# Tetris

| Andrew | Constantin | Gregory | Thomas |
|---|---|---|---|
| 54327 | 54439 | 54786 | 54341 |

# Introduction

Nous avons implémenté une version à deux joueurs du jeu Tetris. Pour ce faire nous nous sommes basés sur un modèle simplifié du jeu de base, dans lequel nous avons ajouté un système de **client/serveur** qui héberge les parties en cours ainsi qu'une base de données stockant différentes informations telle le nombre de lignes supprimées ou meilleurs scores du joueur. Le résultat est jeu ou 2 joueurs peuvent s'affronter dans une partie de Tetris.

# Description du jeu


## Généralité

*Tetris* est principalement composé d'un champ de jeu où des pièces de formes différentes, appelées « tétrimino », descendent du haut de l'écran. Durant cette descente le joueur peut uniquement déplacer les pièces latéralement et leur faire effectuer une rotation sur elles-mêmes jusqu'à ce qu'elles touchent le bas du champ de jeu ou une pièce déjà placée. Le joueur ne peut ni ralentir la chute des pièces, ni l'empêcher, mais il peut dans certaines versions l'accélérer.

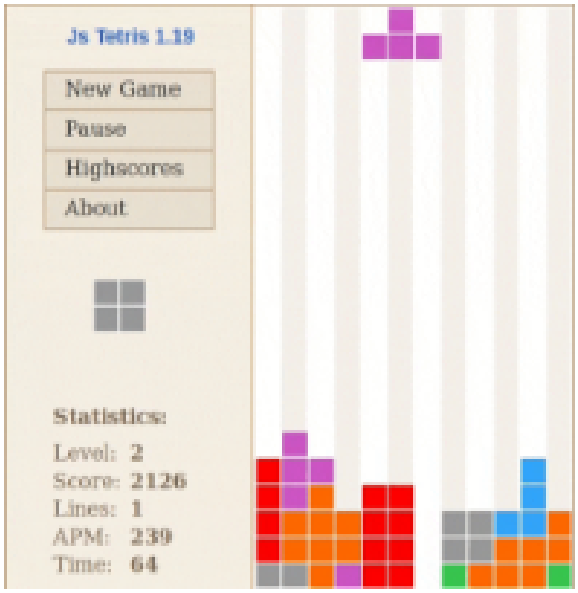



Illustration d'une pièce apparaissant


Illustration de fin de partie

Le but pour le joueur est de réaliser le plus de lignes possibles, afin de garder de l'espace pour placer les futures pièces. Une fois une ligne complétée, elle disparaît, et les blocs placés au-dessus chutent d'un rang. Si le joueur ne parvient pas à faire disparaître les lignes suffisamment vite, l'écran peut alors se remplir jusqu'en haut. Lorsqu'un tétrimino dépasse du champ de jeu, et empêche l'arrivée de tétrimino supplémentaires, la partie se termine. Le joueur obtient un score, qui dépend essentiellement du nombre de lignes réalisées lors de la partie.

Le jeu ne se termine donc jamais par la victoire du joueur. Avant de perdre, le joueur doit tenter de compléter un maximum de lignes. Pour ce faire, il peut éliminer plusieurs lignes d'un coup, ce qui lui rapporte des points de bonus dans la plupart des versions du jeu. Il est possible de compléter jusqu'à 4 lignes en même temps, grâce au tétrimino en I : ce coup est appelé un *Tetris*, comme le nom du jeu.

Au fur et à mesure que le joueur complète des lignes, son niveau augmente, accroissant le nombre de points réalisés par ligne complétée. Dans la plupart des versions sur jeu, la vitesse de chute des pièces augmente à chaque prise de niveau, laissant moins de temps au joueur pour réfléchir au placement. Le principal objectif

pour le joueur consiste alors non pas à gagner, comme dans la plupart des jeux vidéo, mais à obtenir le plus grand score, via des combinaisons et de l’endurance. (Wikipédia)

## Pièces


Les pièces de *Tetris*, sur lesquelles repose entièrement le jeu, sont des « tétriminos ». Il en existe sept formes différentes, toutes basées sur un assemblage de quatre carrés, d'où le mot « Tetris » du préfixe grec *tetra-* qui signifie « quatre ». Le joueur peut faire tourner plusieurs fois, à gauche et/ou à droite selon la version, de 90° n'importe quel bloc pour le poser de la façon désirée pendant qu'il descend. (Wikipédia)

## Champ de jeu

Le champ de jeu, ou « matrice », est l'espace dans lequel tombent les tétriminos. Il dispose toujours d'une grille en arrière-plan, visible ou non, dont les cases sont de la même grandeur que les carrés des tétrimino, et que ceux-ci suivent dans leur chute. Ce champ de jeu est de dix cases de largeur et de vingt-deux cases de hauteur selon les *Tetris Guidelines*. (Wikipédia)

