|  |  |
| --- | --- |
| EMT | |
| Tetris | |
| Rapport de projet de l’atelier de programmation orienté objet |

|  |
| --- |
| Membrez Matteo  22/12/2020 |

Table des matières

[1 Introduction 1](#_Toc58916295)

[2 Glossaire 1](#_Toc58916296)

[3 Etapes du projet 1](#_Toc58916297)

[3.1 Choix du projet 1](#_Toc58916298)

[3.2 Apprentissage du GameFrameWork 1](#_Toc58916299)

[3.3 Jeu difficilement réalisable avec le GameFrameWork 1](#_Toc58916300)

[3.4 Réalisation de la partie graphique 2](#_Toc58916301)

[3.4.1 Frame 2](#_Toc58916302)

[3.4.2 Label 3](#_Toc58916303)

[3.5 Réalisation de la zone de jeu 3](#_Toc58916304)

[3.5.1 Implémentation de la grille de jeu 3](#_Toc58916305)

[3.5.2 Création des différentes pièces 3](#_Toc58916306)

[3.6 Gestion de la pièce qui tombe 3](#_Toc58916307)

[3.6.1 Détection du fond du tableau 3](#_Toc58916308)

[3.6.2 Détection des collisions avec d’autres pièces 3](#_Toc58916309)

[3.7 Timer 4](#_Toc58916310)

[3.8 Evenements d’appui sur les touches 4](#_Toc58916311)

# Introduction

Durant l’atelier de programmation orientée objet, nous avons dû programmer un jeu en C++ ou Java sans l’aide d’outils tel que Unity ou Unreal Engine. Je me suis orienté vers le jeu Tetris qui est l’un des jeux vidéo les plus emblématiques.

# Glossaire

C++

Le [C++](https://fr.wikipedia.org/wiki/C%2B%2B) est un langage de programmation permettant la programmation sous de multiples paradigmes (programmation procédurale, orientée objet, générique).

# Etapes du projet

## Choix du projet

Lors du début de l’atelier, notre enseignant M. Conus, nous a demandé de choisir un jeu simple en 2D que nous pourrions réaliser en C++ ou en JAVA. J’ai pensé à plusieurs jeux comme Tetris, Snake et Pac-Man, mais j’ai finalement choisi de m’orienter vers Tetris. Je me suis d’abord orienté vers le C++, puis sur du JAVA et je suis finalement revenu sur du C++. Ce choix a été effectué en fonction des conseils de notre enseignant et du GameFrameWork qu’il nous avait mis à disposition.

## Apprentissage du GameFrameWork

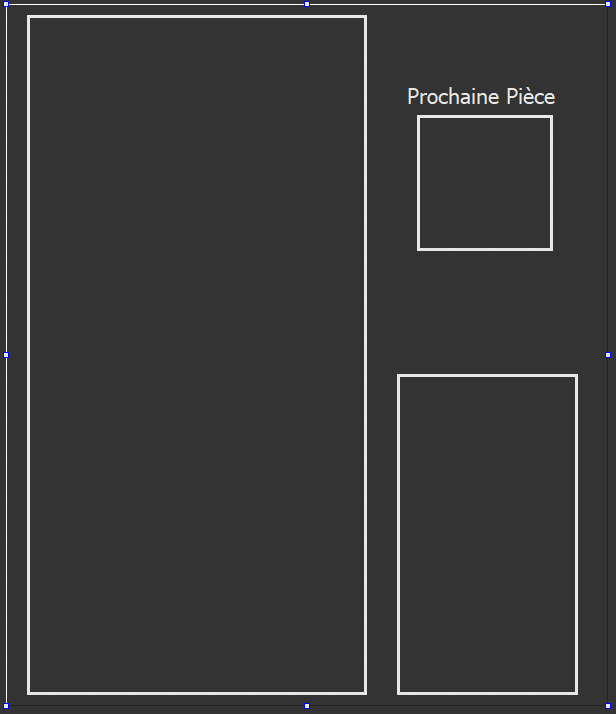
Notre enseignant nous a fourni un GameFrameWork et un tutoriel qui nous explique comment fonctionne le langage C++ et les fonctionnalités de base. Etant un peu perdu au début, j’ai réalisé le tutoriel et je me suis basé sur le code fournit pour débuter mon projet, ce qui m’a permis de mieux comprendre le fonctionnement du C++ et de pouvoir commencer à créer mon jeu.

## Jeu difficilement réalisable avec le GameFrameWork

Après avoir commencé à créer les bases de mon jeu via le GameFrameWork, et après avoir fait le point avec notre enseignant, nous en avons conclu que l’utilisation du GameFrameWork n’était pas des plus adaptées à la réalisation du Tetris. Du fait que le jeu se gère exclusivement dans un tableau, l’affichage sera juste une représentation de ce même tableau. Pour mieux aborder le développement du jeu, notre enseignant m’a conseillé de gérer le jeu dans un tableau (logique) et de redessiner une représentation du tableau à chaque fois que celui-ci change.

