esprit Se former autrement	**Atelier Jeu Tic Tac Toe**	
	Matière : Projet 1A Classe(s) : 1A Equipe : Projet 1A	Unité pédagogique : Algorithmique & Programmation Année universitaire : 2021-2022

Introduction

Nous souhaitons développer un jeu de tic-tac-toe à deux adversaires : le joueur humain et le joueur machine. L'objectif est d'appliquer un algorithme d'intelligence artificielle, le **MinMax**, pour aider la machine à trouver le **prochain coup optimal**. Un coup optimal est celui qui maximise les chances de la machine de gagner ou de minimiser les chances de la victoire du joueur humain, en fonction de la configuration actuelle du plateau de jeu. Pour ce faire, nous vous proposons de suivre la démarche suivante, commençant par la définition de la structure du jeu ainsi que les en-têtes des fonctions que vous devrez développer pour le mettre en œuvre.

1. La structure Tic

Nous allons utiliser un enregistrement pour rassembler tous les éléments graphiques et leurs caractéristiques.

Nous avons besoin de :

- Une image de background contenant une grille de 9 cases (vous avez le choix de concevoir cet élément graphique)

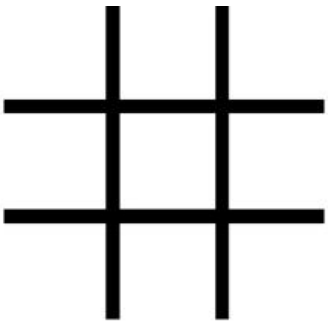


Figure1 : Exemple de background du jeu Tic Tac toe

- Une image qui s'affiche quand c'est le tour du joueur(exemple: **X**)

- Une image qui s'affiche quand c'est le tour de l'ordinateur (exemple:**O**)

- Un tableau (tabsuivi) de 9 entiers initialisés à des zéros. Ce tableau sera rempli au fur et à mesure de l'avancement de la partie : à chaque cellule on affecte :

    - 0 si la case n'est pas encore jouée
    - 1 si l'ordinateur a joué (**O**)
    - -1 si le joueur humain a joué (**X**)

- La position de la première case dans l'image (les autres peuvent être calculés automatiquement si on connait les dimensions d'une case)

- Le numéro du tour initialisé à 0 et qui ne peut pas dépasser 8. (9 tours au maximum)

- La police du texte à afficher à la fin de la partie : TTF_Font

- La couleur du texte a afficher à la fin de la partie:SDL_Color

- Le texte à afficher à la fin de la partie : char []

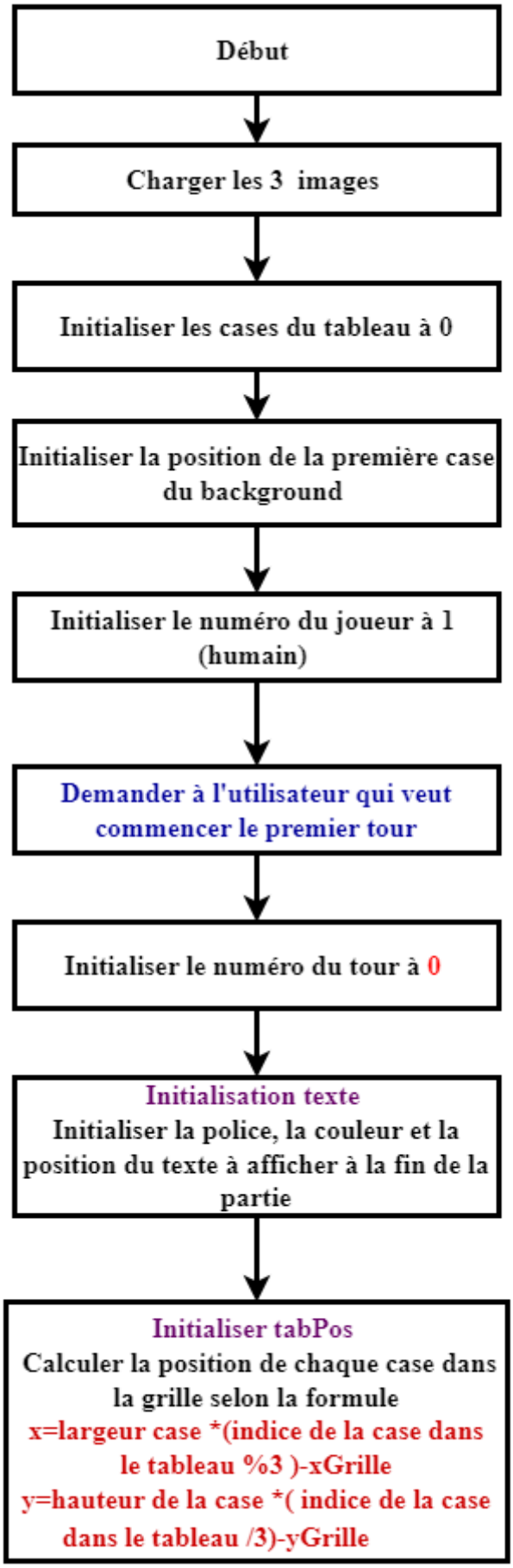
- La position du texte à afficher à la fin de la partie : SDL_Rect

