ENSTA -
École Nationale Supérieure de Techniques Avancées

Rapport du projet IN204 2020/1 -

Tetris

Fabricio VELLONE
Javier Andres MARTINEZ BOADA

13 février 2020

Table des matières

1 Introduction

1.1 Aperçu du projet

L'objectif était de créer un projet de jeu Tetris pour la réalisation du cours de programmation orientée objet (IN2014). L'objectif était de créer un jeu multijoueur en réseau et à un seul joueur, où le langage utilisé pour la mise en œuvre était le C++.

Pour aider à cette création, notamment en matière de multimédia et de connexion multijoueur (networking), on a utilisé une API open-source appelée SFML (Simple and Fast Multimedia Library). Il nous a fourni des outils pour travailler avec la manipulation des fenêtres, les audios, les sockets réseau TCP/UDP. Pour compléter la compréhension de ce puissant outil, un eBook a été utilisé pour le développement de jeux utilisant SFML[1].

1.2 Compréhension du Besoin

Afin d'atteindre l'objectif proposé, et après une évaluation préalable de ce qui pourrait être utilisé, nous avons commencé à structurer la hiérarchie du programme. Étant donné que pour la réalisation de ce projet, il serait nécessaire de créer une classe de "surveillance" de la transition de l'état de jeu. Il serait chargé de récupérer les informations pertinentes de chaque "vue" - nous appellerons toutes les scènes du programme telles que le titre, le menu, etc.

En outre, il a été jugé nécessaire d'implémenter certaines classes d'abstraction de composants tels que les étiquettes et les boutons, car leur implémentation récurrente simplifierait l'utilisation de classes personnalisées pour ces derniers. Ainsi, même avec l'utilisation des bibliothèques de base de la SFML - "Graphics" et "Networking", son utilisation a été simplifiée à un niveau supérieur (comme un "mini-framework"), où nous avons utilisé comme base pour construire les vues les boutons, les étiquettes, et une classe d'utilitaires, où le concept de relativité était abstrait par rapport à l'écran généré - puisque la SFML ne doit fixer que les pixels.

Toujours en ce qui concerne ce qui serait nécessaire pour composer le projet dans son ensemble, nous avons cherché à utiliser des matrices pour mettre en œuvre les petits morceaux, ou tétrominos - comme on les appelle. Et pour clore ce qui serait nécessaire pour créer le jeu, nous devrions créer une classe qui jouerait le rôle d'hôte du serveur et une autre qui serait juste le "client" pour recevoir les informations et rendre le multijoueur possible.

2 Compilation

Pour faire bon usage du programme, il doit être utilisé sous Linux (l'OS de développement était Ubuntu 18.04). Pour le compiler, vous devez télécharger les packs SFML (API graphique utilisée). Le téléchargement du fichier peut se faire dans le lien : `https://www.sfml-dev.org/download/sfml/2.5.1/`

Pour faire bon usage du programme, il doit être utilisé sous Linux (l'OS de développement était Ubuntu 18.04). Pour le compiler, vous devez télécharger les packs SFML (API graphique utilisée). Le téléchargement du fichier peut se faire dans le lien :

Après avoir téléchargé le paquet nécessaire, il suffit d'accéder au dossier racine du programme par le terminal et d'utiliser la commande "make" pour construire l'application. Après avoir exécuté la commande précédente, il suffit de l'exécuter comme on le fait normalement sous linux, mais en vérifiant que la compilation se trouve dans le dossier "build" : "./build/program".

3 Description Fonctionnelle

Comme mentionné dans la section 1.2 après avoir compris ce qui serait nécessaire pour construire le jeu, les bases de la bibliographie et des cours ont été mises en pratique. L'idée du programme lui-même, est d'utiliser la classe de gestion pour sauvegarder les informations pertinentes de chaque vue, et d'échanger des informations entre elles. Par exemple, pour pouvoir accéder à une option d'un menu et pouvoir y revenir après l'avoir saisie. Pour mettre en œuvre cette logique, nous utilisons une stratégie de type pile, où allié à cette classe de gestionnaires, nous mettons en œuvre un moyen de stocker les couches de vue (Figure 1), et de les supprimer de la file d'attente lorsqu'elles sont utilisées, ils peuvent même tout supprimer pour fermer l'application.

