Projet C++

Tetris

Touraya El Hassani
Adjanie Kengni

Encadré par : Cecile Braunstein

19 janvier 2019

Table des matières

Table des figures

1 Description de l'application développée

L'application développée se présente sous forme de jeu, avec une interface graphique. Ce jeu est "Tetris". Il est composé d'un board où des pièces de formes différentes (de couleur jaune) descendent de haut en bas dans le board. Durant cette descente, le joueur peut déplacer les pièces latéralement et leur faire effectuer une rotation sur elles-mêmes jusqu'à ce que la pièce touche une autre pièce déjà présente ou le bas du board. Le but est de compléter toutes les lignes au fur et à mesure. Une fois la ligne complétée, elle disparait et laisse la place à la ligne au dessus. Dans notre jeu, le but est de detruire dix lignes pour pouvoir passer au niveau suivant. Mais si les lignes ne sont pas détruitent et qu'une pièce touche le haut du board alors le joueur a perdu.

Pour le bon fonctionnement de l'application, il est nécessaire d'installer le module SFML , qui est déjà installé sur les ordinateurs de l'école. Puis il suffit de lancer les lignes de commandes suivantes :

> make

> ./tetris.exe

Une fois le jeu lancé, on remarque une bande-son que l'on peut trouver dans le dossier assests/ mais également des images des tetriminos (les pièces) ainsi que le fond du board. Pour commencer à jouer, il suffit de suivre les instructions sur le board : pour commencer la partie, il faut appuyer sur ENTER (sur le clavier), si l'on souhaite faire une pause pendant la partie, il faut appuyer sur P, pour reprendre le jeu juste après, il faut re appuyer sur P. Une fois le jeu fini, c'est à dire quand le joueur a perdu, il suffit d'appuyer sur ENTER pour quitter le jeu.

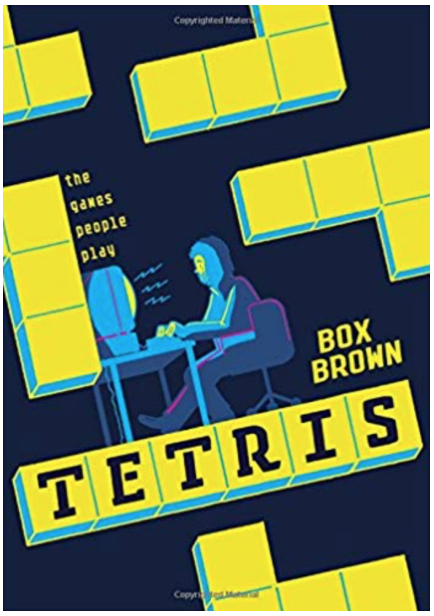

FIGURE 1 – Tetris

2 Utilisation des contraintes

Voici notre diagrammme de classe UML complet :

