Tetris

Hugo Favier
Alexandre Dalibard Brun
Mathieu Deschanvres

Décembre 2023

Contents

1 Introduction

Pour ce projet, nous devions réaliser un Tetris légèrement modifié. Le principe est le même : nous avons un plateau de jeu sous forme de grille et des pièces de différentes formes. Avec ces pièces, nous devons couvrir le plus d'espace possible de la grille, sans laisser de trou entre les pièces. Contrairement au Tetris traditionnel, dans lequel lorsqu'une ligne est complète, elle disparaît, ici, aucune pièce ne disparaît. En fait, dans notre version, au début de la partie, un certain nombre de pièces est déjà placé sur le plateau. Le joueur doit ensuite les déplacer pour remplir le plus d'espace. Une fois qu'il considère avoir fini ou qu'il a dépassé le nombre de coups maximal, la partie est terminée et on compte alors les points. Pour cela, on regarde l'aire du plus grand rectangle formé par les pièces qu'il a placées. On compte ensuite le nombre de cases de ce rectangle et on obtient le nombre de points du joueur. Le but du jeu pour le joueur est de faire le plus de points possibles.

Pour réaliser ceci, nous avons implémenté le modèle MVC dans notre jeu. Le modèle est représenté par les classes du package " pièces ", qui vont former les pièces, mais également par la classe PlateauPuzzle qui forme le plateau de jeu. Pour la vue, elle est représentée par toutes les classes du package " gui ", elles vont former les différents composants nécessaires pour une partie du jeu. Enfin, le contrôleur est représenté par la classe Contrôleur. Elle va se charger de créer le modèle et la vue, puis gérer le bon déroulement de la partie. Pour mieux comprendre ceci, voici un diagramme de classe :

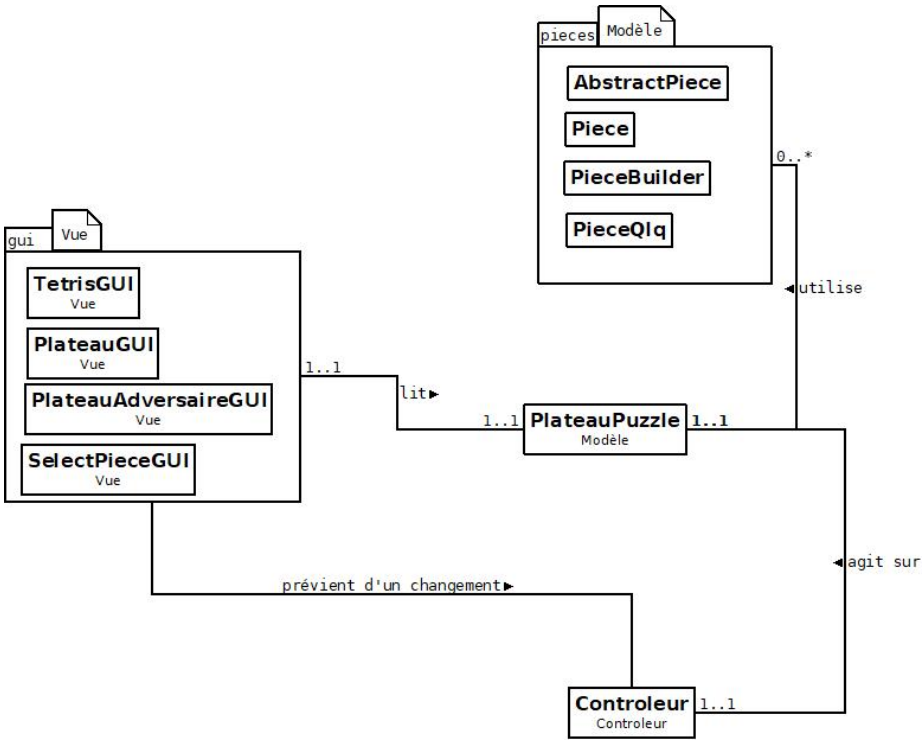

Figure 1: Diagramme de classes

2 Les Pièces

Dans cette partie, nous allons voir le fonctionnement de la partie Piece de notre projet. Vous pouvez voir ci-dessous un diagramme décrivant les liens entre les différentes classes utilisé pour définir et utiliser des pièces (voir figure 2).

