académie Créteil

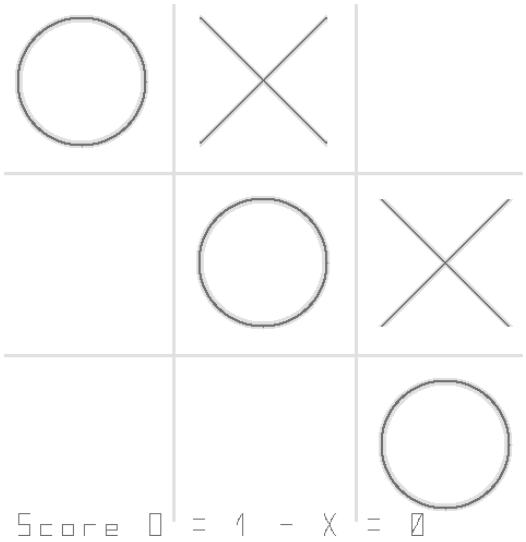
# BTS SNIR

Lycée Louis Armand
94 - Nogent sur Marne

# Module C/C++

# Jeu de morpion :

# génération de la grille en bmp




## Table des matières

# Objectifs et ressources

## Objectifs

Programmer un jeu de morpion en mode console. Le programme vérifiera la grille et affichera les scores des deux joueurs. Une image bitmap de la grille sera générée. Codé dans un premier temps en langage C (création de fonctions), le programme sera réagencé pour créer une classe en C++. La création de l'image se fera grâce à l'utilisation d'une classe donnée (SNImage) permettant de dessiner dans un bitmap.

- Tableau à 2 dimensions
- Algorithmie
- Langage C
- Classe en C++
- Utilisation d'une classe de traitement d'image