- Un tableau de 9 positions (SDL_Rect) : tabPos

2 . La fonction InitialiserTic

Dans cette fonction, nous allons initialiser tous les champs de la structure précédente selon le prototype suivant :

Void initialiserTic (tic * t) ;


Début
Charger les 3 images
Initialiser les cases du tableau à 0
Initialiser la position de la première case du background
Initialiser le numéro du joueur à 1 (humain)
Demander à l'utilisateur qui veut commencer le premier tour
Initialiser le numéro du tour à 0
Initialisation texte
Initialiser la police, la couleur et la position du texte à afficher à la fin de la partie
Initialiser tabPos
Calculer la position de chaque case dans la grille selon la formule
x=largeur case *(indice de la case dans le tableau %3)-xGrille
y=hauteur de la case *(indice de la case dans le tableau /3)-yGrille

Dans la figure 2, vous avez 2 exemples de background. Dans le premier, la position de la première case est (x,y) et dans la deuxième la position est (0,0).

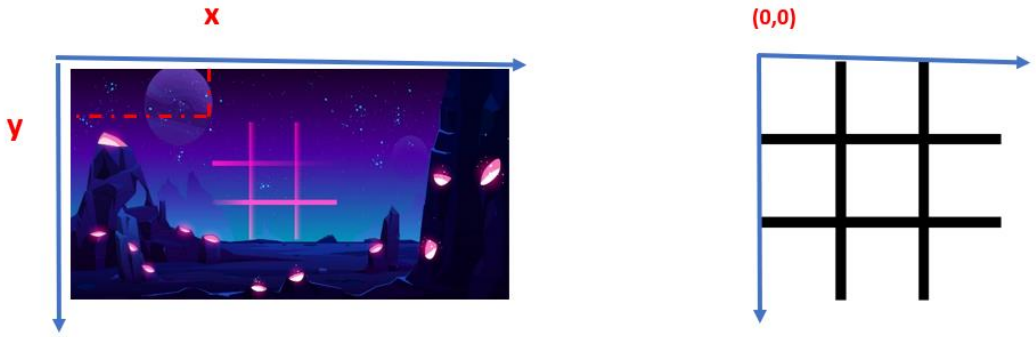


Figure 2 : Exemple d'initialisation de position selon le background

3. La fonction afficherTic

Cette fonction affichera le background et les éléments (**O**/ **X**) selon les valeurs du tableau tabsuivi. Elle doit respecter le prototype suivant :

void afficherTic(tic t,SDL_Surface* ecran);

Exemple : Si le tableau contient les valeurs suivantes :

-1	0	0	1	1	0	-1	0	0

On aura l'affichage suivant :

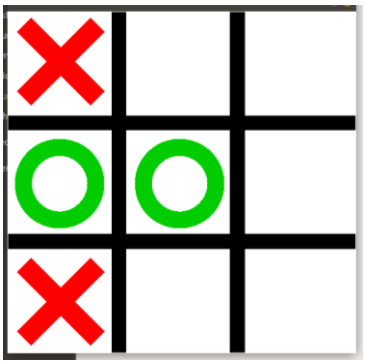


Figure 3 : Exemple d'affichage

3.4 La fonction atilgagne

C’est la fonction qui va vérifier à chaque fois s'il y a un gagnant, elle retourne :

- 1 si c'est la machine qui a gagné
- -1 si c'est le joueur humain qui a gagné
- 0 sinon

Elle doit respecter le prototype suivant :

int atilganer(int tabsuivi[]);

Si la grille est numérotée de 0 à 8 :

0	**1**	**2**
3	**4**	**5**
6	**7**	**8**

Les lignes gagnantes donc seront :

```
int lignes_gagnantes[8][3] = {{0,1,2},{3,4,5},{6,7,8},{0,3,6},{1,4,7},{2,5,8},{0,4,8},{2,4,6}};
```

et pour qu'un joueur gagne, il suffit que

```
tabsuivi[lignes_gagnantes[i][0]] = tabsuivi[lignes_gagnantes[i][1]]
                                 = tabsuivi[lignes_gagnantes[i][2]]
```

4. La fonction Resultat

Une fois la partie terminée, Cette fonction permet d’afficher un message (Nul, Défaite, victoire), selon le retour de la fonction précédente.