2.1 L'interface et abstractPiece

On peut compter trois fonctionnalités importantes dans les pièces. Ces fonctionnalités sont celles qui seront utilisées par le plateau pour déplacer les pièces.

- rotation
- estVide
- setPosition

2.1.1 rotation

cette fonction consiste en la rotation de la grille qui contient la pièce. Celle-ci est faite avec la création d'une nouvelle grille puis l'insertion de la pièce avec

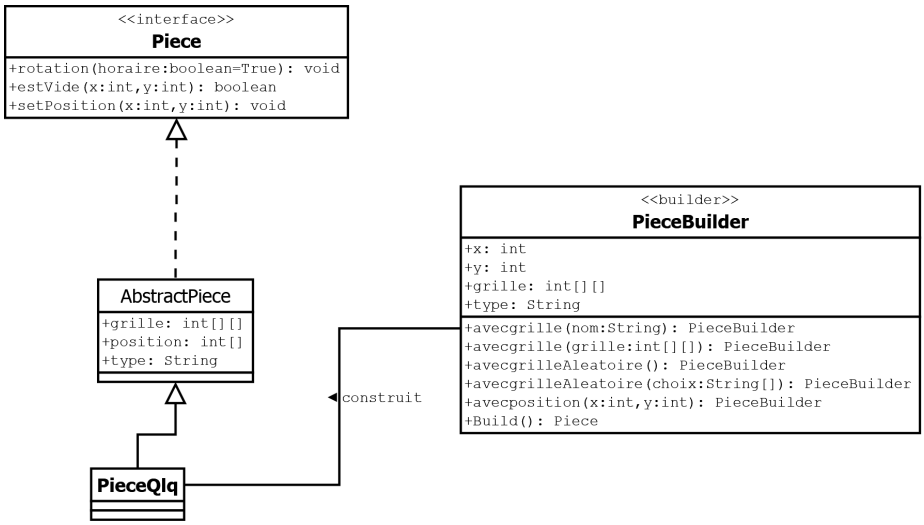

Figure 2: Diagramme des pièces

des boucles et des indices x et y. On peut noter que pour effectuer une rotation avec :

- x
- y
- n, la hauteur de la pièce avant rotation
- m, la largeur de la pièce avant rotation

Pour une rotation horaire : $(x, y) = (y, x - n - 1)$
Et pour une rotation antihoraire : $(x, y) = (y - m - 1, x)$

2.1.2 estVide

Cette fonction permet de voir si une case de la pièce contient une partie de la pièce ou non, cette fonction s'est révélée problématique dans la mesure où les coordonnées insérées en paramètre sont celles de la grille qui contient la pièce et non celle de la grille.

2.1.3 setPosition

cette fonction est très utile pour le plateau, mais ne sert à rien en ce qui concerne la pièce, il s'agit d'un setter tout à fait normal.

2.2 La classe PieceQlq

Cette classe ne fait que construire une pièce avec les paramètres qui lui ont été insérés. Le builder l'utilise lors de la création d'une nouvelle pièce.

Il est possible d'implémenter d'autres classes similaires, mais auxquelles on a ajouté des propriétés particulières, son rôle est donc primordial dans la modularité du code

2.3 Le builder

Le builder PieceBuilder est la Classe qui permet de créer une nouvelle Pièce, chaque méthode excepté build servent à spécifier les pièces, par exemple, on peut récupérer une pièce aléatoirement, une pièce pré-faite (comme T ou L), ou l'ajout de pièces personnalisées.

Attention cependant, l'ajout de pièces personnalisées n'étant pas prioritaire dans le développement, elles ne sont pas compatibles avec toutes les fonctionnalités du projet.Cependant, il s'agit d'une voie possible si le développement devait être repris.

