Projet programmation : Sliminos

Notre projet est disponible à l'adresse suivante : `https://gitlab.dsi.universite-paris-saclay.fr/fiona.hak/Projet_Tetris`.

Résumé

Ce projet avait pour but de créer un jeu Tétris en Java. Il est organisé en 3 parties : le modèle, le contrôleur et l'implémentation graphique. Nous avons suivi l'architecture suggérée pour le modèle (incluant les wall kicks) et la partie simple du contrôleur afin de faire passer les tests. Notre objectif était de créer un jeu original auquel nous souhaiterions continuer à jouer lorsqu'il sera fini : c'est pourquoi nous avons entièrement adapté la partie graphique (nous expliquons nos choix d'implémentation ci-dessous). Cela nous a permit d'obtenir une interface stylisée, basée sur la thématique d'un autre jeu, Slime Rancher [Figure 1]. Nous avons eu cette idée en regardant le wiki de tétris qui proposait un lien vers un jeu mettant en scène des personnages similaires [3].

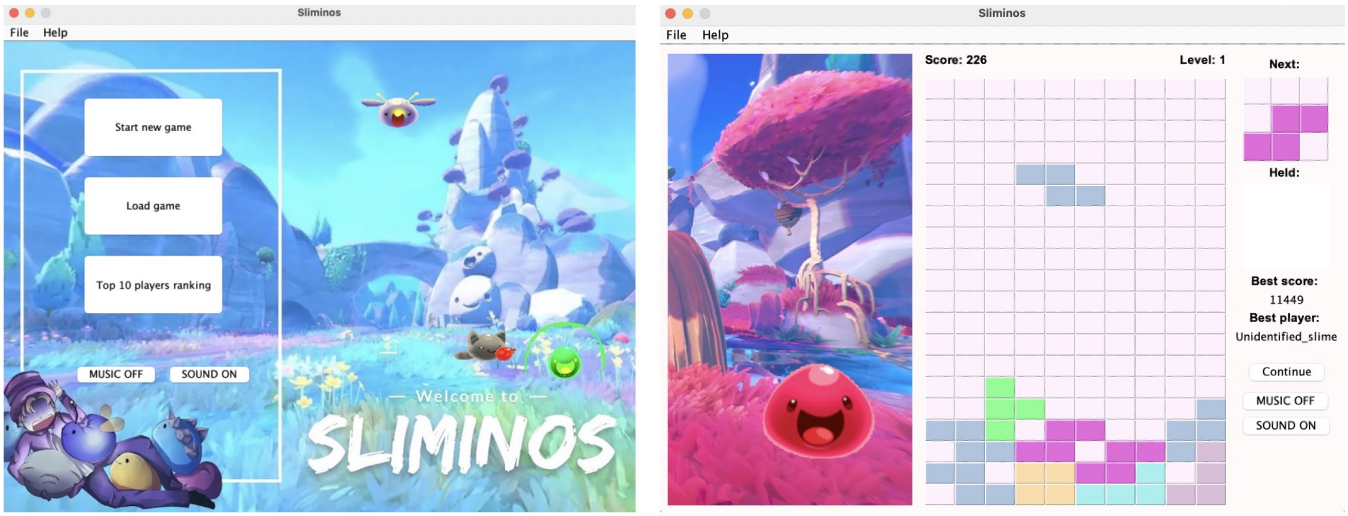

Figure 1 : Pages d'accueil et du jeu. L'accès aux boutons des fonctionnalités est identique sur les deux pages grâce à la barre de menu en haut de la frame.

Notre jeu est composé de diverses fonctionnalités qui nous semblaient pertinentes. Tout d'abord, au niveau du jeu en lui même, il y a la possibilité de sauvegarder une partie à trois emplacements différents, chargeables à nouveau en un clic, sans avoir à chercher un fichier en particulier dans ses répertoires en local [Figure 2.A.]. De manière analogue, il est donc possible de charger ces parties depuis ces trois emplacements (option load from save) [Figure 2.B.]. Il est néanmoins donné le choix au joueur de charger une partie depuis son ordinateur grâce à l'option load from file [Figure 2.C.].

Nous avons rendu l'accès à ces options plus simple en créant une barre de menu en haut de la fenêtre de jeu. Celle-ci contient en plus une option pour revenir sur la fenêtre du home depuis celle du jeu, une option pour sortir complètement du jeu (exit), et un menu "help" qui affiche un guide avec les différentes touches à utiliser pour jouer [Figure 2.C.].
Toutes ces options sont aussi accessibles grâce à des raccourcis notés à côté de leur nom [Figure 2.C.].