## Logiciels

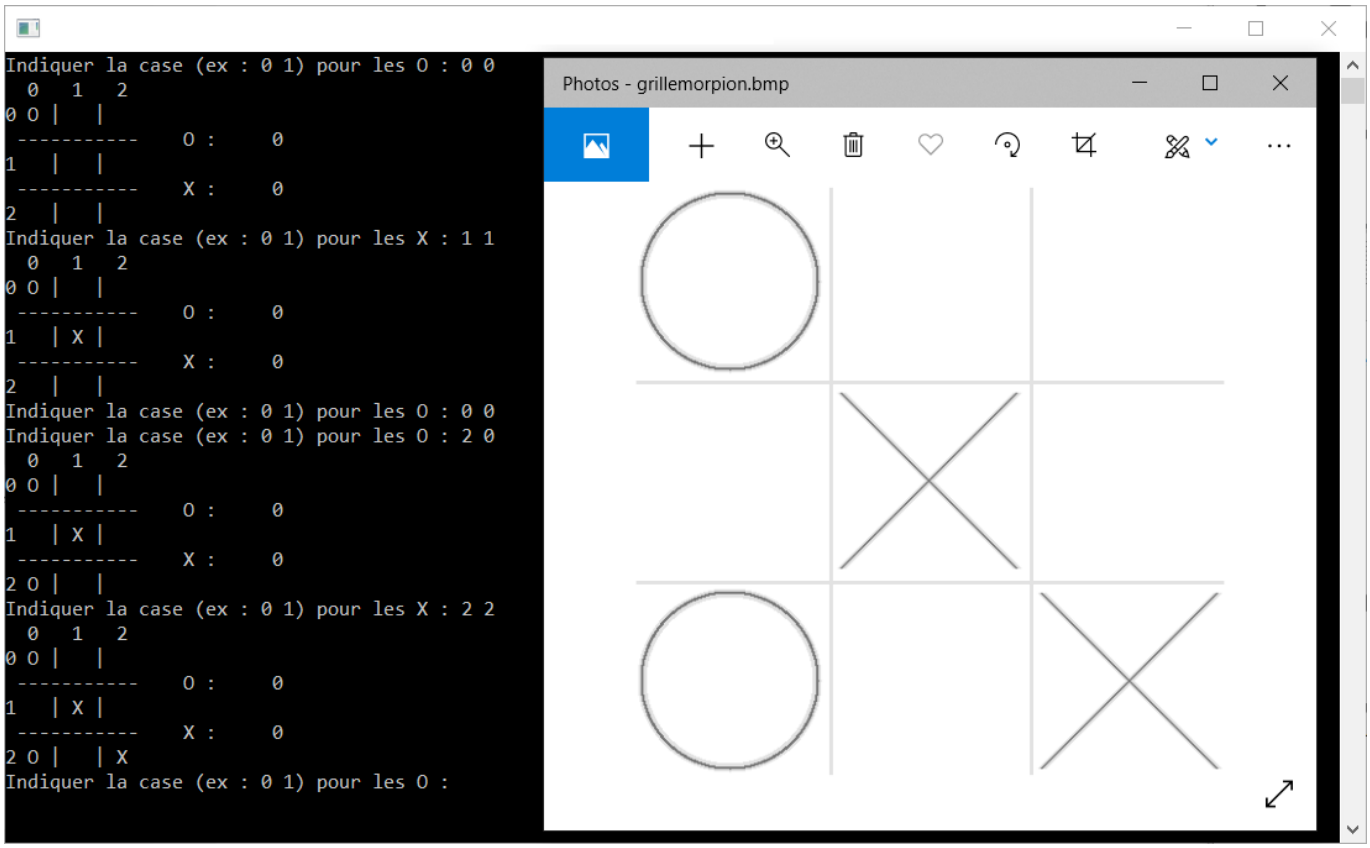
- C++ Builder
- CodeBlocks

## Ressources

- SNImage.h
- SNImage.o (pour CodeBlocks)
- SNImage.obj (pour C++ Builder)

## Annexe

- Diagramme de classes



# – Défi 1 – Codage du jeu en langage C

Nous allons tout d'abord coder le jeu en utilisant des variables globales et des fonctions. Le code sera programmé en langage C, mis à part l'utilisation de **cout** de la classe C++ **iostream** pour les affichages. Pour un code 100 % langage C, utiliser **printf**() – https://fr.wikipedia.org/wiki/Printf – à la place de **cout**, et **scanf()** – https://fr.wikipedia.org/wiki/Scanf – à la place de **cin**. Dès que le jeu sera fonctionnel, les variables globales seront regroupées dans une classe pour devenir des attributs, le fonctions deviendront des méthodes. La classe pourra alors être utilisée dans un autre environnement, et pourra être utilisée dans une interface graphique.

### Création des variables globales (déclarées en dehors du main) :

- Un tableau **morpion** à 2 dimensions de 3x3 caractères.
- 3 variables de type caractères : **rond**, **croix**, **vide**, initialisées respectivement à **'O'**, **'X'**, **' '**.
- 2 variables entières pour les scores : **scoreRond** et **scoreCroix**.

Donner la déclaration des variables globales.

### Initialisation de la grille :

La fonction **InitialiserGrille()** permet de stocker **vide** dans toutes les cases du tableau **morpion** (il est nécessaire d'utiliser 2 boucles **for** imbriquées).

Donner le code de la fonction InitialiserGrille.

### Affichage de la grille :

La fonction **AfficherGrille()**, permet d'afficher le tableau à 2 dimensions dans un quadrillage, ainsi que les scores :

//exemple d'affichage :

```
      0   1   2
    0   |   |
     -----------     O :     0
    1   |   |
     -----------     X :     0
    2   |   |
```

### Main :

Initialiser les scores à 0, puis appeler les 2 fonctions pour obtenir le résultat ci-dessus.

Pour faire un test d'affichage, intercaler entre l'appel de l'initialisation et celui de l'affichage :

```
morpion[0][0]=croix;
morpion[1][0]=rond;
scoreRond=4;
```

Coder et tester le programme en mode console.

//exemple de sortie console :

```
   0   1   2
0 X |   |
 -----------     O :     4
1 O |   |
 -----------     X :     0
2   |   |
```

Mettre les 3 instructions de test d'affichage en commentaire.

Ajouter, au début du **main**, 2 variables entières **ligne** et **colonne**, ainsi qu'une variable **pion**, de type caractère. Initialiser **pion** à la valeur de la variable **rond**. Ajouter la demande d'un coup pour les **O** ou les **X** (valeur de la variable **pion**) permettant de saisir la ligne et la colonne du coup (utiliser **cout** et **cin**) .

Coder et tester la saisie du coup.

//exemple de sortie console :

```
   0   1   2
0   |   |
 -----------     O :     0
1   |   |
 -----------     X :     0
2   |   |
Indiquer la case (ex : 0 1) pour les O :
```

Stocker la valeur de **pion** dans **morpion[ligne][colonne]**, puis appeler à nouveau **AfficherGrille()** :

Coder et tester la prise en compte du coup.