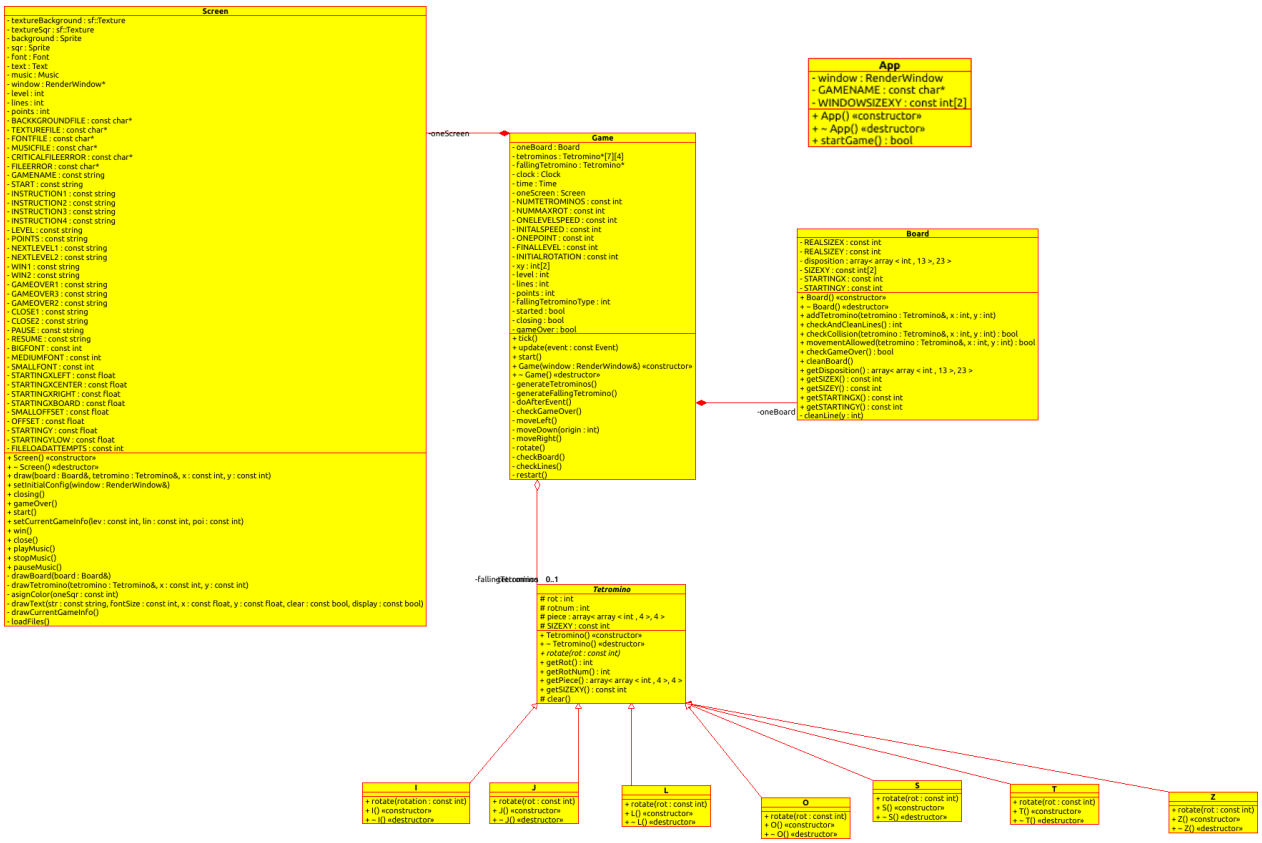

FIGURE 2 – Diagramme de classe UML

Comme on peut le voir dans la figure ci-dessus, nous avons eu besoin de **12 classes** et nous avons **2 niveaux d'hiérarchie** au niveau de la classe Tetrimino avec 7 classes pour les différentes pièces. Notre dossier comporte un fichier **Makefile** qui facilite la compilation du code. De plus, notre code a été **commenté** pour faciliter la compréhension de quiconque qui veut le comprendre. Une fois le code lancé, nous remarquons qu'il n'y a **pas d'erreur avec Valgrind**.

Notre code contient un **conteneur array** qui est un conteneur qui encapsule des tableaux de taille constante (connue à la compilation). Ce conteneur se trouve dans la classe Tetromino et sert à stocker la taille du tetromino (de taille 4) pour pouvoir le dessiner soit en forme de T, de O, de I etc.

L'intégralité de l'application et ses éventuels révisions se trouvent sur notre dépôt git : `https://github.com/Tourayaeh/Tetris`

3 La classe Tetromino

Cette classe permet de représenter les pièces du jeu et gère leur rotation. Tetris comporte 7 types de pièces d'où nos 7 classes hierarchisées de la classe Tetromino. Chaque pièce est composée de 4 blocs et sont appelées suivant leur forme O, I, S, Z, L, J, T comme on peut le voir dans la figure ci-dessous.

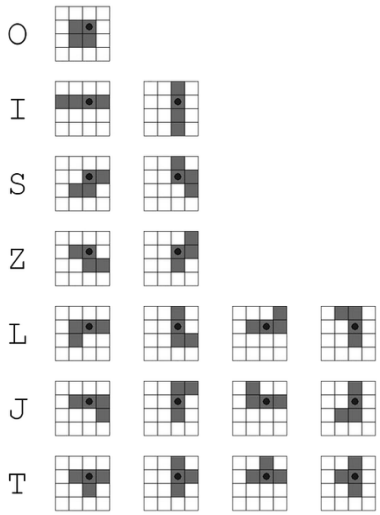

FIGURE 3 – Piece du Tetris

Ci-dessous le diagramme UML de la classe Tetromino et sa hierarchisation :
IMAGE
Comme on peut le voir, le point sur la pièce dans le bloc représente son point de pivot.
Pour représenter ces pièces dans notre code, il suffit de les représenter dans un tableau à 2 dimensions n*n de longueur et de largeur 2. Il faut mettre la valeur 1 là ou il y a le bloc normal et 2 pour le bloc de pivot. De plus, il ne faut pas oublier d'ajouter la couleur de la piéce : dans notre cas, la couleur JAUNE.
Exemple avec la pièce O :

```
void O::rotate(int rotation)
{
        if (rotation == 1)
        {
                clear();
                piece[0][0] = colors::YELLOW;
                piece[1][0] = colors::YELLOW;
                piece[0][1] = colors::YELLOW;
                piece[1][1] = colors::YELLOW;
        }
}
```

FIGURE 4 – Piece O

4 La classe Board

Cette classe représente l'aire du jeu, c'est elle qui définira les méthodes de manipulation des pièces (déplacements, rotations) et la logique principale du jeu (le jeu est-il fini ? , y a t'il des lignes à supprimer ? etc.), la gestion des collisions...