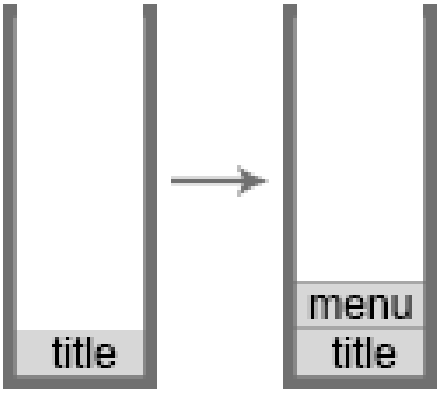

FIGURE 1 – Example de stockage

Ensuite, au moyen de "pops", "stacks" et "pushs", nous pouvons manipuler la fenêtre qui est utilisée en ce moment avec vos informations que nous voulons conserver.

Le jeu commence par une vue de titre, où une touche est censée passer à la vue suivante : le menu. Vous pouvez y choisir parmi 4 options :

— Singleplayer
— Multiplayer
— Help
— Exit

Comme les noms eux-mêmes le disent déjà, accédez aux prochaines vues contenues dans le programme, selon le principe du "pop" et "push", où chaque fois qu'une vue est sélectionnée, la suivante est prise et placée sur la pile et celle sur laquelle on travaille est mise en œuvre.

L'affichage de l'aide ne contient que les commandes utilisées dans le jeu, l'accès multijoueur aux menus de création et de recherche du serveur. En ce qui concerne l'acteur unique, nous avons les travaux de construction les plus accentués. Où, pour concevoir chaque carré du plateau de jeu, il a fallu mettre en place une nouvelle classe qui en est responsable ; le plateau (20x10) est un vecteur d'entiers, où il stocke 1 pour rempli et 0 pour non rempli. Et en fonction de la position et de la couleur de l'image, selon les règles de vérification, elle est toujours appelée fonction chargée de dessiner le jeu à l'écran.

Les tetrominos eux-mêmes sont également constitués de vecteurs entiers, qui ont une pièce centrale comme base et à partir de là utilisent une coordonnée relative pour dessiner. La rotation est basée sur le principe que ces pièces qui entourent la base établie changent de position selon le principe (x ;y) = (y ;-x).

Pour établir la connexion, la bibliothèque du SFML a été utilisée pour apporter des modifications au réseau. Nous y avons utilisé des prises et un type de connexion UDP pour simplifier les choses, où le principe selon lequel les deux utilisateurs vont écouter en permanence, et s'ils remarquent qu'un autre essaie ou reçoit/connecte, font le contraire, Et ils n'ont pas besoin de beaucoup d'authentifications.

Tout au long du projet, plusieurs outils C++ ont été utilisés, tels que la notation d'objet etl'héritage - un peu partout, comme nous l'avons déjà mentionné et nous le remarquerons plusieurs fois dans celui-ci. Des connaissances telles que le Template étaient nécessaires pour permettre l'utilisation d'une gestion efficace du contexte. Pour faciliter de nombreuses utilisations du programme, telles que le déplacement des tétrominos, il a utilisé la surcharge des opérateurs pour simplifier la comparaison entre le tétromino et la matrice ou même entre le tétromino et lui-même pour se déplacer horizontalement. Certaines connaissances sur les Threads ont été appliquées pour permettre au serveur de rechercher des personnes en permanence.

4 Description de l'Architecture

En ce qui concerne l'architecture générale des classes dans le projet, les classes présentées dans la Figure 2 ont été utilisées, ce qui donne une vue d'ensemble du projet, en ce qui concerne les divisions faites dans le projet et les classes appartenant les unes aux autres, et leurs liens. Comme vous pouvez le voir, la classe State est utilisée pour créer toutes les vues générées dans le jeu, car comme on l'avait dit précédemment, elle gère les priorités et stocke les variables de projet importantes pour passer d'une vue à l'autre.

En tant que structure, elle a été divisée en grands groupes, qui sont ceux de l'utils, du game, des views, du network et des states. L'idée est que chaque secteur soit indépendant et non pas à l'intérieur de l'autre, pour faciliter sa mise en œuvre.

