Projet : construction d'un jeu *Tétris* en C++

1. Introduction

Ce projet en C++ a pour objectif de réaliser le jeu *Tétris* en 2D. Il va vous permettre de mettre en application les concepts vu en cours et TP. Pour réaliser ce jeu, vous disposerez d'un code de base vous permettant d'afficher dans une fenêtre graphique des primitives 2d ainsi que du texte.

Principe : Le but est de placer des formes (7 formes cubiques différentes) qui descendent les unes après les autres, du haut vers la bas d'un tableau pour en former des lignes horizontales pleines. Dès qu'elle est complète, la ligne est détruite et tous les cubes au-dessus de la ligne descendent d'une rangée et permet de remporter des points.

2. Présentation du code de base

2.1. Les bases d'une application graphique interactive

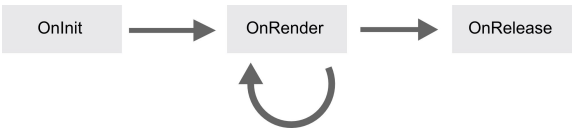

Figure 1: *Processus d'une application 3d.*

Comme illustré par la figure 1, une application 2d ou 3d se décompose toujours en 3 étapes majeures :

1. la phase d'initialisation *OnInit* qui permet de construire des éléments qui seront utilisé sur l'ensemble de l'application (ici par exemple vous aurez besoin d'une instance de la classe du jeu qui nécessite une phase d'initialisation et sera appelé tout au long du jeu) ;
2. la phase d'affichage *OnRender* qui est appelée à l'infini tant que l'utilisateur n'arrête pas volontairement l'application (en fermant la fenêtre par exemple). Notez bien que cette phase d'affichage se décompose également en plusieurs sous parties comme la mise à jour de votre jeu suivi de l'affichage du jeu ;
3. une dernière phase *OnRelease* appelée à la toute fin de l'application qui permet de détruire les éléments utilisés au cours de votre jeu (on pense ici à la mémoire allouée tout au long du jeu).

Nous allons reprendre ces concepts pour faire le Tétris. Pour construire votre jeu, il vous est fourni un code de base permettant l'affichage d'une "scène" en 2D. Aller le chercher sur *Moodle*, puis placez vous dans le répertoire *./ww* et compiler le projet avec la commande *make tetris*.

Comme vous pouvez le voir, le code fourni contient déjà un certain nombre de classes organisées dans plusieurs répertoires (répertoire */src*). Seul le fichier *main* et le répertoire *tetris* nous intéresse. Sachez que les autres fichiers sont là uniquement pour gérer le fenêtrage, l'affichage et les événements clavier de notre jeu.

Pour ceux qui ont un PC personnel sous linux, installer les librairies glut avec la commande *sudo apt-get install freeglut3-dev*.

2.2. L'espace de travail

Si vous lancez une première exécution du code (commande *../bin/tetris.bin*), vous remarquerez qu'à l'écran s'affiche du texte et un carré de couleur. Tout le code permettant ce résultat se trouve dans la classe *CProjetTetris.h/cpp*. D'ailleurs, voici la liste des fichiers dans lesquels vous allez travailler :

- *CProjetTetris.h/cpp* : cette classe hérite d'une classe de plus haut niveau gérant le fenêtrage, affichage et événements clavier. Ce qui est important de comprendre, c'est que cette classe pilote votre application à travers les méthodes *OnInit*, *OnRender* et *OnRelease* et est dédiée à :
  - la création, mise à jour et destruction de votre jeu (classe *CTetrisGame.h/cpp*)
  - l'affichage de votre jeu
- la classe *CTetrisGame.h/cpp* : il s'agit de la classe gérant l'intégralité de la mécanique de votre jeu. Elle permet en particulier de gérer la construction/destruction d'une pièce, gestion des lignes à détruire, *etc*.
- la classe *CRandomizer.h/.cpp* est déjà entièrement codée, elle sert d'outils pour générer une nouvelle pièce aléatoirement.
- les autres fichiers (déjà entièrement codés) *TetrisUtils.h CTGameTable.h* sont des outils dont vous aurez besoin. On y reviendra pendant la construction du jeu.