Puis, du côté des fonctionnalités, nous avons développé un système de "ranking" qui permet au joueur qui fait un score s'incluant dans le top 10 enregistré de l'afficher dans la liste du podium disponible sur le home ("top 10 players ranking") [Figure 2.D.]. Le meilleur score et le pseudo du meilleur joueur sont aussi affichés sur la page du jeu, sous le tétromino hold [Figure 1].
Nous avons rendu disponible un bouton pause sur la page du jeu.
Pour finir, nous avons implémenté de la musique et du son, qui peut être activé/désactivé à tout moment sur les deux fenêtres. Le son correspond à un bruit que font les tétrominos lors d'un hard ou soft drop [Figure 1].

Du côté de l'amélioration graphique, nous avons placé des images sur notre thématique en fond du home et à gauche sur la page du jeu. Les slimes (petits personnages en boule) sont tous animés et bougent en boucle sur le home. Sur la page du jeu, l'animation est plus poussée car elle est déclenchée par une action du joueur. Le slime sur la page du jeu en Figure 1 est le personnage affiché par défaut. Lorsque l'on fait un hard drop, l'animation va changer et le slime va "manger un tétromino". Enfin, lorsque la partie est finie, une nouvelle animation se déclenche.
Sur la fenêtre du jeu, nous avons également choisi d'afficher le tétromino suivant, le tétromino "hold" (si aucun n'est retenu, un carré blanc seul s'affiche comme sur la figure 1), le score courant et le niveau.

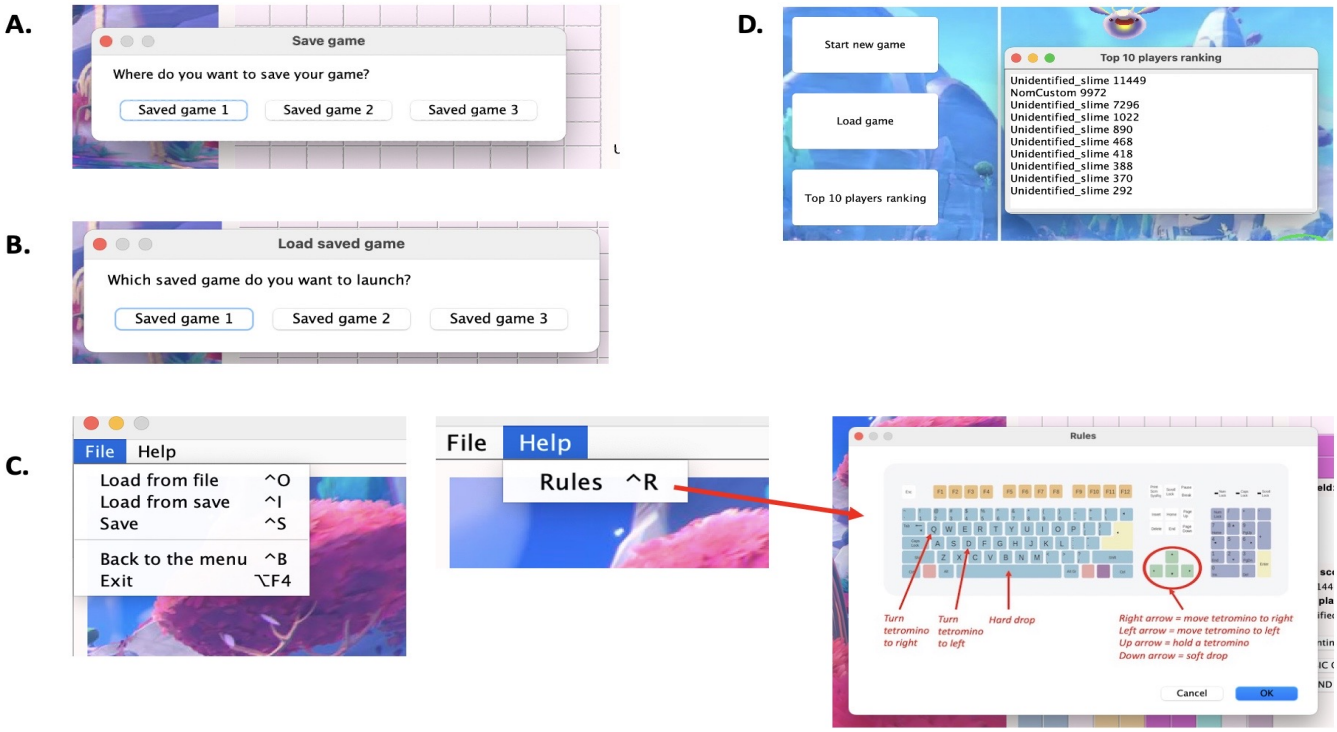

Figure 2 : Fonctionnalités du jeu. A. Sauvegarder le jeu dans des emplacements spécifiques. B. Charger une partie depuis ces emplacements. C. Barre de menu. D. Le top 10 des meilleurs joueurs.