Le Tetris classique comporte une aire de 20 lignes et 10 colonnes. Nous allons la représenter par un tableau à 2 dimensions de 20 lignes et 10 colonnes.

Certaines de nos méthodes sont :
- void addTetromino : c'est cette methode qui permet l'ajout des pieces,
- int checkAndCleanLines : qui permet de voir si une ligne est rempli ; si oui, elle la supprime,
- bool checkCollision : qui permet de gérer les collisions avec, soit les murs sur le côté soit avec une autre pièce,

et d'autre méthodes qui permettent de vérfier que le jeu est fini, de nettoyer le board, de vérifier si le mouvement est autorisé ...

5 La classe Game

La classe Game qui contrôle essentiellement le jeu et son déroulement.

Dans cette classe, nous avons le constructeur qui génère tous les élèments pour le bon fonctionnement du jeu tel que :
- Afficher le board,
- Initialiser le score à 0 ainsi que le niveau,
- Initialiser les pièces.

Cette classe possède plusieurs méthodes tel que :
- **void start()** qui s'occupe de l'initialisation du jeu en sélectionnant la première et la prochaine pièce de manière aléatoire.
- une méthode **generateTetrominos** qui permet de créer les pièces au fur et à mesure alors que la méthode **generateFallingTetrominos** permet de définir la pièce suivante qui va apparaitre sur le board.

D'autre fonctions sont présentes pour vérifier si le joueur a perdu, pour éffectuer les rotations à droite, à gauche...

6 La partie dont nous sommes le plus fiers

```
//générer les pièces avec toutes leurs rotations
void Game::generateTetrominos()
{
        for (int i = 0; i < NUMTETROMINOS; i++)
        {
                int rot = 1;

                for (int j = 0; j <= rot; j++)
                {
                        switch (i)
                        {
                                case 0: { tetrominos[i][j] = new I; break; }
                                case 1: { tetrominos[i][j] = new J; break; }
                                case 2: { tetrominos[i][j] = new L; break; }
                                case 3: { tetrominos[i][j] = new O; break; }
                                case 4: { tetrominos[i][j] = new S; break; }
                                case 5: { tetrominos[i][j] = new T; break; }
                                case 6: { tetrominos[i][j] = new Z; break; }
                                default:
                                        break;
                        }

                        if (j == 0){
                                rot = tetrominos[i][j]->getRotNum();
                        }
                        else{
                                tetrominos[i][j]->rotate(j + 1);
                        }
                }
        }

}
```

FIGURE 5 – Méthode generateTetrominos()

La partie dont nous sommes le plus fiers est la méthode qui permet de génèrer les pièces avec toutes leurs rotations. Ici, NUMTETROMINOS est le nombre de pièce total que l'on possède, c'est à dire 7 pour notre jeu. La boucle parcours ces 7 pieces et en choisit une aléatoirement pour l'afficher dans le board. La piece choisie est une des pièces que l'on a decrit plus haut. Une fois que la pièce a été choisie, la boucle "if" génère la rotation de celle-ci. Si on l'enleve, il n'y a plus de rotation pour les pièces. Le joueur peut rotationner les pièces grâce aux flèches du clavier de son ordinateur.

7 Conclusion

Au moment du rendu du rapport notre application n'est pas complètement opérationnelle et necessite quelque révisions car quand on lance le jeu, un seul type de pièce est afficher pour jouer

car nous rencontrons un probleme avec le temps. En effet, nous avons mis un srand(time(NULL) dans la fonction **generateFallingTetromino** au lieu de la mettre dans la fonction principalle du jeu. Actuellement le jeu propose 50 niveaux. Cependant on peut étendre cela si le joueur le souhaite.

Références

[1] Tetris tutorial in C++ platform independent focused in game logic for beginners :
http://javilop.com/gamedev/tetris-tutorial-in-c-platform-independent-focused-in-game-logic\
-for-beginners/

[2] Projet TETRIS
http://sylvainmarechal.chez-alice.fr/iris1/tetris/html/tetris.html#
_Toc55357210

[3] Text-based Tetris game
https://codereview.stackexchange.com/questions/74293/text-based-tetris-game