Maintenant que vous connaissez votre environnement de travail, voyons comment concevoir le *tetris*.

3. Conception du jeu

Il existe un certain nombre de possibilités plus ou moins difficile pour concevoir ce jeu. Ici, nous vous en proposons une. **Il vous est demandé de la suivre scrupuleusement**.

3.1. Une histoire de repère

Information essentielle : toutes les coordonnées et indices que vous utiliserez à la fois dans la fenêtre et pour les

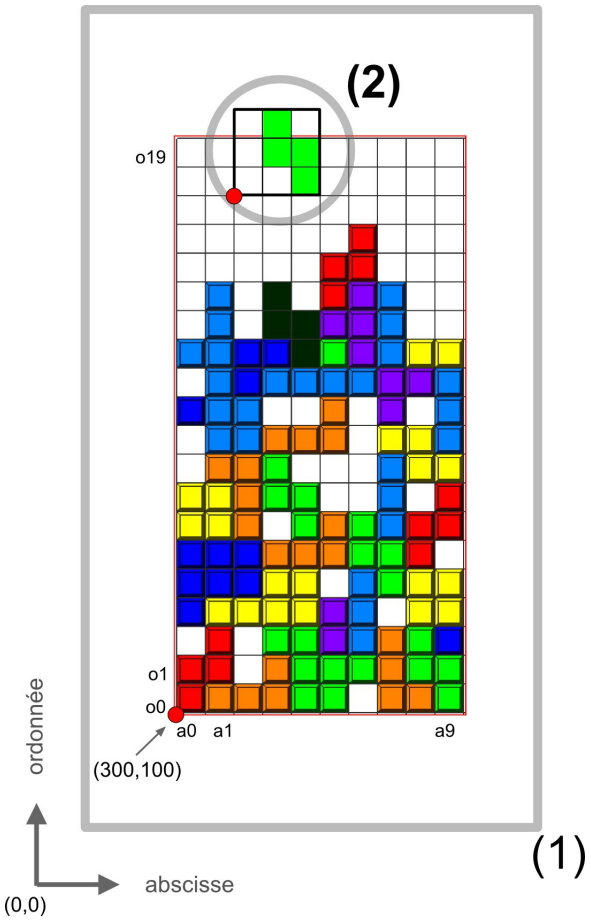

Figure 2: *Exemple de tétris. En (1) la grille du tetris. En (2) le tableau associé à une pièce. Les cercles en rouge représentent les "origines" des tableaux (en bas à gauche toujours !).*

tableaux ont comme repère $(0,0)$ en bas à gauche ! Ce point est illustré dans la figure 2 par :

- les coordonnées de la fenêtre $(0,0)$ et la position du tetris dans la fenêtre $(300,100)$
- les indices dans la grille du tetris $a0,a1,...$ (abscisse), $o0,o1,...$ (ordonnée)
- les points en rouges symbolisent l'origine des grilles du tetris et la grille associée à une pièce

3.2. Mécanique du jeu

Le jeu se décompose en 2 parties bien distinctes :

- la partie affichage (classe *CProjetTetris*) qui utilise le jeu tetris (classe *CTetrisGame*)
- la partie jeu (classe *CTetrisGame*) qui :
  - gère la grille du jeu (construction et mise à jour), représenté par le cadre (1) dans la figure 2.
  - gère le score (construction et mise à jour)
  - gère **la** pièce (classe *CPieceAbstract*) du jeu en cours (construction, rotation, collision et destruction), représentée par le cercle (2) dans la figure 2.

