EMT

Tetris

Rapport de projet de l’atelier de programmation orienté objet

Membrez Matteo
24/01/2021

TABLE DES MATIERES

1 INTRODUCTION

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

2 GLOSSAIRE

C++

Le C++ est un langage de programmation permettant la programmation sous de multiples paradigmes (programmation procédurale, orientée objet, générique).

Java

Le Java est un langage de programmation orienté objet

GameFrameWork

Projet contenant déjà plusieurs fonctions de base pour commencer à programmer un jeu.

3 ETAPES DU PROJET

3.1 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.

3.2 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 fourni pour débuter mon projet, ce qui m'a permis de mieux comprendre le fonctionnement du C++ et de pouvoir commencer à créer mon jeu.

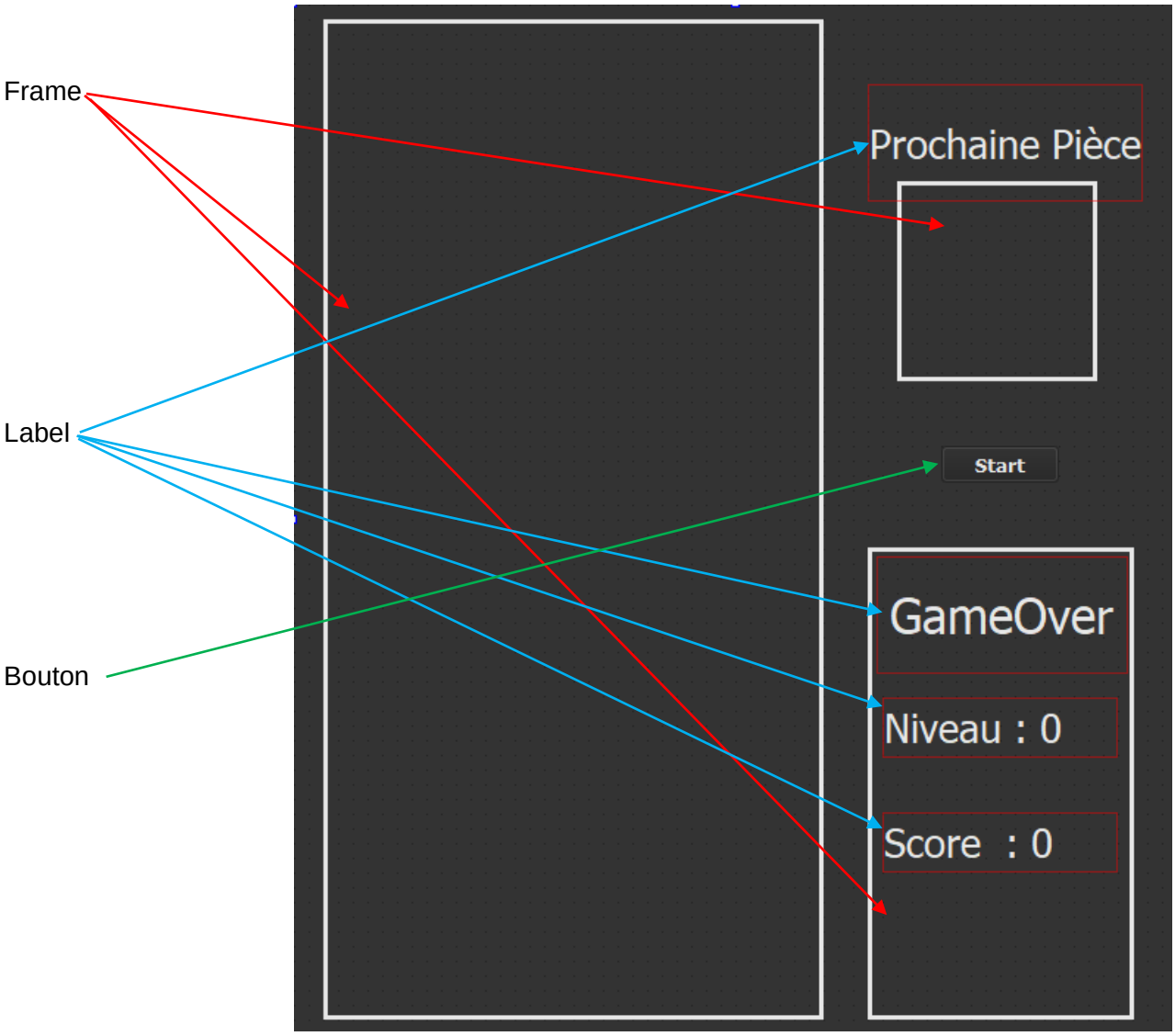
3.3 JEU DIFFICILEMENT REALISABLE AVEC LE GAMEFRAMEWORK

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.

3.4 REALISATION DE LA PARTIE GRAPHIQUE

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

Voici une représentation des différents éléments positionnés :



La partie graphique est composée de trois éléments : les frames, les labels et les boutons.