Les images ont été toute récupérées (sauf une) sur deviantart et sont distribuées sous licence creative common qui nous permet de les utiliser sous leur forme initiale, sans utilisation commerciale. L'image modifiée du slime qui mange un tétromino sur la page du jeu est distribuée sur le wiki du jeu original Slime Rancher sous la licence "Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)" qui nous permet de la partager et de l'adapter [2]. La musique est distribuée sous licence "CC0 1.0 Universal (CC0 1.0) Public Domain Dedication" [1].

Point sur les packages

Le projet a été testé sur MacOS, avec IntelliJ sous Open JDK 17.

0.1 Le package model

Pour ce package, nous avons suivi de très près les suggestions d'implémentation. Nous avons implémenté toutes les méthodes de grid, score et tetromino. Tous les tests proposés pour le modèle (TetrisGridTest, TetrominoProviderTest, MarathonScoreComputerTest et TetrominoTest) passent, y compris les wall kicks.
Nous avons ajouté un test dans TetrominoTest (testRotate) car nous avons ajouté une fonction dans TetrominoImpl qui gère la rotation des tétrominos (prend la transposée et l'inverse de la matrice traitée pour obtenir toutes les formes). C'est la seule modification apportée sur les suggestions. De plus, un test dans TetrisGridTest a été ajouté (testHardDropEmpty) pour une question de debuggage, car nous avions un soucis au niveau du hard drop qui faisait sortir les tétrominos de la grille si celle-ci était vide.

Au niveau de la structure, elle est restée la même que celle proposée, on a donc : TetrominoProviderImpl qui implémente l'interface TetrominoProvider, TetrominoImpl qui implémente Tetromino, ScoreComputerImpl

qui implémente ScoreComputer, TetrisGridImpl implémente TetrisGrid et SynchronizedView qui implémente TetrisGridView. Il n'y a pas d'héritage. L'architecture est donc assez binaire avec une classe (sauf TetrisCoordinates) qui implémente une interface.

0.2 Le package control

Pour ce package, étant donné qu'il est directement lié au view, nous avons modifié quelques parties de son architecture. Le changement majeur est au niveau du VisualGamePlayer, que nous avons laissé vide car au vu de l'implémentation qu'on a fait sur le view, il est devenu inutile.
Concernant la partie logique, nous n'avons pas fait de changement. L'ensemble des tests de SimpleGameManagerPlayerTest passent et aucun n'a été ajouté. Les méthodes du game manager ont toutes été implémentées.
Au niveau de la structure, nous l'avons laissé presque intacte, même si le VisualGamePlayer pourrait être supprimé. Il y a donc une nouvelle classe Game, une classe abstraite, non instanciable, pour le code de base, qui nous permet d'éviter de dupliquer du code entre un mode visuel et simple. Elle regroupe les fonctions get/set et les fonctions utiles pour charger et sauvegarder les fichier. Elle implémente l'interface donnée GameManager. SimpleGameManager et VisualGameManager (managers du jeu) héritent de la classe abstraite mère Game pour récupérer le code de base du game manager. Comme Game implémente GameManager, ces deux classes filles n'ont pas besoin de le réimplémenter. De manière analogue, SimpleGamePlayer implémente GamePlayer directement comme il n'hérite de rien.

0.3 Le package view

Nous avons entièrement modifié ce package car on arrivait pas réellement à voir comment les interfaces proposées se coordonnaient. Nous avons de plus trouvé intéressant l'idée d'apprendre à organiser nous même du code en Java.