//exemple de sortie console :

```
   0   1   2
0   |   |
 -----------     O :     0
1   |   |
 -----------     X :     0
2   |   |
Indiquer la case (ex : 0 1) pour les O : 1 2
   0   1   2
0   |   |
 -----------     O :     0
1   |   | O
 -----------     X :     0
2   |   |
```

### Enregistrement et vérification du coup :

Une nouvelle fonction **bool EnregistrerCoup(int ligne, int colonne, char pion)** doit être créée pour vérifier que la case est bien vide. Il faut déplacer dans cette fonction la ligne du programme principal stockant **pion** dans **morpion[ligne][colonne]**. Avant de stocker la valeur de **pion** dans le tableau :

- Si les valeurs de **ligne** et **colonne** sortent du tableau ou bien si la case n'est pas vide : retourner **false**.
- A la fin de la fonction retourner **true**.

Donner le code de la fonction EnregistrerCoup.

### Main :

Déclarer 2 booléens **coupValide** et **nouveau** initialisés à **true**.

Placer la saisie du coup dans un **do...while** : tant que **coupValide** est **false** le message est affiché et la saisie est effectuée à nouveau. Dans le **do...while**, appeler la fonction **EnregistrerCoup** et stocker le résultat dans **coupValide**. Placer le tout dans un autre **do...while** dont la condition est : **nouveau** a la valeur **false**.

Voici la structure complète du programme principal :

```
Déclaration et initialisation des variables.
FAIRE
     SI nouveau est vrai
          appeler InitialiserGrille
          mettre nouveau à faux
     FINSI
     appeler AfficherGrille
     FAIRE
          saisir ligne est colonne
          appeler EnregistrerCoup
          et stocker le résultat dans coupValide
     TANTQUE coupValide est faux
     SI pion est égal à rond
          stocker croix dans pion
     SINON
          stocker rond dans pion
     FINSI
TANTQUE nouveau est faux
```

Donner le code du programme principal.

Coder et tester le jeu.

//exemple de sortie console :

```
      0   1   2
    0   |   |
     -----------    O :     0
    1   |   |
     -----------    X :     0
    2   |   |
    Indiquer la case (ex : 0 1) pour les O : 1 3
    Indiquer la case (ex : 0 1) pour les O : 1 2
      0   1   2
    0   |   |
     -----------    O :     0
    1   |   | O
     -----------    X :     0
    2   |   |
    Indiquer la case (ex : 0 1) pour les X : 1 2
    Indiquer la case (ex : 0 1) pour les X : 1 1
      0   1   2
    0   |   |
     -----------    O :     0
    1   | X | O
     -----------    X :     0
    2   |   |
```

### Grille complète :

La fonction **bool GrilleComplete()** est chargée de signaler que la grille est complète.

Pour toutes les cases du tableau morpion :

- Si une case est vide retourner **false**.
- Retourner **true** à la fin de la fonction.

Donner le code de la fonction GrilleComplete.

### Main :

Déclarer un caractère **reponse**.

Après avoir enregistré le coup, appeler la fonction **GrilleComplete()**. Si cette fonction retourne **true** :

- Afficher **"Nouvelle partie O/N ? "** et stocker la réponse de l'utilisateur dans **reponse**.
- Mettre **nouveau** à **true** et afficher la grille.

Il faut modifier alors la condition du dernier **do...while** : tant que **reponse** vaut **'O'** ou **nouveau** vaut **false**.

Coder et tester le programme.

### Gain :

Il nous reste plus qu'à coder la fonction **char Gain()** qui retourne **'O'** si les ronds gagnent, **'X'** si ce sont les croix, **' '** (espace) si aucun joueur ne gagne. Il est nécessaire de tester les 8 possibilités de gain : lignes, colonnes et diagonales. Il est aussi possible, mais plus difficile, d'utiliser des boucles.

Donner le code de la fonction Gain.

### Main :

Après avoir enregistré le coup et avant d'avoir tester **GrilleComplete()**, appeler la fonction **Gain()**. Si la valeur retournée n'est pas **' '**, afficher qui a gagné puis proposer une nouvelle partie. Incrémenter alors les scores. Si personne ne gagne et que la grille est complète afficher **"Partie nulle"**.

Coder et tester le programme final du jeu de morpion en mode console.