3.2.1. Séquence du jeu

Séquentiellement, voici comment se décompose le jeu (classe *CTetrisGame*) :

1. initialisation (à partir de *OnInit*)

   1.1 Initialisation/construction de la grille de dimension 10×20 avec des cases vides de couleurs blanches
   1.2 Initialisation du score
   1.3 Construction d'une pièce au hasard

2. Gestion du jeu (à partir de *OnRender*)
   définition d'une collision : quand une case est occupée par deux pièces ou quand une pièce dépasse un bord du jeu

   2.1 ACTION UTILISATEUR : Rotation / descente / déplacement de la pièce due aux évènements clavier
   2.1.1 collision avec bords / avec une autre pièce de la grille ? oui -> revenir dans la configuration précédente

   2.2 Mise à jour du jeu (méthode *Update* de *CTetrisGame*) : ACTION AUTOMATIQUE
   2.2.1 descente de la pièce
   2.2.2 collision avec le bas du jeu / ou avec une autre pièce de la grille ? non : fin de la méthode, oui : aller en (2.2.3)

   2.2.3 Oui à la question (2.2.2)
   a. game over ? oui : fin de la méthode (renvoyer l'état du jeu), non : aller en (2.2.3.b)
   b. annuler (2.2.1) (revenir dans la configuration précédente)
   c. insérer la pièce dans la grille (mettre à jour les cases non vide avec la couleur de la pièce)
   d. destruction des lignes pleines si nécessaire
   e. destruction de la pièce en cours
   f. mise à jour du score
   g. construction d'une pièce au hasard (fin de la méthode)

3. Fin du jeu (à partir de *OnRelease*)

   3.1 destruction du jeu (et donc des éléments qui ont nécessité une allocation mémoire)

4. Cahier des charges

Le détail des classes à développer se trouve à la figure 3.

A chaque classe codée DOIT être associé un lanceur *testNomClasse.cpp* qui effectue un test unitaire de la classe.