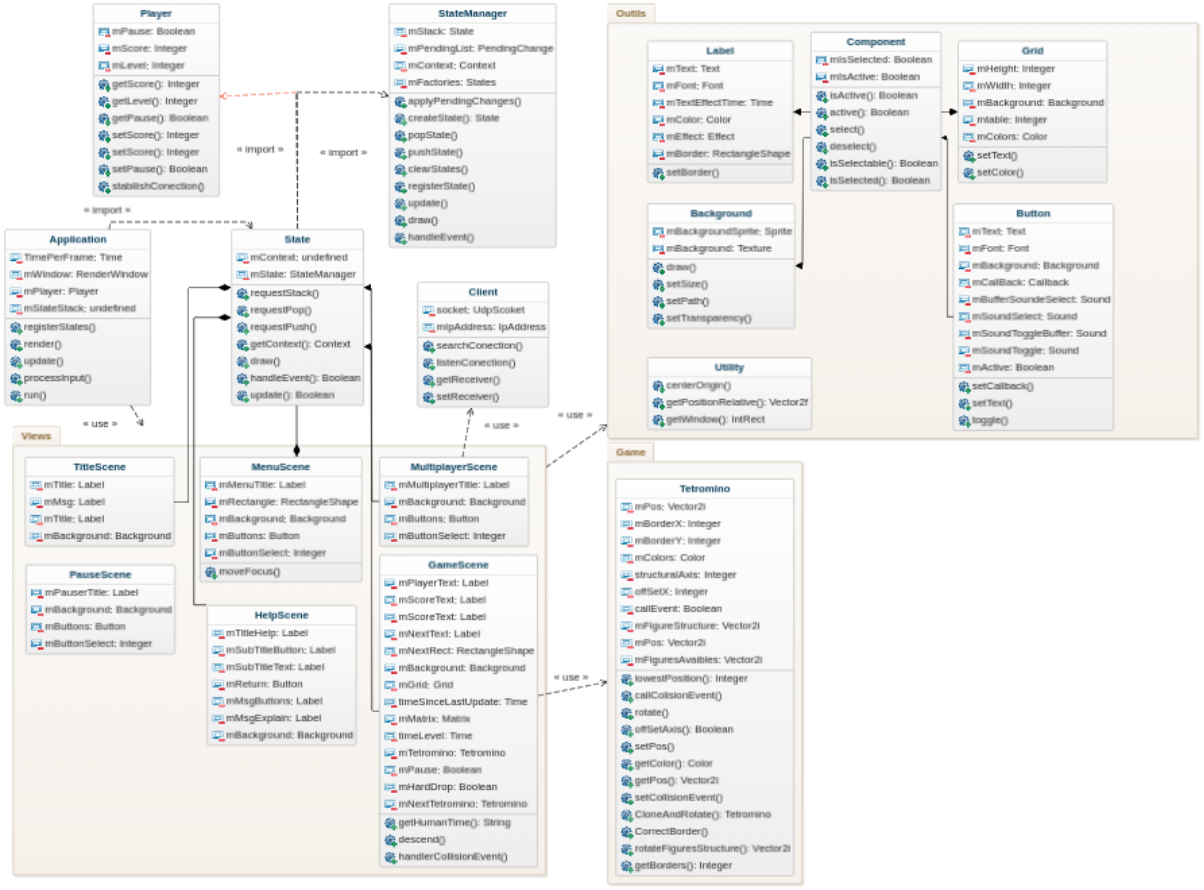

FIGURE 2 – Architecture des classes du projet

4.1 Classes StateManager et State

En tant que membre de la classe principale, la classe StateManager est responsable de la gestion de ce qui est en attente ou non. Il est responsable de la création des états (des vues), de leur affectation soit en "attente" -(empilés), "attente" lorsqu'il est retiré et préparé pour entrer comme vue principale, "actif" où par poussée il le met en exécution et par SFML, commence à le dessiner et à le faire passer comme champ d'exécution principal.