Après avoir supprimé les interfaces proposées, nous avons décidé de découper la view en deux frames : MainFrame qui comprend la page d'accueil (Home) et Game qui contient la frame de jeu. MainFrame et Game sont donc des classes filles de JFrame et implémentent l'interface Frame que nous avons réalisé (permet de gérer le focus nécessaire pour la communication jeu/clavier qui se perd lors de certaines actions et les sorties des frames qui sont des actions similaires dans les deux cas). Game implémente aussi ActionListener et KeyListener pour respectivement les interactions (boutons, etc...) et le keyboard. Dans le MainFrame, Home hérite de JPanel et implémente également ActionListener (pour les boutons des options, lancer le jeu, le ranking, etc...) et de Frame (focus, quitter la page).

Pour dessiner la grille et les tétrominos, nous avons choisi de faire un "drawer". Le principe est qu'on a un panel qui a une taille en hauteur et largeur définie. Les cases sont affichées en divisant la taille du jpanel par rapport à la taille de la grille ou du tétromino (dans le cas du hold ou du next). On applique ensuite des couleurs à ces cases en fonction du type (empty, fond, tétromino). A chaque mouvement on repaint la grille. Pour celà, on a une interface qui contient des fonctions communes aux classes du drawer. On a une classe abstraite TetrominoDrawer qui regroupe ce qui est en commun (taille des carrés et coloration). TetrominoDrawer implémente l'interface Drawer et hérite de JPanel. La classe GameDrawer hérite de TetrominoDrawer et sert à peindre la grille de jeu en récupérant les informations du game manager. De manière analogue, Hold et Provider héritent de TetrominoDrawer et servent respectivement à tracer les carrés avec les tétrominos hold et next, comme visible dans la Figure 1. Ils fonctionnent de manière identique GameDrawer mais se basent sur la taille des tétrominos.

La classe MenuBar hérite de JMenuBar et implémente ActionListener. Le principe est le même que pour le menu Home, mais pour un type différent, ici une barre de menu.

La classe ImagePanel implémente JPanel et est utilisée pour gérer l'insertion des images.

Enfin, pour la gestion de l'audio, on a deux classes principales qui sont Music et Sound, qui héritent de Thread, qui permet de lancer des process en parallèle. Elles permettent de charger les fichiers audio et contiennent des fonctions pour activer selon des modes précis (boucles...) les audios.
Ces classes sont utilisées par la classe SoundMenu, qui hérite d'un JPanel et implémente ActionListener. Cette classe permet de gérer l'activation ou la désactivation des audio en fonction des actions sur les boutons. Pour finir, Settings est une classe de communication inter-classes qui permet de partager l'état de l'audio et de le laisser actif ou non lorsqu'on change de frame.

Nous avons utilisé des interfaces uniquement lorsque nous avions besoin de relier des objets à d'autres, pour appliquer des modifications génériques à plusieurs classes. Or dans ce package, beaucoup de fonctions ne

répondaient à ces nécessités, donc contrairement aux autres, nous n'avons pas fait en sorte que toutes les classes n'implémentent pas des interfaces.

Conclusion

Pour étendre notre projet, nous aurions souahité faire un multijoueur ou deux joueurs pourraient jouer sur deux grilles différentes sur le même ordinateur. L'idée aurait été d'afficher deux grilles reliées à des touches différentes pour que chacun puisse jouer sur le même clavier.
Par la suite, il aurait été intéressant de faire une IA qui aurait plusieurs niveau d'intelligence (de débutant à expert par exemple), et permettre au joueur de choisir dans le cadre du multijoueur de jouer contre l'IA (et choisir le niveau de difficulté), ou avec un autre joueur.

L'aspect de création graphique était la plus amusante selon nous. Le fait de construire soit même les classes et de voir le jeu fonctionner, de pouvoir le customiser selon nos envies était motivant.

Concernant les difficultés, nous avons eu plus de mal à entrer dans les parties pré-construites du projet. En effet, comprendre la logique déjà implémentée nous à davantage fait réfléchir à comment est-ce qu'il fallait faire pour faire passer des tests, quitte parfois à coder des fonctions sans réellement comprendre pourquoi elles existaient au départ, plutôt qu'à suivre notre logique à nous.
De plus, nous avions au départ des problèmes basiques liés à une compréhension incomplète des bases de Java.

References

[1] *Droits d'utilisation de la musique.* URL: `https://musescore.com/thepixelpenguin/ruins-theme`.

[2] *Droits d'utilisation des images du wiki du jeu Slime Rancher.* URL: `https://slimerancher-archive.fandom.com/wiki/Slime_Rancher_Wiki:Copyrights`.

[3] *Inspiration du Tétris Wiki.* URL: `https://tetris.wiki/Puyo_Puyo_Tetris_2`.