Programmation d'un Othello 2 joueurs

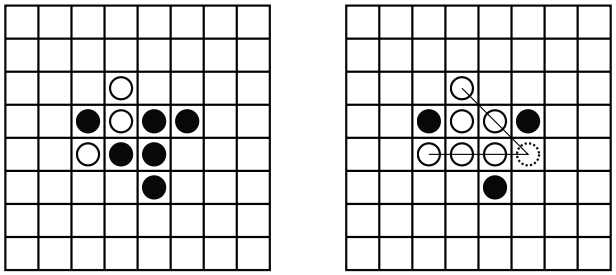
Objectif :

Utiliser les compétences acquises pour programmer un jeu simple. S'initier au "picking" (sélection à l'aide de la souris d'un objet dans le monde OpenGL).

Règles du jeu :

Deux joueurs (pions bancs et pions noirs) jouent à tour de rôle (en commençant par les pions noirs) en posant un pion de leur couleur sur le plateau.

Pour pouvoir placer un pion, un joueur doit obligatoirement capturer des pions adverses. Pour capturer un/des pions, il faut que le pion ajouté encercle, à l'aide d'un pion déjà sur le plateau, une ligne d'au moins un pion adverse (ligne verticale,horizontale ou diagonale). Tous les pions ainsi capturés changent de couleur. Sur l'exemple ci-dessous, en plaçant le pion blanc (en pointillé) le joueur capture deux pions noirs sur la ligne horizontale et un pion noir sur la diagonale.



Quand un joueur ne peut pas placer de pion, il passe son tour. La partie s'arrête quand aucun des joueurs ne peut jouer. Le gagnant est alors celui qui possède le plus de pions sur le plateau.

Cette programmation est à faire de façon incrémentale, avec une validation épate par étape.

1 Initialisation : mettre les pions de départ dans la grille
2 Picking : gestion des interactions
3 Gestion des règles du jeu

1 Travail à faire

1.1 Initialisation

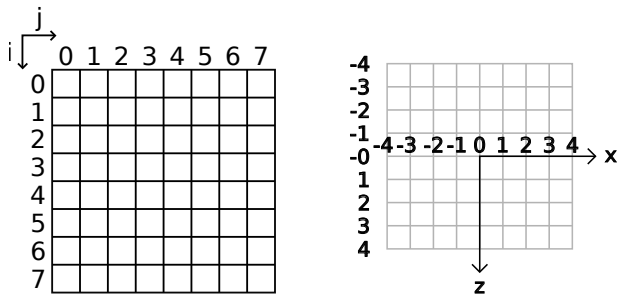
1. Afin de gérer les pions, on utilise un tableau à deux dimensions qui correspondra à l'espace de jeu :

```
1    Pions= [[None]*8 for i in xrange(8)]
```

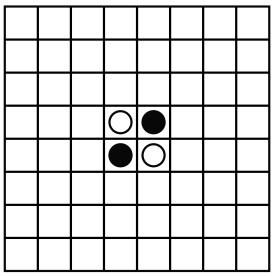
La variable **Pions[i][j]** correspond à l'emplacement (i,j) sur le plateau. Pour chaque case, il y a trois valeurs possibles :

0 La case est occupée par un pion noir
1 La case est occupée par un pion blanc

2 La case est vide (pas de pion)



Écrire la fonction **initGrid()** qui initialise la grille de la façon suivante (toutes les cellules de la grille sont initialisées à 2, sauf les quatre du centre) :



2. Écrire la fonction **drawPions()** qui permet de dessiner tous les pions présents sur le plateau.

1.2 Picking

1. À l'aide du tutoriel de picking, compléter la fonction **MouseFunc** en calculant les coordonnées (i,j) de la grille correspondant aux coordonnées (x,y) de la position de la souris sur l'écran. On utilisera les fonctions **glReadPixels()** et **gluUnProject()**. Remarque : on manipule ici, trois repères différents : le repère 2D (x,y) de la fenêtre, le repère 3D (x,y,z) du monde OpenGL et le repère 2D (i,j) de la grille.

2. Sans tenir compte, dans un premier temps, des règles du jeu, permettre au joueur de placer alternativement un pion sur le plateau.

3. À chaque changement de joueur, le plateau effectue une rotation de 180 degrés. Pour cela, écrire la fonction **animationRotation(nbCall)** qui permet de tourner le plateau avant de changer de joueur.

1.3 Mise en place des règles

1. Écrire la fonction **testerCase(i,j)** qui permet de tester l'ajout d'un nouveau pion dans une case. Cette fonction doit, pour les 8 directions possibles (haut, bas, gauche, droite et les 4 diagonales), tester si la case permet de capturer des pions adverses, et si c'est le cas elle doit

changer la couleur de ces pions. Si un ou plusieurs pions ont été capturés, la couleur du joueur en cours (variable globale **CouleurEnCours**) doit changer (pour permettre à l'autre joueur de pouvoir jouer). Sinon il ne se passe rien, le joueur peut essayer une autre case. **Attention :** tester et valider les directions une par une ! Pour vous aider, vous trouverez le début du pseudo-code de cette fonction dans la section 2.

2. Il reste alors deux chose à gérer :
   - le cas où un joueur est bloqué et ne peut plus placer de pion (il doit alors passer son tour)
   - la fin de la partie : quand les deux joueurs ont passé leur tour, il faut compter les pièces et annoncer le joueur vainqueur.

2 Pseudo code de TesterCase(i,j)

Algorithm 1 TesterCase(i,j)

```
if la cellule (i,j) est vide then
  //Il faut tester les huit directions :
  //SUD
  for k allant de 1 au nombre de lignes-1 do
    if la cellule (i,j+k) correspond à un pion du joueur courant et que k>1 then
      for kp allant de 0 à k do
        On change la couleur du pion présent dans la cellule (i,j+kp)
      end for
    else if la cellule (i,j+k) est vide or la cellule (i,j+k) correspond à un pion du joueur
    courant then
      On sort de la boucle for (break)
    end if
  end for
  //NORD
  for k allant de 1 à j+1 do
    if la cellule (i,j-k) correspond à un pion du joueur courant et que k>1 then
      for kp allant de 0 à k do
        On change la couleur du pion présent dans la cellule (i,j-kp)
      end for
    else if la cellule (i,j-k) est vide or la cellule (i,j-k) correspond à un pion du joueur courant
    then
      On sort de la boucle for (break)
    end if
  end for
  //EST
  ...
  //OUEST
  ...
  //NORD-EST
  ...
  //SUD-EST
  ...
  //NORD-OUEST
  ...
  //SUD-OUEST
  ...
  if le joueur courant a récupéré des pions adverses then
    On change de joueur
  end if
end if
```