Rapport sur la modélisation du Projet Tetris - DEV4 2024

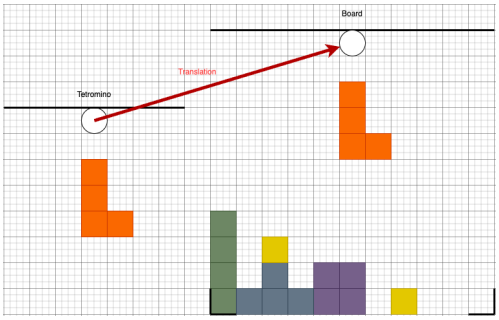
Auteurs: Enzo Renard 60904 - Julien Delcombel 60189

Modèle:

• Classe Tetromino

ATTRIBUTS:

- **int id:** Un identifiant pour représenter la forme spécifique du tetromino
- **Position reference**: La position commune aux tetrominos pour l'insertion et le déplacement sur le board.
- **Position centre:** la position centrale du tetromino permettant une rotation de la pièce
- **List<Position> relativePositions:** La liste des positions qui font la forme du tetromino. Positions relatives au point de référence sur le tableau pour une facilité de déplacement et du suivi de la pièce sur le plateau.



Transition d'une pièce du Bag vers le Board

MÉTHODES:

- **Tetromino():** Constructeurs pour initialiser le tetromino avec sa forme et sa position de départ.
- **rotateClockwise(), rotateCounterClockwise:** Méthode nécessaire à la rotation de la pièce dans le sens des aiguilles d'une montre.
- **move(int dx, int dy):** méthode général d'un mouvement de la pièce. Utilisé par le game lors de l'instruction d'un utilisateur ou la descente automatique.

La classe Tetromino est conçu pour gérer tous les aspects liés au comportement individuel des pièces de Tetris, y compris leur mouvement, leur rotation et leur représentation graphique, tout en fournissant les informations nécessaires à la classe Board pour qu'elle puisse gérer les règles du jeu et les interactions entre les différentes pièces.

• Classe Board

ATTRIBUTS:

- **Tetrimino[] tetrominos:** Un tableau de tetrominos sur le plateau de jeu. On retrouve avec ce tableau, les tetrominos posés le plateau ainsi que le tetromino actif. Nous retrouvons ainsi les informations des pièces présentes sur le plateau.
- **Position reference**: La position pour l'insertion et le déplacement des tetrominos
- **int height**, **int width**: Définissent les dimensions du plateau
- **boolean[][] occupied**: Afin de gérer les collisions et pour avoir un accès direct à une position sur le plateau, nous mettons à jour un tableau 2D de boolean pour savoir si une case et occupé ou non par une pièce. Facilite donc la gestion des collision ainsi que des lignes complètes.

MÉTHODES:

- **Grid():** Initialise la grille, préparant le plateau pour le jeu
- **bool moveActive(Direction direction)**, **bool rotateActive()**: Les méthodes qui gèrent le mouvement de la pièce courante sur le plateau et sa validité afin de modifier les positions et maintenir à jour le tableau de boolean.
- **bool isOutside(int row, int col)**, **bool isFree(int row, int col)**, **bool isLineComplete()**, **void clearLine(int row)**, **void moveLineDown(int row)**, **int removeCompletedLines():** Fournissent

la logique nécessaire pour gérer le plateau, y compris la vérification des positions, la suppression des lignes et le déplacement des lignes.
- **void addBrick(Tetromino tetromino):** Ajoute un tetromino sur le plateau de jeu
- **bool isGameOver()**: vérifie si l'état du plateau implique une fin de la partie

• Classe Bag

ATTRIBUTS:
- Bag instance
- List<Tetromino> possibleTetrominos: La liste de tout les tetrominos qui peuvent être inséré dans le sac. Les 7 par défaut ou selon les choix de l'utilisateur
- List<Tetromino> tetrominos: La liste des tetrominos présent dans le bag qui peuvent être pioché du sac et inséré sur le board

MÉTHODES:
- **Bag getInstance()**:
- **Tetromino getNext()**: le prochain tetromino dans le sac qui ira sur le board
- **void shuffle()**: Permet de mélanger l'ordre des tetrominos dans le sac

La classe Bag est chargé de fournir les pièces du jeu au plateau tout en alternant l'ordre de sortie toutes les n pièces.

• Classe Game

ATTRIBUTS:
- **Bag bag**, **Board board**: Composants principaux du jeu, gérant les pièces et le plateau
- i**nt score**, **bool gameOver**, **int level**: suivent le score, l'état du jeu et le niveau. Elements pour l'expérience utilisateur.

MÉTHODES:
- **Game(Bag bag, Board board, int score = 0, bool gameOver = false, int level = 1)**: initialise une nouvelle partie avec les composants nécessaires
- **void moveActive(Direction direction)**, **void rotateActive(), void dropActive()** : Interface pour contrôler le tetromino actif sur le board
- **addPoints(in point:int)**, **int increaseLevel()**: Gère le score et le niveau de difficulté de la partie
- **getPoints(int numberOfLines, int droppedLineDuringMove):** Renvoie le nombre de points obtenu par le nombre de ligne supprimée et du nombre de ligne descendu lors d'un drop.
- **play():** La boucle principale d'une partie

Notre classe centrale de l'application qui est chargé de mettre en relation les différents éléments du Tetris, de gérer le déroulement d'une partie, de fournir les informations nécessaires à l'affichage et de valider et effectuer les actions de l'utilisateur.

Vue:

DESIGN PATTERN OBSERVER

Observateurs: **GameView**, **ScoreView** et **BagView**. Ces classes sont responsable de l'affichage du jeu pour l'utilisateur.

Observé: **Game**. Etant la classe principale de notre jeu et possédant les informations d'une partie en cours, c'est cette classe qui est observé et notifie la vue des changements via les classes technique **Observer** et **Observable**.

Ce modèle nous permet de séparer la logique du jeu (**Game**) de sa présentation (**GameView**, **BagView**, **ScoreView**). Lorsque l'état du jeu change (par exemple, score mis à jour, changement de pièces dans le sac, …), **Game** informe automatiquement les vues. Cela permet une synchronisation en temps réel entre la logique du jeu et l'interface utilisateur. Les vues n'ont pas besoin de connaître les détails internes de **Game**. Elles réagissent seulement aux notifications qu'elles reçoivent. Cela réduit les couplages entre les différentes parties de notre application.

Controller:

DESIGN PATTERN COMMAND:

Toujours dans une logique de séparer l'interface utilisateur de la logique de notre application, nous mettons en place le pattern Command où les actions spécifiques du jeu (comme la rotation ou le mouvement des pièces) sont encapsulées dans des objets de commande distincts comme **RotateClockwiseCommand, RotateCounterClockwiseCommand, MoveRightCommand, MoveLeftCommand, MoveDownCommand, DropCommand** stocké par **Invoker. GameController** agit comme un pont entre l'interface utilisateur et les actions à exécuter par **Game**.