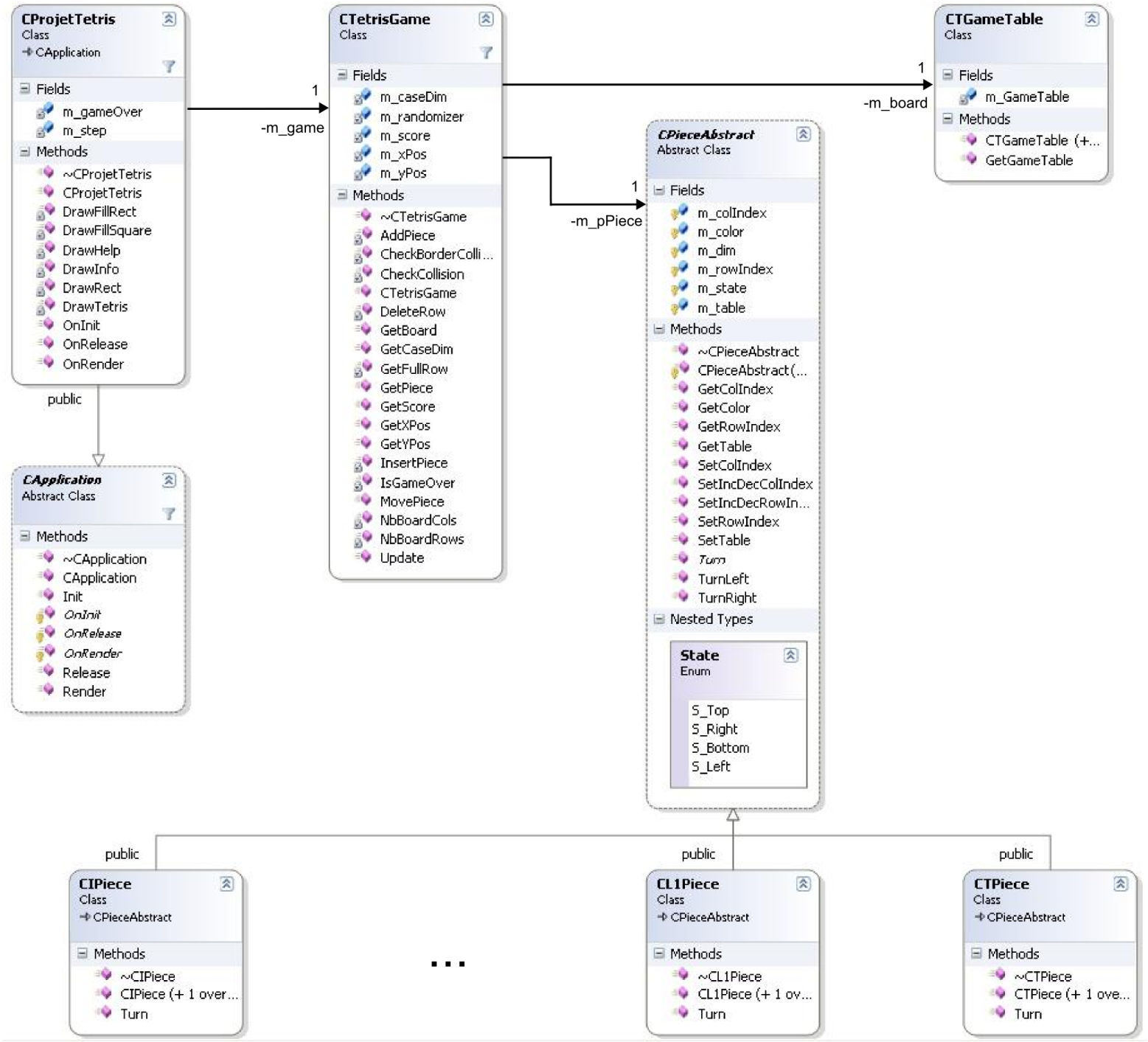

Figure 3: *Diagramme de classe du jeu.*

4.1. Partie 1

Commencer par la classe abstraite *CPieceAbstract* et ensuite le codage des 7 classes qui définissent les pièces du jeu : *CIPiece*, *CTPiece*, *COPiece*, *CL1Piece, CL2Piece, CZ1Piece, CZ2Piece*.

Tester la classe au terminal mais aussi à travers son affichage dans la fenêtre graphique en utilisant la méthode *DrawTetris()* de la classe *CProjetTetris*. Faites un répertoire de test avec les classes *CProjetTetris* + le fichier de test (*testCUnePiece*) contenant le lanceur.

4.2. Partie 2

Codage de la classe *CTGameTable* et ensuite de la classe *CTetrisGame* SANS collisions.

Pour la classe *CTetrisGame*, procéder suivant cet ordre :

1. coder les méthodes simples : accesseurs essentiellement.
2. coder la méthode *InsertPiece()* qui insère la pièce courante dans la grille Tetris. Cela signifie mettre à "true" les cases de la grille occupées par la pièce et donner à ces cases la couleur de la pièce (voir *CTGameTable*). Tester en affichant cette grille dans la fenêtre graphique (méthode *DrawTetris()* de la classe *CProjetTetris*) et observer l'apparition de la pièce dans la grille du jeu.
3. méthode *GetFullRow()* : doit retourner le numéro d'une ligne de la grille qui est occupée sur toute sa longueur par des pièces (-1 sinon).
4. méthode *DeleteRow(unsigned int rowIndex)* : doit supprimer de la grille la ligne passée en paramètre. Toutes les autres lignes doivent se décaler de 1 case vers le bas de la grille.
5. méthode *IsGameOver()* : retourne vrai si le sommet de la pièce dépasse le haut de la grille.
6. méthode *AddPiece()* : consiste d'une part à tirer au hasard une nouvelle pièce et, d'autre part, à créer cette pièce.
7. méthode *MovePiece (PieceAction action)* : en fonction de l'ACTION UTILISATEUR (*PieceAction*), répercuter le mouvement sur la pièce. Ne PAS encore se pré-occuper des collisions (bords et autres pièces) pour l'instant. Cette méthode renvoie "GameOver" si la partie est terminée, "Collision" si la pièce est entrée en collision avec la ligne du fond ou une autre pièce ou "OK" si le mouvement s'est réalisé sans aucune collision. Tester en affichant la pièce dans la fenêtre graphique (méthode *DrawTetris()* de la classe *CProjetTetris*) et observer le mouvement de la pièce suite aux actions de l'utilisateur récupérées dans la méthode *OnRender()* de la classe *CProjetTetris*.
8. méthode *Update(unsigned int step)* :