Pour faciliter les tests, on donne la fonction **SimulerGrille()** qui permet de tester la fonction **Gain()** pour chaque position gagnante :

```
void SimulerGrille()
{    int alignement=0;
      for(char pion=rond; pion<=croix; pion+=(croix-rond))
      {     for(int alignement=0;alignement<8;alignement++)
            {     InitialiserGrille();
                  //lignes :
                  if(alignement<3)
                  {     for(int j=0;j<3;j++) morpion[alignement][j]=pion;
                  }
                  //colonnes :
                  else
                  if(alignement<6)
                  {     for(int i=0;i<3;i++) morpion[i][alignement-3]=pion;
                  }
                  //diagonales
                  else
                  if(alignement==6)
                  {     for(int i=0;i<3;i++) morpion[i][i]=pion;
                  }
                  else
                  if(alignement==7)
                  {     for(int i=0;i<3;i++) morpion[i][2-i]=pion;
                  }
                   AfficherGrille();
                   cout<<"Gain : "<<Gain()<<endl;
            }
      }
}
```

//Correction du programme principal :

```
      char reponse='N';
      bool nouveau=true;
      char pion=rond,pionVictorieux=vide;
      int ligne,colonne;
      bool coupValide;
      scoreRond=0,scoreCroix=0;
      //SimulerGrille();
      do
      {   if(nouveau)
            {     InitialiserGrille();
                  nouveau=false;
            }
            AfficherGrille();
            do
            {     cout<<"Indiquer la case (ex : 0 1) pour les "<<pion<<" : ";
                  cin>>ligne>>colonne;
                  coupValide=EnregistrerCoup(ligne,colonne,pion);
            }
            while(!coupValide);
            if(pion==rond) pion=croix; else pion=rond;
            pionVictorieux=Gain();
            if(GrilleComplete()||pionVictorieux!=vide)
            {   AfficherGrille();
                  if(pionVictorieux!=vide) cout<<"Victoire des "<<Gain()<<endl;
                  else cout<<"Partie nulle"<<endl;
                  if(pionVictorieux==croix)scoreCroix++;
                  if(pionVictorieux==rond)scoreRond++;
                  cout<<"Nouvelle partie O/N ? ";
                  cin>>reponse;
                  nouveau=true;
                  clrscr();
            }
      }while(reponse=='O' || !nouveau);
```

# – Défi 2 – La classe Morpion

Il est parfois plus simple de coder un programme directement dans le programme principal, en déclarant en global les variables importantes. Puis en regroupant en fonctions les morceaux cohérents de codes qui répondent à une fonctionnalité. En regroupant également dans des fonctions les morceaux de codes qui sont utilisés plusieurs fois. Lorsque l'ensemble du programme est fonctionnel il est nécessaire de faire une classe afin de rendre les fonctionnalités réutilisables dans un autre programme plus complexe.

Créer la classe Morpion à partir du code réalisé précédemment : les variables globales deviennent des attributs, les fonctions deviennent des méthodes. Les attributs doivent être privés et les méthodes publiques (sauf si elle ne sont pas utilisées dans le programme principal). Lorsque un attribut doit être utilisé dans le programme principal, il faut créer une méthode d'accès qui retourne la valeur de l'attribut. Si l'attribut doit être modifié par le programme principal : il faut créer une méthode prenant en argument la valeur à stocker dans l'attribut.

En partant de votre code du défi précédent, proposer un diagramme de classes. Y a-t-il besoin de créer des méthodes d'accès ? Si oui, pour quel(s) attribut(s).

La déclaration de la classe sera placée dans **morpion.h**, la définition des méthodes dans **morpion.cpp**.

**Main :**

Créer un objet de la classe Morpion et remplacer chaque appel de fonction par un appel de méthode avec l'objet créé.

Coder et tester le programme en mode console.

Une correction du diagramme de classes est donnée en annexe.

# – Défi 3 – Génération de l'image de la grille

La classe **SNImage** permet de dessiner dans une image bitmap 24 bits (**.bmp** ). Le diagramme de classe est donné en annexe. Sont donnés également : le fichier **.h** , le **.o** (ou le **.obj** ) : ce dernier est le code des méthodes de la classe déjà compilé, il dépend d'un compilateur.

Créer un nouveau programme afin de tester les fonctionnalités de SNImage. Exécuter le programme vide afin de créer le répertoire d'exécution.

Placer le fichier morpion.bmp (une image blanche de 600x600 en bitmap 24 bits créée avec Paint par exemple) dans :

- Win32\Debug pour C++ Builder,
- bin\Debug pour CodeBlocks, ainsi que dans le même répertoire que votre code source.

Placer :

- SNImage.obj dans Win32\Debug pour C++ Builder
- SNImage.o dans obj\Debug pour CodeBlocks.