Pour la classe State, vous trouvez les valeurs stockées, c'est-à-dire que le StateManager fonctionne en conjonction avec la classe State, comme si l'un était responsable des fonctions de gestion et les autres du stockage, comme un grand centre administratif. Pour rendre le travail plus complet, le noyau de joueurs est ajouté à ce centre, comme on peut le voir dans la Figure 3 :

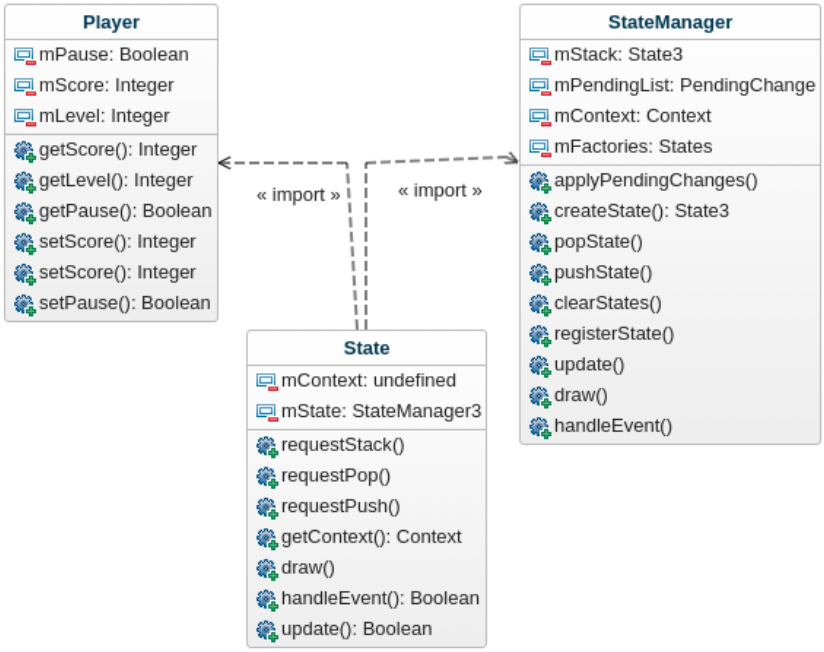

FIGURE 3 – Liens entre les States and composition

4.2 Classes des View

Pour les points de vue, ils partent tous du même principe. Tous sont importants, surtout dans la classe des joueurs individuels, car c'est dans cette classe que seront exécutés les plus grands nombres d'acier. L'idée de laisser chaque classe "indépendante" des autres, avec un certain niveau d'interaction, est donc nécessaire à ce stade. La Figure 4 démontre avec un peu plus de précision :

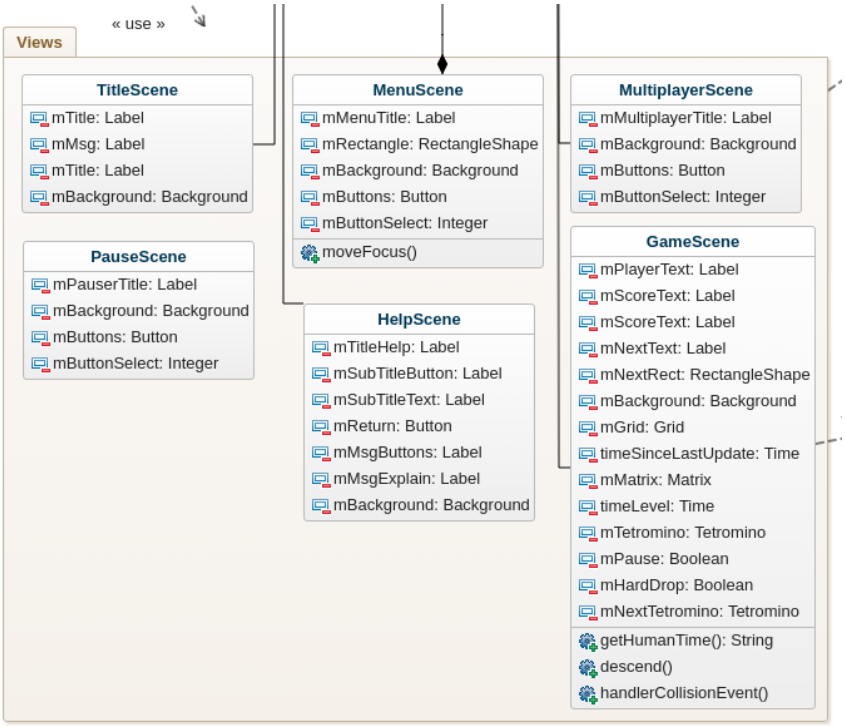

FIGURE 4 – Vision générale des Views

De nombreuses fonctions de dessin sont appelées à tout moment pour continuer à afficher les images dessinées à l'écran et, pour le jeu, donner l'idée de mouvement et de gravité.