## Réalisation de la partie graphique

La partie graphique est réalisée dans un fichier **.ui** directement modifiable dans QtCreator.

Voici une représentation des différents éléments positionnés :

Frame

Label

La partie graphique est composé de deux éléments : les frames et les labels.

### Frame

Les frames sont des délimitations que l’ont peu affecter à une classe afin d’y apporter des modification par le code. Dans le cas du Tetris, j’ai affecté chaque frame à une classe qui va gérer l’ensemble des événements qui doivent se produire à l’intérieur.

### Label

Les labels sont de simples textes affichés à l’écran. Dans mon cas, je les utilise pour indiquer au joueur quelles informations sont affichées dans les différentes frames.

## Réalisation de la zone de jeu

La zone de jeu représente le jeu Tetris en lui-même, c’est dans cette zone que la plus grande partie logique va être gérée.

### Implémentation de la grille de jeu

Le jeu Tetris est géré entièrement dans deux tableaux à deux dimensions de 10x20 chacun. Le premier tableau va gérer la pièce qui est actuellement en mouvement et le deuxième tableau va gérer les pièces qui ont déjà été positionnées.

Afin de représenter la tableau graphiquement, il m’a fallut dessiner un quadrillage, pour cela, j’ai utilisé la fonction **paintEvent** qui est une fonction déjà existante que j’ai dû réimplémenter.

### Création des différentes pièces

Afin de gérer les 7 pièces de Tetris, j’ai créé un générateur pseudo aléatoire qui me permet de choisir l’une des pièces et de l’ajouter à mon tableau qui gère la pièce en mouvement. Pour définir si une case est remplie ou non, j’ai créé un type énuméré qui contient soit **FREE** si la case est vide, soit **FILLED** si la case est pleine.

## Gestion de la pièce qui tombe

### Détection du fond du tableau

La pièce qui tombe doit pouvoir reconnaître quand elle arrive au fond du tableau et s’il y a une pièce en dessous d’elle. Etant donné que ma pièce est constituée de 4 carreaux indépendants les uns des autres, j’ai dû en définir un qui vérifierait si le mouvement est possible. Pour cela, j’ai créé une fonction qui retourne la position du carreau le plus bas de la pièce par rapport au bas du tableau. Ceci m’a permis d’arrêter ma pièce lorsqu’elle arrivait à la fin du tableau.

Dès qu’une pièce arrive au fond de la grille, je la retire du tableau de la pièce mobile et je l’ajoute au tableau des pièces fixes.

### Détection des collisions avec d’autres pièces

Cependant, la manipulation décrite si dessus ne me permettait pas de gérer les collisions avec les autres pièces, pour cela j’ai dû vérifier à chaque déplacement si la case en dessous du positionnement de la pièce en mouvement était remplie ou non. Si c’était le cas alors la pièce était retirée du tableau de la pièce mobile et ajouté à celui des pièces fixes.

Dès qu’une pièce est placée, l’ajout d’une nouvelle pièce est commandé.

## Timer

Le timer est essentiel dans le jeu Tetris, c’est lui qui va permettre d’actualiser le tableau et donc, d’actualiser l’affichage du jeu. Je me suis heurté à un problème à la création de ce timer car je ne trouvais pas le moyen de le gérer. Je me suis tout d’abord orienté vers les **thread** mais il s’est avéré que cette solution me compliquait beaucoup la tâche. Après discussion avec notre enseignant, il m’a orienté vers la bibliothèque **QTimer** qui permet de gérer facilement l’exécution d’une tâche répétitive.

Le timer permet d’actualiser l’état du jeu et de faire descendre la pièce à chaque actualisation. Il permet également de vérifier s’il faut ajouter une nouvelle pièce.

## Evenements d’appui sur les touches

Pour contrôler la pièce en mouvement, il a fallu gérer l’appui sur les touches du clavier. Cet événement peut être géré grâce à une fonction déjà existante qui est **keyPressEvent** qu’il a fallu réimplémenter. J’ai un peu butté sur ce point car ça ne fonctionnait pas dans un premier temps, même en suivant les différents tutoriel qui expliquait le fonctionnement de cette fonction.

Puis j’ai finalement identifié le problème, il s’agissait du fait que la zone de jeu n’avait pas le focus au démarrage du jeu. Pour palier à ce problème, j’ai utilisé la fonction existante **setFocusPolicy** qui m’a permis de forcer le focus sur la zone de jeu à sa création.

## Rotation des pièces

Chaque pièce est constituée de 4 carreaux indépendants les uns des autres. Ces carreaux sont, en réalité, la représentation des cases remplies dans le tableau logique. Afin de gérer les rotations, j’ai dû utiliser deux étalons qui me renseignent sur la position X et Y de la pièce après chaque mouvement afin de redessiner la pièce en fonction de la future orientation à l’appui sur la touche de rotation. Chaque pièce possède un carreau qui est immobile et qui représente le centre de rotation. Les 3 autres carreaux viennent s’agencer autour de ce carreau.

A chaque rotation, il faut redéfinir les bordures de la pièce étant donné qu’elle n’a plus forcément la même hauteur et largeur.