



Projet DEV4 – C++

Rapport d'analyse de Tetris en C++

Auteurs : Angielczyk Marcel 60453 & Derboven Téo 60073

Introduction :

Le présent rapport est destiné à fournir une analyse approfondie des classes utilisées dans notre programme de jeu Tetris, développé en langage C++. L'objectif de cette analyse est de présenter une vision claire de la structure du modèle du jeu et d'identifier les classes qui jouent un rôle essentiel dans son fonctionnement.

L'analyse au format starUML est disponible dans un fichier à part.

Classes :

Ci-dessous, une liste exhaustive des classes du modèle avec, pour chaque classe, la liste, le plus souvent exhaustive, de ses attributs ainsi qu'une liste de ses principales méthodes.

Block

Les blocs sont la base du jeu. Ils forment les Tétrominos et les lignes de la grille.

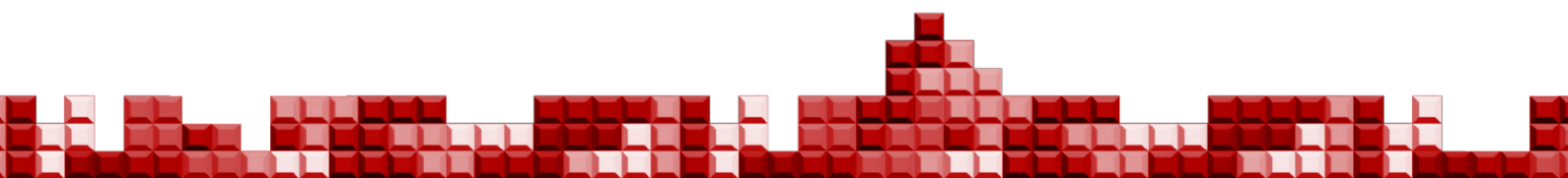
```
/** Attributs **/  
  
color: Color // la couleur du bloc (sera utilisé dans la vue)
```

Tetromino

Un tétromino est une pièce du jeu. Formés par des blocs, ils peuvent prendre toutes sortes de formes.

```
/** Attributs **/  
  
shape: Block[][] // blocs qui forment le tétromino
```

```
/** Méthodes principales **/  
  
get(x: int, y: int): Block // renvoie le bloc à la position donnée  
isOccupied(x: int, y: int): boolean // détermine si une position est occupée par un bloc. Permet un accès plus facile pour les conditions  
rotate(direction: RotateDirection) // rote le tétromino dans le sens donné
```





Line

Une ligne fait partie de la grille de jeu. Elle peut être occupée par des blocs. Elle propose une méthode `isFull()` qui détermine si la ligne est pleine de blocs, faire des lignes de blocs étant un des buts majeurs du jeu tetris.

```
/** Attributs */

content: Block[] // blocs qui forment la ligne
```

```
/** Méthodes principales */

get(position: int): Block // renvoie le bloc à la position donnée
isFull(): boolean // détermine si la ligne est remplie de blocs
clear() // vide la ligne de ses blocs
```

Grid

C'est la grille de jeu. Elle est composée de lignes pour les blocs qui ne font plus partie d'un Tétromino et d'un Tétromino courant. Ce dernier est celui qui est occupé à tomber et que le joueur peut contrôler.

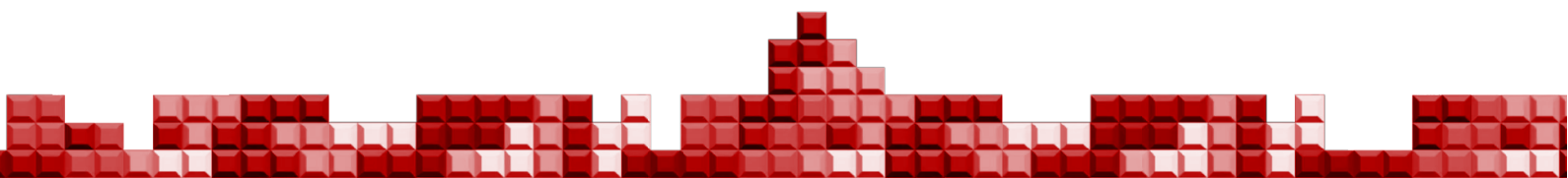
```
/** Attributs */

grid: Line *[] // lignes qui composent la grille
current: Tetromino // Le tétromino actuellement occupé à tomber
currentCol: int // la colonne du tétromino courant, relatif à son haut gauche
currentRow: int // la ligne du tétromino courant, relatif à son haut gauche
```

Pour l'attribut grid, nous pensons créer une série d'objets dynamiques Line. L'attribut grid serait donc un tableau de pointeurs de Line. Ainsi, quand une ligne est remplie et qu'il faut la supprimer, il suffit juste de décaler tous les pointeurs qui précèdent celui qui représente la ligne pleine et d'insérer la ligne qui aura été vidée au début du tableau. Il est moins coûteux de déplacer un pointeur plutôt qu'un objet Line, ou encore pire, bloc par bloc.

```
/** Méthodes principales */

get(row: int, col: int): Block // renvoie le bloc à la position donnée
insert(tetromino: Tetromino) // insère un tétromino au centre haut de la grille
getFullLines(): List<int> // donne une liste de toutes les lignes remplies
removeLine(line: int) // supprime la ligne en position donnée (si elle est pleine)
moveCurrent(direction: MoveDirection) // déplace le tétromino courant horizontalement ou vers le bas
rotateCurrent(direction: RotateDirection) // tourne le tétromino de + ou - 90°
dropCurrent() // place le tétromino là où il y a de la place en le laissant tomber
```