Il est très important de souligner que chaque vue a un EventHandler spécifique pour vos besoins. Toutefois, les points de vue sont essentiellement contenus dans la logique suivante :

```
Result : Draws
Initialization;
while Fenetre est ourvert do
    Affiche background;
    Affiche labels;
    Affiche texts;
    Affiche choses pertinents;
    ...
end
Result : Updates
Contrôle permanent;
while Fenetre est ourvert do
    if constante de temps supérieure à dt then
        Effects;
    end
    reset constante de temps;
    ...
end
Result : EventHandler
while Fenetre est ourvert do
    if L'événement pour les utilisateurs a été appelé then
        switch Event do
            cas individuels;
        end
    end
    ...
end
```

Fondamentalement, les vues sont formées de ces 3 fonctions qui sont appelées à tout moment à effectuer tout ce qui est nécessaire, et à maintenir le bon fonctionnement des interactions entre les entrées et ce qui est montré à l'écran.

4.3 Les Outils

Comme base pour la création des vues et de la partie visuelle du projet, le dossier responsable des utilitaires s'occupe de faciliter les vues et de faire une abstraction et de "packager" plus de fonctions dans le même outil pour faciliter sa mise en œuvre. Par exemple, l'implémentation de la fonction dans l'utilitaire "getPositionRelative()" la rend

beaucoup plus facile à utiliser. La figure suivante nous donne une idée plus globale de ce dont il s'agit ce dossier.

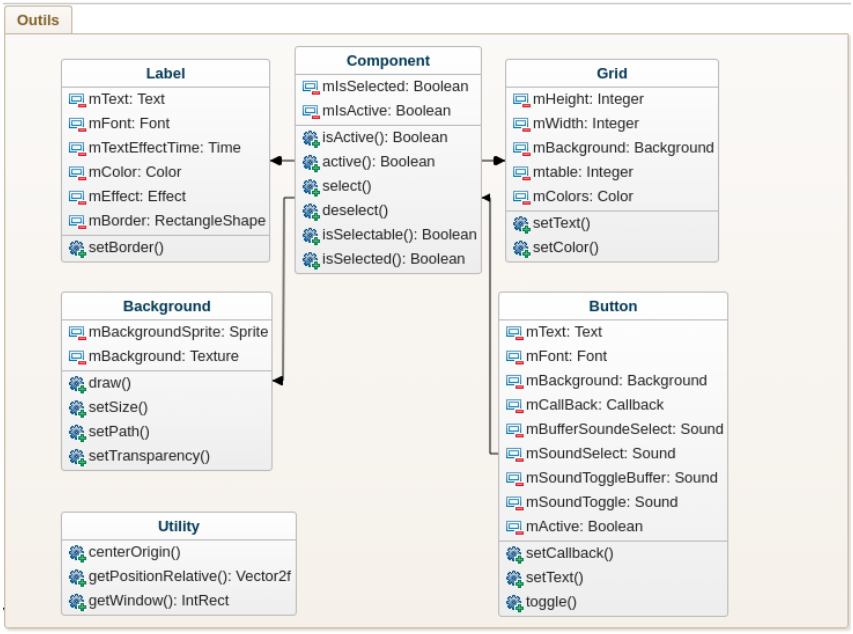

FIGURE 5 – Les classes outils

Cette fonction fonctionne comme suit :

Result : Utility : :getPositionRelative(proportions)
$x \leftarrow$distance maximale de X Divise par la proportion choisie;
$y \leftarrow$distance maximale de Y Divise par la proportion choisie;

4.4 Le Tetromino

Pour la classe responsable du jeu, Tetromino, comme on appelle la pièce Tetris, est chargé de dessiner de petits carrés dans toute la zone d'échantillonnage requise. Il est d'abord initialisé avec la taille de l'univers du jeu, puis il commence à tirer les pièces selon le code, en vérifiant à chaque instant s'il y a une collision, s'il n'y en a pas, la pièce sera tirée.