3 Strategy

Pour réaliser notre projet, nous avons utilisé un design pattern comportemental nous servant à définir quelles méthodes l'ia utilisera pour jouer. Pour ce faire, nous avons une classe Context ainsi que les classe Strategy, context est la classe qui va interagir avec le contrôleur. Elle nous permet de choisir une stratégie et de l'exécuter.

Les stratégies ont été ajoutées de manière à respecter le design pattern, il n'y a donc pas de paramètre dans le constructeur et le plateau est précisé au moment de son exécution Pour le moment, nous avons implémenté 2 stratégies, l'empilement et les choix aléatoires :

3.1 L'empilement

L'empilement est une stratégie simpliste qui consiste en l'agglutinement de toutes les pièces dans le coin supérieur gauche en évitant d'utiliser toujours les mêmes pièces.

3.2 les choix àléatoires

Cette stratégie utilise, comme son nom l'indique, des choix aléatoires

4 Le plateau

Le plateau représente le cœur du tetris, ses fonctions sont celles qui contiennent les principales fonctionnalités et règles du jeu.

4.1 PlateauPuzzle

PlateauPuzzle est le centre de tous les types de plateaux, il contient les fonctions qui régule le jeu. Ces fonctions sont :

- checkIntersection
- intersectionRectanglePiece
- checkCaseIntersectionPiece
- checkColision
- taillePlusPetitRect
- plusGrandRect

Notons que les quatre premières ont des noms similaires, cependant bien que toutes servent le même objectif, trouver les collisions entre les pièces, chacune ont leur rôle spécifique. Le deux dernières quant à elles sont celles qui calculent le score

4.1.1 checkIntersection

checkIntersection est la fonction la plus simple, elle vérifie simplement si les grilles de pièces sont superposées, sont n'étant pas de trouver les collisions, mais simplement de trouver où il pourrait y en avoir.

4.1.2 intersectionRectanglePiece

intersectionRectanglePiece s'utilise après checkIntersection, son rôle est de renvoyer l'intersection entre deux pièces.

4.1.3 checkCaseIntersectionPiece

checkCaseIntersectionPiece vérifie si l'intersection contient une collision, elle nécessite l'utilisation de intersectionRectanglePiece au préalable.

4.1.4 checkColision

checkColision est la fonction qui utilise toutes celles vu ci-dessus pour renvoyer si deux pièces sont en collision ou non.

Pour limiter les problèmes qui pourraient être causés par l'inefficacité des fonctions ci-dessus, checkColision cherche à exécuter un minimum de fonctions.

4.1.5 taillePlusPetitRect

cette fonction renvoie l'aire du plus petit rectangle (parallèle aux axes) recouvrant l'ensemble des pièces, cette aire pourra être récupérée pour trouver le gagnant

4.1.6 plusGrandRect

cette fonction renvoie l'aire du plus grand rectangle (parallèle aux axes) dans l'ensemble des pièces, cette aire pourra être récupérée pour trouver le gagnant (la personne ayant réussi la forme la plus compacte gagne) si la fonction précédente n'en trouve pas.

5 La vue

Comme dit précédemment, la vue est représentée par les classes du package gui. Tout d'abord, PlateauAdversaireGUI et PlateauGUI vont représenter les plateaux de jeu d'un joueur humain et d'un joueur contrôlé par l'ordinateur. Ensuite, SelectPieceGUI permet d'afficher la pièce sélectionnée au joueur humain, ce qui aide à avoir une meilleure vision de ses actions au cours de la partie. Enfin, TetrisGUI va assembler toutes les classes précédentes et ajouter d'autres composants nécessaires au jeu, par exemple, le bouton qui termine la partie et affiche le score final. C'est cette classe qui représente réellement la vue de notre jeu.