Dans l'environnement de développement choisi, ajouter le fichier (o ou .obj) au projet :

- C++ Builder : clic droit sur Project1.exe dans le gestionnaire de projet > Ajouter… Win32\Debug\SNImage.obj
- CodeBlocks : Settings > Compiler and debugger… > Linker settings > Add obj\Debug\SNImage.o

### Test de la classe SNImage :

Dans le fichier du programme principal, inclure **SNImage.h**. Dans le main, créer un objet **img** de la classe **SNImage**. Appeler la méthode **Charger**, en passant **"morpion.bmp"** en argument, afficher la largeur et la hauteur de l'image en appelant les méthodes Largeur() et Hauteur(). Ne pas oublier d'ajouter **cin.get()** afin que la console ne se ferme pas.

Tester le programme et vérifier les dimensions affichées.

Ajouter la sauvegarde dans une autre image en appelant la méthode **Sauvegarde** et en passant **"morpionGrille.bmp"** en argument. Après exécution le nouveau fichier est créé dans le même répertoire que l'image d'origine. Avant la sauvegarde, appeler la méthode **Negatif()**. L'image créée après exécution devient noire.

Tester le programme et vérifier la présence de la nouvelle image.

### Dessiner la grille :

La classe **SNImage** utilise 2 structures : **Pixel** et **Coordonnee**. **Pixel** est composé des 3 composantes d'un pixel : **rouge**, **vert** et **bleu** ce sont des octets. **Coordonnee** contient 2 entiers : **ligne** et **colonne**.

En étudiant SNImage.h, donner les déclarations des structures Pixel et Coordonnee.

Commenter chaque ligne -hors commentaire- du code ci-dessous et vous référant à SNImage.h (ou bien au diagramme de classes).

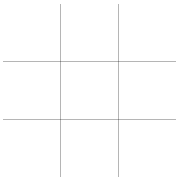
```
img.Charger("morpion.bmp");
Coordonnee debut;
Pixel couleur;
couleur.rouge=0;
couleur.bleu=0;
couleur.vert=0;
debut.ligne=200;
debut.colonne=0;
img.Colorier(debut,600,1,couleur);
//void Colorier(Coordonnee coord,int l,int h,Pixel couleur);
img.Sauvegarder("grillemorpion.bmp");
```

Tester le code ci-dessus, et vérifier qu'une ligne horizontale est tracée dans l'image.



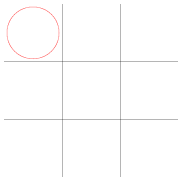
Coder et tester le programme permettant de tracer les 4 lignes de la grille.

Il est possible, mais pas si simple, de faire une boucle.



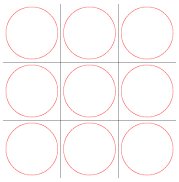
### Tracer un rond :

En reprenant le code de tracé d'une ligne, remplacer l'appel de la méthode **Colorier()** par celle de **TracerCercle()** : **void TracerCercle(Coordonnee centre, int rayon, int epaisseur, Pixel couleur)**. Les coordonnées du premier cercle sont **(100,100)**, le rayon **90** par exemple.



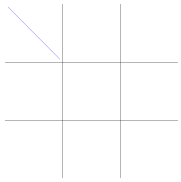
Coder et tester le programme permettant de tracer le premier cercle rouge.

Tracer maintenant les 9 ronds en utilisant une double boucle **for (i de 0 à 2 et j de 0 à 2)**. Les coordonnées des centres des cercles sont alors **100*(2*i+1)** pour la ligne, **100*(2*j+1)** pour la colonne.



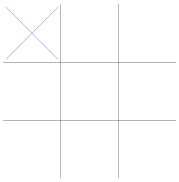
### Tracer une croix :

Utiliser cette fois la méthode **void TracerDiagonale(Coordonnee debut,Coordonnee fin, int epaisseur, Pixel couleur)**. Cette méthode à besoin des coordonnées de fin de la diagonale. Tracer la 1ère diagonale de **(10,190)** à **(10,190)**.

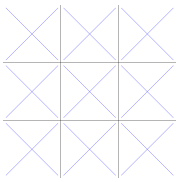


Coder et tester le programme, la première diagonale doit être tracée.

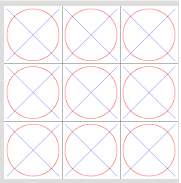
Coder et tester le programme permettant de tracer la 2ème diagonale afin d'obtenir la 1ère croix.