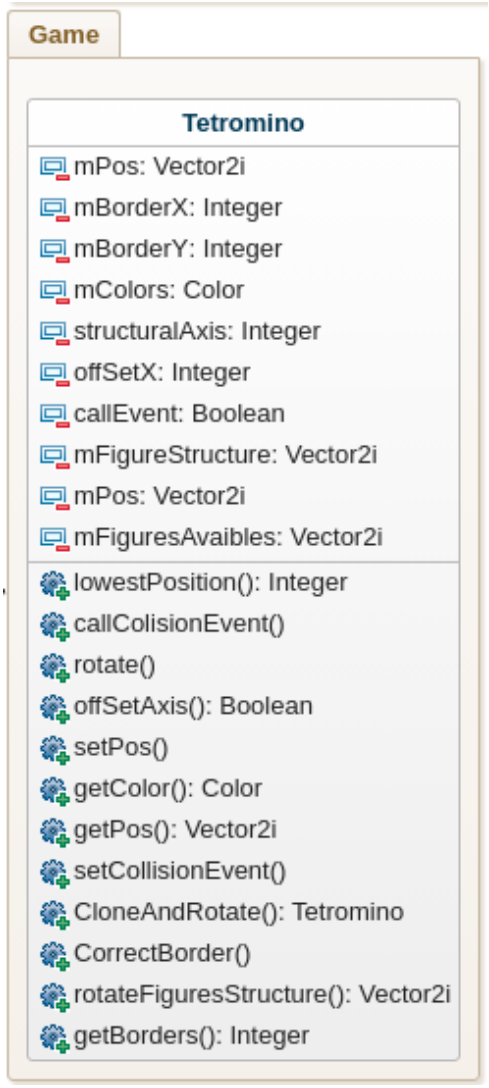

FIGURE 6 – Classe Tetromino

Dans la figure ci-dessus, il est possible de vérifier plus en détail les fonctions contenues dans cette classe et ses variables. Par conséquent, le pseudo-code relatif à la classe ressemblerait à ce qui suit :

```
Result : Le joue
while la fenêtre est ouvert do
    if temps d'échantillonnage supérieur à dt then
        Movement;
        if aucune partie dans l'espace suivant then
            faire le movement;
        end
        if Rotation then
            if aucune partie dans l'espace suivant then
                tetromino.x ← tetromino.y;
                tetromino.y ← −tetromino.x;
            end
        end
        temps ← Zero;
    end
    affiche tetromino ;
end
```

4.5 Le Serveur

Le mode de connexion choisi est l'UDP, étant donné sa facilité d'envoi et de réception des données, ce qui facilite la connexion entre les joueurs. Pour cela, il a utilisé les outils API, parvenant à créer un mini serveur de base pour l'échange de données. Le serveur fonctionne donc comme suit :

— En sélectionnant "Créer une connexion", un fil est créé en écoutant pendant 10 secondes quelqu'un qui veut se joindre au jeu.

— Un autre joueur doit choisir de se joindre, d'envoyer un message et de voir si quelqu'un attend la connexion.

— Lorsque les deux reçoivent la connexion, le jeu commence

Après le balayage, le serveur envoie des variables de contrôle aux deux joueurs où la vue du jeu revient, mais maintenant avec cela en mode multijoueur, prêt à être joué à deux.

5 Futures Mises à Jour

Pour les prochaines mises à jour du projet, des audios pourraient être placés tout au long du jeu afin de donner une meilleure expérience au joueur. Des effets pendant le jeu

serviraient également à améliorer l'expérience, comme la pièce fantôme pour montrer où elle s'insère ou les effets lorsque vous pouvez compléter les lignes

En ce qui concerne les performances du projet, il serait nécessaire de procéder à davantage d'arrêts autres que ceux déjà mis en œuvre par l'API utilisée pour accroître l'efficacité en ce qui concerne l'ordinateur

Pour la partie multijoueur, il est possible de faire un multijoueur local contre un bot, avec une connaissance un peu plus approfondie des algorithmes qui utilisent l'IA ou même un petit code qui résout le problème de la place de la pièce.

Vérifier la large utilisation des paquets, dans le cadre de l'échange exhaustif de données.

Références

[1] J. Haller, H. V. Hansson, and A. Moreira, *SFML Game Development*. Packt Publishing, 2013.