnées de la case sur laquelle clique le joueur courant :

quit = getPlayedCell(pWindow, &x, &y);

getPlayedCell retourne un int qui vaut :

- 0 si le joueur a cliqué sur une cellule
- 1 si le joueur a demandé de fermer la fenêtre

Les paramètres attendus sont :

- Un pointeur sur TicTacToeWindow (retourné par initializeWindow()).
- Un pointeur sur un entier (int *) pour récupérer l'abscisse de la case cliquée (la valeur est mise à jour par getPlayedCell).
- Un pointeur sur un entier (int *) pour récupérer l'ordonnée de la case cliquée (la valeur est mise à jour par getPlayedCell).

La ligne suivante permet de libérer la mémoire utilisée par la fenêtre une fois que vous n'en avez plus besoin (avant de quitter le programme par exemple) :

destroyWindow(pWindow);

Le paramètre attendu est un pointeur sur TicTacToeWindow (retourné par initializeWindow()).

Dans un troisième temps, vous ajouterez un mode de jeu contre une intelligence artificielle exploitant l'algorithme dit « min/max » (avec le code complémentaire fournit avec ce sujet).

La ligne suivante permet de récupérer la case à jouer considérée comme le meilleur coup à jouer selon l'algorithme min/max :

int cellToPlay = getBestAction(gameState, playerTurn);

La valeur retournée est un int correspondant à l'index du tableau considéré comme le meilleur coup à jouer selon l'algorithme min/max.

Les paramètres attendus sont :

- Un tableau d'entiers de 9 cases avec le mappage des index correspondant à celui décrit précédemment dans le sujet.
- Un entier correspondant au tour du joueur dont c'est le tour :
    - 1 pour le joueur 1
    - -1 pour le joueur 2