Cette fonction doit respecter le prototype suivant:

void Resultat (int tabsuivi[],SDL_Surface* ecran);

5. La fonction libererMemoire

Cette fonction permet de libérer les ressources liées aux images et au texte. Elle doit respecter le prototype suivant : **void liberer(tic t);**

6. La fonction minimax

La fonction récursive **minimax** calcule le score en parcourant l'arbre des possibilités. Elle doit

respecter le prototype suivant : **int minimax (int tabsuivi[9], int joueur) ;**

L'algorithme de cette fonction est le suivant :

```
Fonction minimax(tabsuivi[9], joueur)
Var
gagnant, meilleure_valeur, valeur_coup : entier
Début
   gagnant← atilganer(tabsuivi) // Vérifier s'il y a un gagnant ou si la partie est terminée

   Si (gagnant ! = 0) alors
      Si ((joueur = 1) et (gagnant = 1)) alors
      // Si la machine a gagné
         Retourner 1
      Sinon si ((joueur == -1) et (gagnant == -1)) alors
         // Si le joueur humain a gagné
         Retourner -1
      Sinon
      // c'est une partie nulle
         Retourner 0
   Fin Si                                          // Si la partie est terminée

   Si (joueur = 1) alors    // Initialiser la meilleure valeur en fonction du joueur actuel

      meilleure_valeur ← INT_MIN (-INFINI) // Initialiser à une valeur très petite pour la maximisation
   Sinon
      meilleure_valeur ← INT_MAX (+INFINI)  // Initialiser à une valeur très grande pour la minimisation
   Fin Si
   Pour i de 1 à 9 faire // Pour chaque case vide de tabsuivi, simuler le coup et calculer sa valeur
      Si (tabsuivi[i] = 0) alors

         tabsuivi[i] ←joueur // Simuler le coup pour le joueur actuel

         // Calculer la valeur du coup en appelant récursivement minimax pour l'autre joueur
         valeur_coup ← minimax(tabsuivi, -joueur)

        tabsuivi[i] ←0    // Annuler le coup pour revenir à l'état précédent

         Si (joueur = 1) alors  // Mettre à jour la meilleure valeur en fonction du joueur actuel

               // Maximisation pour la machine
           meilleure_valeur <- max(meilleure_valeur, valeur_coup)
         Sinon
              // Minimisation pour le joueur humain
            meilleure_valeur <- min(meilleure_valeur, valeur_coup)
         Fin Si
      Fin Si
   Fin Pour
```

```
    // Retourner la meilleure valeur trouvée après avoir exploré tous les coups possibles
    Retourner meilleure_valeur
Fin
```

7. La fonction calcul_coup

Elle doit respecter le prototype suivant : **void calcul_coup (int tabsuivi[9]) ;**

Cette fonction va mettre à jour le **tabsuivi[i]** à 1 si le score rendu par la fonction **minimax(tabsuivi,-1)** est maximal.

```
Fonction calcul_coup(tabsuivi[9])
    meilleure_valeur ← INT_MIN (INFINI) // Initialiser à une valeur très petite pour la maximisation
    meilleur_coup ← -1 // Initialiser le meilleur coup à jouer
    pour i de 1 à 9 faire
        Si (tabsuivi[i] = 0) alors
            // Simuler le coup pour la machine (représentée par 1)
            tabsuivi[i] ← 1
            // Calculer la valeur du coup en utilisant l'algorithme minimax
            valeur_coup ← minimax(tabsuivi, -1) // Recherche de la valeur minimale pour le joueur humain
            // Annuler le coup pour revenir à l'état précédent
            tabsuivi[i] ← 0
            // Mettre à jour le meilleur coup si la valeur du coup est meilleure
            si (valeur_coup > meilleure_valeur) alors
                meilleure_valeur ← valeur_coup
                meilleur_coup ←i
            fin si
        fin si
    fin pour

    // Jouer le meilleur coup trouvé
    si (meilleur_coup != -1) alors
        // Afficher le coup à jouer
        afficher("Le prochain coup à jouer est à la case ", meilleur_coup)
```

// Mettre à jour le tableau avec le meilleur coup

tabsuivi[meilleur_coup] ← 1 // Marquer la case comme jouée par la machine

Sinon

// Afficher un message d'erreur si aucun coup valide n'a été trouvé

afficher ("Aucun coup valide trouvé.")

Fin si

Fin fonction

8 La fonction principale main

En respectant la conception de la boucle du jeu, l'appel des fonctions doit se faire selon le pseudo code suivant:

Début main

t:tic

ecran: SDL_Surface

Continuer,x,y, tour: entier

initialiserTic(&t)

tour=1(le joueur commence), continuer=1

Tant que continuer ==1

afficherTic(t.tabsuivi, ecran)

SI (t.tour<9 && **atilgagne**(t.tabsuivi)==0) **alors**// Tant qu'on n'a pas atteint le dernier tour et qu'aucun des 2 joueurs n'a encore gagné

Si ((t.tour)%2=1) //tour du PC

alors **calcul_coup**(t.tabsuivi); t.tour++

sinon

SDL_WaitEvent(&event);

Si(event.type= SDL_MOUSEBUTTONUP) **alors**

x=event.button.x/largeur d'une case;

y=event.button.y/hauteur d'une case;

coup=3*y+x;

t.tabsuivi[coup]=-1;

t.tour++

sinon si(event.type= SDL_QUIT) alors continuer=0

Fin Si

Fin Si

sinon

Resultat(t.tabsuivi, ecran)

Fin SI

Fin Tant que

liberermemoire(t);

Fin main