   - si step = 0 (délai d'attente écoulé) et aucune pièce encore dans le jeu, créer la pièce en appelant *AddPiece()*,
   - sinon, si step = 0 (et pièce existante), forcer la descente de la pièce de 1 case vers le bas de la grille en appelant *MovePiece(...)*. Tester en appelant AUTOMATIQUEMENT (à chaque fois que step = 0) *Update(...)* dans la méthode *OnRender()* de la classe *CProjetTetris* et observer la descente de la pièce dans la fenêtre graphique (méthode *DrawTetris()* de la classe *CProjetTetris*).

A CE STADE, votre jeu fonctionne à peu près :

- Bien vérifier que la méthode *DrawTetris()* de la classe *CProjetTetris* est complète : affichage de la grille Tetris + de la pièce (courante) dans la fenêtre.
- Dans la méthode *OnRender()*, enchaîner correctement les appels à *Update, MovePiece et DrawTetris* (dans l'ordre).
- Limite : aucun contrôle des collisions et aucune insertion de pièce pour l'instant.

4.3. Partie 3

Terminer le codage de la classe *CTetrisGame* AVEC collisions.

1. Écrire *CheckBorderCollision()* : renvoie vrai si après une action utilisateur, la pièce DÉPASSE les bords du jeu à droite ou à gauche. REPLACER la pièce à la position qui précède la collision.
2. Écrire *CheckCollision()* : renvoie vrai si après une action utilisateur OU action automatique, la pièce DÉPASSE le fond du jeu ou se SUPERPOSE (partiellement) à une pièce existante (à gauche, à droite ou en bas).
3. Modifier la méthode *MovePiece* de la manière suivante : après le mouvement utilisateur, appeler *CheckBorderCollision* (qui replace éventuellement la pièce à la position qui précède la collision) puis *CheckCollision*. Si *CheckCollision* renvoie vrai alors REPLACER la pièce à la position qui précède la collision et tester "GameOver". Suivant le résultat renvoyé par *CheckCollision* et l'appel de *IsGameOver*, retourner le message "OK", "Collision" ou "GameOver".
4. Modifier la méthode *Update* de la manière suivante : Si *MovePiece* renvoie "Collision" il faut insérer (*InsertPiece()*) la pièce dans le jeu et éventuellement supprimer la dernière ligne pleine. Sinon, cette méthode retourne "GameOver" ou "OK".

5. RENDU

Votre projet doit impérativement compiler et s'exécuter sous Linux et sur les machines du réseau "Enseignement".

Le projet se déroule sur 6 X 1h30 et fera l'objet d'une note de contrôle continu (15 pts). Il est à réaliser en binôme et l'archive sera déposée sur Moodle le vendredi 21 février à 23h55 au plus tard. A partir du samedi 22 février, chaque jour de retard entraîne une pénalité supplémentaire de 2 points.

L'archive doit contenir :

- un répertoire /src qui contient tous les fichiers *.cpp et *.h de l'application, y compris les fichiers fournis et les fichiers de test (*testCTPiece* etc.),
- un fichier makefile qui contient l'ensemble de toutes les instructions de compilation du projet et qui place les fichiers compilés dans le répertoire /bin,
- un fichier readme :
  - qui décrit l'état d'avancement du projet,
  - qui reprend clairement la liste des classes qui peuvent être compilées et testées grâce au makefile fourni.

Le fichier makefile permettra de compiler séparément les différents lanceurs : d'une part les lanceurs qui testent chacune des classes développées et d'autre part le lanceur final (*main.cpp*) qui exécute l'application finale (*tetris.bin*).