Pour le joueur humain, il a trois interactions possibles avec la vue. La première est un clic sur son plateau. S'il clique sur une pièce, elle est alors sélectionnée, sinon il ne se passe rien. Une fois une pièce sélectionnée, il peut cliquer à nouveau sur la grille pour déplacer la pièce. La seconde arrive une fois qu'une pièce est sélectionnée. Il peut alors faire un clic molette ou un clic droit pour effectuer une rotation de la pièce. Un clic molette fera tourner la pièce dans le sens antihoraire et un clic droit dans le sens horaire. Enfin, il peut cliquer sur le bouton " terminé ". Cela termine la partie et affiche son score ainsi que s'il a gagné, perdu ou fait une égalité. Voici à quoi cela ressemble :

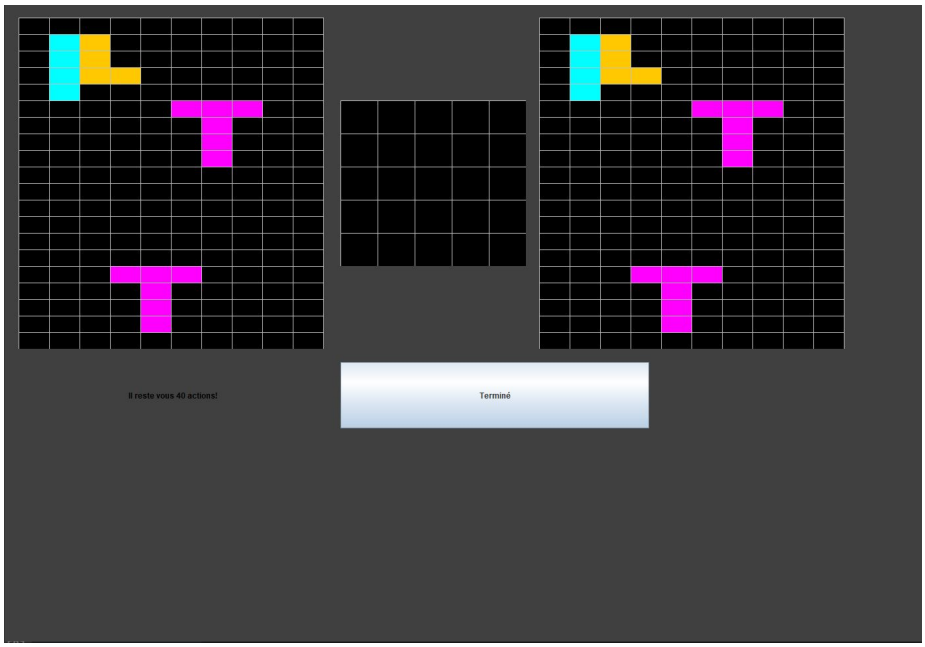

Figure 3: Exemple de partie

6 Le contrôleur

Le contrôleur est la classe Contrôleur. Elle va créer la vue grâce à TetrisGUI et deux modèles : le plateau du joueur et le plateau de l'adversaire avec PlateauPuzzle et Adversaire. C'est TetrisGUI qui reçoit les clics sur les différents composants et qui va prévenir le contrôleur qu'une action est effectuée en appelant une des méthodes de ce dernier. Ensuite, c'est le contrôleur qui va décider de modifier ou non le modèle et la vue en fonction de la situation.

7 Problèmes lors du développement

7.1 Redondance

Malgré la redondance des parcours lors des tests de collisions, nous estimons malgré cela que celles-ci sont indispensables à la compréhension de cette partie du code.

8 Conclusion

Nous avons donc réalisé un jeu de "tetris" répondant à ces modalités

- toutes les pièces sont exposées sur l'aire de jeu dès le début de la partie
- à tout moment, le joueur peut choisir une pièce en clinquant dessus, et a alors la possibilité de la faire tourner ou de la déplacer,
- son but est de minimiser l'espace occupé par l'ensemble des pièces. Plus précisément, la fonction d'évaluation sera l'aire du plus petit rectangle (parallèle aux axes) recouvrant l'ensemble des pièces,
- lorsque le joueur considère avoir terminé (ou lorsque le nombre maximum d'actions autorisées est atteint), il clique sur un bouton et son score est alors calculé,
- Un Ordinateur est disponible