Bag

Le sac de jeu contient les pièces de tétramino de la partie.

```
/** Attributs **/
```

```
bag: list<Tetromino> // tétramino jouables dans la partie  
index: int // indice du tétramino actuel dans la liste
```

```
/** Méthodes principales **/
```

```
add(tetromino: Tetromino) // ajoute un tétramino dans la liste  
getCurrent(): Tetromino // renvoie le tétramino à jouer. L'indice est ensuite  
incrémenté sans sortir de la liste  
getNext(): Tetromino // renvoie le tétramino qui suivra (pour permettre  
d'anticiper un coup)
```

Game

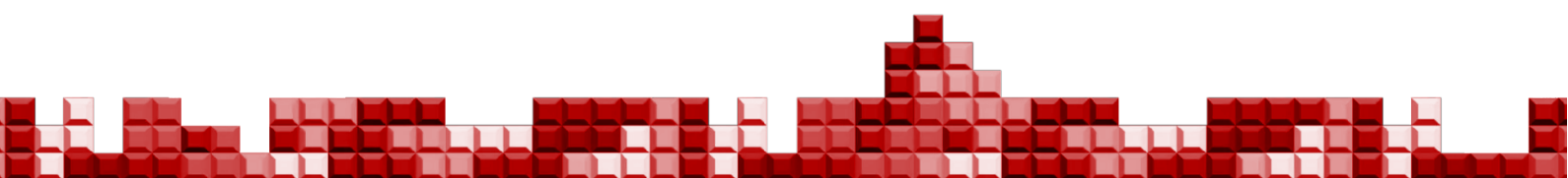
C'est le point d'entrée du modèle. La classe contient le sac et la grille de jeu. Elle permet d'exécuter toutes les actions que l'on peut faire sur le jeu Tetris.

```
/** Attributs **/
```

```
score: long long int // le score actuel du joueur  
grid: Grid // la grille de jeu  
bag: Bag // le sac de jeu  
level: int // le niveau du jeu
```

```
/** Méthodes principales **/
```

```
getGridView(): GridView // renvoie une vue de la grille actuelle  
isGameOver(): boolean // permet de savoir si la partie est terminée  
isWon(): boolean // permet de savoir si la partie a été gagnée. Méthode à  
implémenter dans une sous-classe  
  
goDown() // fait descendre le tétramino actuel d'une ligne  
goLeft() // déplace le tétramino actuel d'une case vers la gauche  
goRight() // déplace le tétramino actuel d'une case vers la droite  
rotateLeft() // tourne le tétramino actuel de -90°  
rotateRight() // tourne le tétramino actuel de +90°  
drop() // laisse tomber le tétramino actuel
```





GameTypeLines

Représente un type de jeu particulier : le jeu se termine après qu'un certain nombre de lignes soit réalisées. Hérite de la classe Game.

```
/** Attributs */  
  
linesToReach: int // les lignes à atteindre  
nbLinesClear: int // le nombre de lignes déjà réalisées
```

GameTypeScore

Représente un type de jeu particulier : le jeu se termine après que le score spécifié soit atteint. Hérite de la classe Game.

```
/** Attributs */  
  
scoreToReach: int // le score à atteindre
```

GameTypeTime

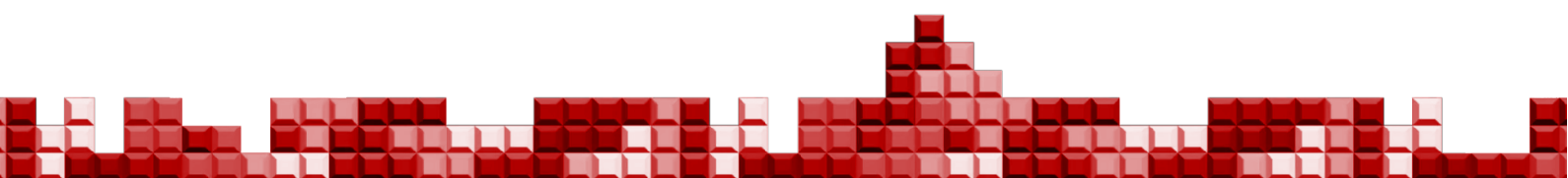
Représente un type de jeu particulier : le jeu se termine une fois le temps spécifié écoulé. Hérite de la classe Game.

```
/** Attributs */  
  
startTime: Time // l'heure de début de la partie
```

GridView

Représente la grille de jeu comprenant le tétramino qui descend actuellement, cette classe ne sera utilisée que pour l'affichage.

```
/** Attributs */  
  
grid: Line[]
```





Énumérations

Color

Définit les différentes couleurs possibles d'un tétramino

```
{BLUE, GREEN, YELLOW, ORANGE, RED}
```

RotateDirection

Définit les deux possibilités de rotation lorsque le tétramino tombe

```
{LEFT, RIGHT}
```

MoveDirection

Définit les possibilités de déplacement lorsque le tétramino tombe

```
{LEFT, RIGHT, DOWN}
```