3.4.1 FRAME

Les frames sont des délimitations que l'on peut affecter à une classe afin d'y apporter des modifications 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.

3.4.2 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.

3.5 REALISATION 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.

3.5.1 IMPLEMENTATION 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 le tableau graphiquement, il m'a fallu dessiner un quadrillage, pour cela, j'ai utilisé la fonction **paintEvent** qui est une fonction déjà existante que j'ai dû réimplémenter.

3.5.2 CREATION DES DIFFERENTES PIECES

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.

3.6 GESTION DE LA PIECE QUI TOMBE

3.6.1 DETECTION 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.

3.6.2 DETECTION DES COLLISIONS AVEC D'AUTRES PIECES

Cependant, la manipulation décrite ci-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.

3.7 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.

3.8 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 buté sur ce point car ça ne fonctionnait pas dans un premier temps, même en suivant les différents tutoriels qui expliquaient 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 pallier 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.

3.9 ROTATION DES PIECES

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.

3.10 SUPPRESSION DES LIGNES COMPLETES

3.10.1 SUPPRESSION DES LIGNES

Dans Tetris, lorsqu'une pièce vient compléter une ligne, celle-ci est supprimée. Pour réaliser cela, il a fallu vérifier si une ligne est complète à chaque fois qu'une nouvelle pièce est positionnée dans le tableau des pièces fixes et vider toutes les cases de la ligne si c'est le cas.

3.10.2 DESCENDRE LES LIGNES

Une fois la ligne supprimée, il faut abaisser toutes les lignes au-dessus de la ligne supprimée. Pour cela il suffit de parcourir le tableau depuis la ligne supprimée et d'incrémenter la position sur l'axe des Y à chaque fois qu'une case remplie est trouvée.

3.11 GAME OVER

Le Game Over intervient lorsqu'une pièce est **posée** au **sommet** d'une des **colonnes**. Lorsque le joueur perd, la grille de jeu est vidée, le bouton « Start » est à nouveau cliquable et un message est affiché pour informer le joueur que la partie est terminée. Le score et les niveaux sont remis à zéro au moment où le joueur relance la partie.

3.12 SCORE & NIVEAUX

3.12.1 SCORE

Le score du joueur augmente quand de nouvelles lignes sont supprimées. Plus le nombre de lignes supprimées en même temps est grand (4 lignes max), plus le nombre de points acquis est conséquent. L'augmentation du score dépend également du niveau dans lequel se trouve le joueur. Le calcul du score est effectué de la manière suivante :

Nombre de lignes supprimées	Calcul
1	**40 * (niveau + 1)**
2	**100 * (niveau + 1)**
3	**300 * (niveau + 1)**
4	**1200 * (niveau + 1)**

3.12.2 NIVEAUX

Les niveaux sont gérés en fonction du score du joueur. Les niveaux sont au nombre de 6 dans l'état actuel du jeu. Chaque niveau augmente la vitesse du jeu ce qui augmente la difficulté. Les niveaux (comme le score du joueur) sont affichés en tout temps au joueur afin qu'il sache où il en est. Palier de chaque niveau en fonction du score :

Niveau	Score nécessaire	Vitesse du jeu en millisecondes
0	**0**	**1000**
1	**500**	**800**
2	**1'000**	**600**
3	**5'000**	**500**
4	**10'000**	**400**
5	**20'000**	**300**
6	**40'000**	**200**

4 PROBLÈMES RENCONTRÉS

4.1 LA PIÈCE NE DESCEND PAS

Constatation : de temps en temps, lorsqu'une nouvelle pièce était créée, elle restait bloquée en haut de la zone de jeu. Ce bug n'apparaissait pas à chaque fois ce qui posait quelques problèmes de résolution.

1ère déduction : le problème provenait très certainement de la méthode permettant de faire descendre les pièces. Dans cette méthode, j'utilise un booléen qui permet d'empêcher la pièce de descendre si elle détecte une collision, le problème vient peut-être de cette variable.

Méthode de débogage :

1. Initialisation de la variable booléenne à ***true*** → Toutes les pièces restent bloquées en haut de la zone de jeu.
2. Réinitialisation de la variable à ***false*** à chaque fois qu'une pièce est posée → Le bug n'apparaît plus.

Résolution : la variable booléenne qui stocke l'événement de collision restait à ***true*** et la pièce ne descendait pas car une collision était détectée. Il a fallu réinitialiser la variable à ***false*** à chaque fois qu'une pièce est posée.

4.2 CARREAUX QUI REMONTENT EN HAUT DE LA ZONE DE JEU

Constatation : pour certaines pièces longues, après avoir effectué une rotation et avoir posé la pièce, les carreaux les plus bas de la pièce remontaient en haut du tableau.

1ère déduction : un problème dans la détection du carreau le plus bas sur lequel se baser pour stopper la progression de la pièce.