Coder et tester le programme permettant de tracer maintenant les 9 croix : les 9 premières diagonales vont de **(200*i+10,200*j+10)** à **(200*i+190,200*j+190)**.



//Correction du tracé complet



```
SNImage img;
img.Charger("morpion.bmp");
Coordonnee debut,fin;
Pixel couleur;
couleur.rouge=0;
couleur.bleu=0;
couleur.vert=0;
for(int i=0;i<4;i++)
{     debut.ligne=200*(i%2+1)*(i<2);
      debut.colonne=200*(i%2+1)*(i>1);
      img.Colorier(debut,599*(i<2)+1,599*(i>1)+1,couleur);
}
couleur.rouge=255;
couleur.bleu=0;
couleur.vert=0;
for(int i=0;i<3;i++)for(int j=0;j<3;j++)
{     debut.ligne=100*(2*i+1);
      debut.colonne=100*(2*j+1);
      img.TracerCercle(debut,90,1,couleur);
}
couleur.rouge=0;
couleur.bleu=255;
couleur.vert=0;
for(int i=0;i<3;i++)for(int j=0;j<3;j++)
{     debut.ligne=200*i+10;
      debut.colonne=200*j+10;
      fin.ligne=200*i+190;;
      fin.colonne=200*j+190;
      img.TracerDiagonale(debut,fin,1,couleur);
      debut.ligne=200*i+10;
      debut.colonne=200*j+190;
      fin.ligne=200*i+190;
      fin.colonne=200*j+10;
      img.TracerDiagonale(debut,fin,1,couleur);
}
img.Sauvegarder("grillemorpion.bmp");
```

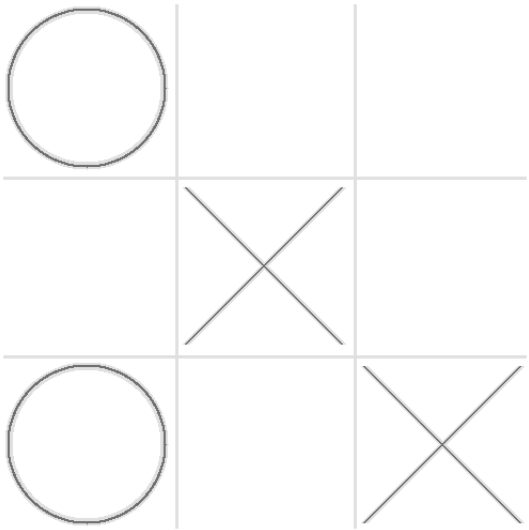
## Tracé de la position :

Le projet de test de la classe **SNImage**, la génération de la grille, des ronds et des croix est validé. Retourner dans le code initial du jeu de morpion, créer une nouvelle méthode de la classe **Morpion** : **TracerMorpionBMP()** y placer le code du tracé de la grille et des pions dans une image. Ajouter des tests afin de tracer uniquement :

- Un rond si **morpion[i][j]==rond**.
- Une croix si **morpion[i][j]==croix**.

Cette méthode doit être appelée dans **AfficherGrille()** :

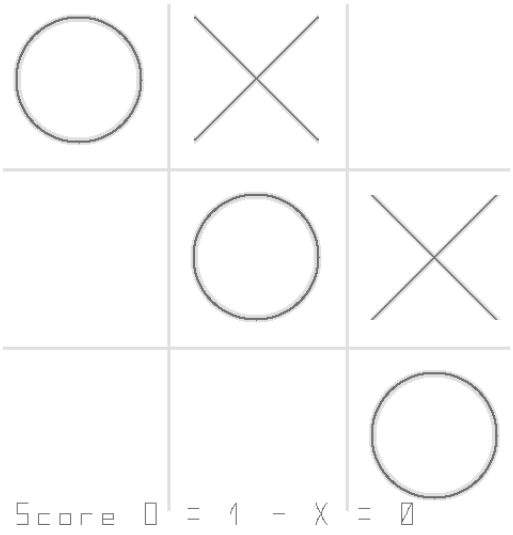
Au cours du jeu la position est mise à jour dans l'image.



//OPTION

En utilisant la méthode **Dessiner36Segments()**, ajouter le score des joueurs sur l'image (utiliser la fonction **sprintf** permettant de placer des entiers **"%d"** ou des caractères **"%c"** dans une chaîne de caractère formatée) : **sprintf(texte,"Score %c = %d - %c = %d",rond,scoreRond,croix,scoreCroix)**.

http://www.cplusplus.com/reference/cstdio/sprintf/.



# Annexe : Diagramme de classes