Méthode de débogage :

1. Utilisation du débogueur afin de vérifier l'index récupéré → L'index est mauvais

Résolution : un test pour définir le carreau inférieur de la pièce s'effectuait une fois de trop, ce qui veut dire que tout dépend de la pièce, l'index était plus petit et donc la pièce descendait plus loin.

5 ETAT DU PROJET

Le projet, dans son état actuel, me semble assez abouti. Les principales fonctionnalités ont été implémentées. Le jeu ne possède pas de gros bug et est très largement jouable. Cependant, certains bugs n'ont peut-être pas encore été découvert.

Le jeu ne peut, pour l'instant, être exécuté qu'à l'aide de l'exécutable créé. Il n'existe pas d'installeur pour bénéficier des raccourcis sur le bureau.

Le jeu est composé d'une fenêtre qui est décomposée en plusieurs parties :

- **La zone de jeu :** C'est là que se déroule le jeu
- **Le score :** C'est là que le score et le niveau actuel du joueur sont affichés
- **La prochaine pièce :** Affiche la pièce qui suivra celle qui est actuellement en jeu

Le jeu comporte également un bouton qui permet de démarrer ou de redémarrer la partie.

6 AMÉLIORATIONS ENVISAGEABLES

Le jeu Tetris peut comporter bon nombre d'améliorations envisageables. Voici quelques exemples de ces possibles améliorations.

6.1 PROCHAINE PIÈCE

L'affichage de la pièce qui va suivre est le seul gros point que je n'ai pas eu le temps d'implémenter. La zone d'affichage existe déjà mais le code n'est pas encore implémenté. Le but serait d'afficher dans cette zone une pièce aléatoire et de la déplacer dans le tableau de jeu au moment où il faut ajouter une nouvelle pièce.

6.2 AMÉLIORATION DES GRAPHISMES

Il serait possible d'améliorer le design des différents éléments sur le plateau de jeu. A l'image du bouton **Start**, la possibilité d'utiliser le langage CSS pour modifier le visuel des éléments permettrait de rendre le jeu plus agréable à regarder.

6.3 JEU EN PAUSE

L'ajout d'un bouton ou d'une touche permettant de mettre le jeu en pause serait une bonne amélioration. Le jeu serait évidemment caché pour éviter toutes sortes de triches.

6.4 MENU D'OPTIONS & ACCUEIL

Un menu permettant de personnaliser le jeu à sa guise pourrait être une bonne amélioration qui rendrait le jeu plus « convivial ». Les paramètres pourraient également proposer la possibilité de changer la difficulté du jeu en accélérant plus vite la partie.

Une page d'accueil pourrait être créée afin d'accueillir le bouton « Jouer » permettant de lancer la partie, et le bouton « Options » permettant de parcourir les options du jeu.

6.5 MULTIJOUEUR

L'ajout d'un mode multijoueur permettrait de diversifier le jeu. On peut alors imaginer un mode « 1 contre 1 » sur le même écran (comme dans les compétitions de Tetris), ou encore un mode en ligne permettant de jouer à plusieurs sans être au même endroit. Evidemment il serait encore possible de joueur en solitaire.

7 CE QUE LE PROJET M'A ENSEIGNÉ

En tout premier lieu, la création de ce Tetris m'a permis d'améliorer mes capacités à gérer les différents délais d'un projet. Cette pratique avait déjà été mise en œuvre dans les précédents travaux mais c'est toujours un bon « entrainement ».

J'ai aussi pu approfondir mes connaissances dans la gestion de projets grâce à l'outil GitHub (déjà utilisé dans de précédents projets), qui m'a rappelé à quel point peuvent être utiles ce genre d'outils.

Durant les modules, nous avons fait de la programmation orientée objet mais dans le langage JAVA, c'était la première fois que je programmais en C++ orienté objet. Au début la différence était un peu gênante par moments mais on arrive finalement assez facilement à s'adapter au langage.

Pendant la conception, il m'est arrivé à plusieurs reprises de ne pas savoir comment fonctionnait une fonction que je devais utiliser. C'est là que j'ai pris le temps de parcourir la documentation officielle de Qt qui est très complète et bien expliquée. Elle m'a permis de régler bon nombre de problèmes dans mon jeu.

J'ai également appris à créer un exécutable avec Qt Creator, c'est un point très important dans la création d'une application, c'est donc une bonne chose d'avoir vu comment le faire sur cet outil.

8 CONCLUSION

J'ai beaucoup aimé ce travail sur le jeu Tetris. Il m'a permis de découvrir l'énorme potentiel de la programmation, dans le sens où on peut faire à peu près tout ce qu'on veut sans réelle limite. Essayer de reproduire un jeu déjà existant permet d'avoir un but concret à atteindre et à ne pas se limiter car une chose peut être compliquée à reproduire. Le jeu doit ressembler au maximum à la version originale. J'ai également beaucoup apprécié le fait de découvrir plus en profondeur le logiciel QtCreator qui nous met à disposition bon nombre d'outils pour réaliser notre projet.