## Chute des tétrimino

Dans le champ de jeu, les tétrimino chutent à partir du haut, avec une vitesse déterminée par le niveau de la partie. Plus le niveau est élevé, plus les pièces tombent vite, obligeant le joueur à placer de plus en plus vite les tétrimino. La plupart des versions accordent un bonus de points lorsque le joueur accélère la vitesse de la chute d'un bloc (appelé *soft drop*). Ce bonus est habituellement proportionnel au temps d'appui. De la même façon, le mouvement *hard drop permet* de poser directement le tétrimino dans la colonne et la position dans laquelle elle est lors de la chute. Cette action rapporte également un bonus de points dans la plupart des versions. (Wikipédia)

La formule utilisé pour calculer la vitesse de descente de Tetrimino est la suivante

$$(0.8 - ((level - 1) * 0.007))^{level-1}$$

## Informations supplémentaires

Au-delà du champ de jeu, la majorité des versions de *Tetris* disposent de plusieurs informations contextuelles. L'élément le plus courant est l'affichage du nombre de lignes complétées depuis le début du jeu, le score (en fonction du nombre de lignes complétées et la vitesse de placement du joueur) et le niveau courant qui détermine la vitesse de chute des pièces.

Ce cadre d'information peut également indiquer le tétrimino suivant, permettant au joueur d'anticiper le coup suivant. Les versions récentes du jeu intègrent une réserve dans laquelle une pièce peut être stockée temporairement, affichée à côté du champ de jeu. (Wikipédia)

| Action | Score | Description |
|---|---|---|
| Single | 100 x level | 1 ligne détruite |
| Double | 300 x level | 2 lignes détruites |
| Triple | 500 x level | 3 lignes détruites |
| Tetris | 800 x level | 4 lignes détruites |
| Soft Drop | 1 x n | Soft drop de n lignes |
| Hard Drop | 2 x n | Hard drop de n lignes |

## Mode multijoueur

Certaines versions intègrent des modes de jeu à plusieurs. Tout comme dans la version solo, le but est de contenir les pièces dans le champ de jeu le plus longtemps possible. Le gagnant peut être déterminé selon plusieurs critères : nombres de points, de lignes réalisées, etc. Sur Game Boy, par exemple, le vainqueur est le premier à réaliser 30 lignes ou à faire perdre l'adversaire.

Les adversaires peuvent également voir l'avancée de l'un par rapport à l'autre. Sur consoles de salon, où tous les concurrents jouent sur un même écran, il est possible de voir le jeu complet de chaque joueur et donc de prévoir quand un adversaire s'apprête à faire un *Tetris*

## Actions

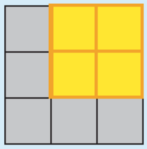
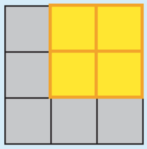
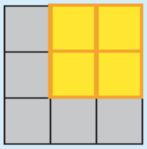
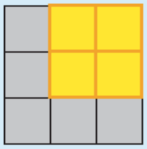
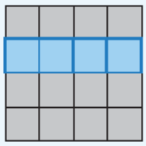
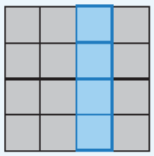
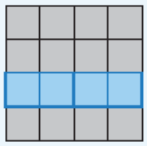
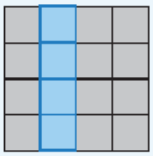
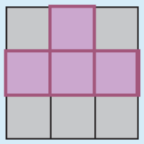
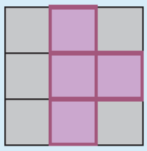
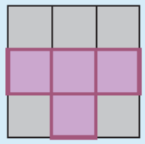
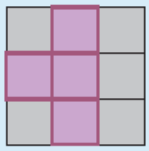
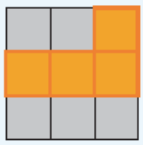
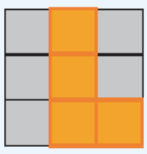
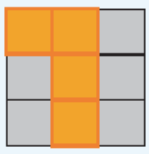
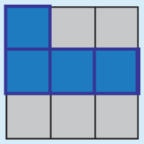
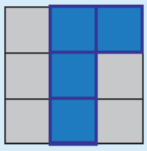
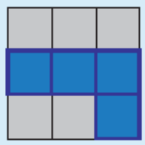
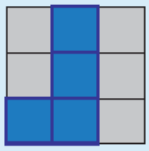
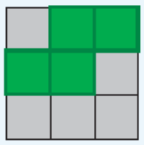
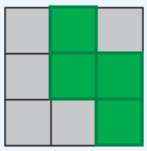
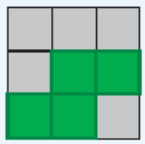
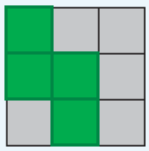
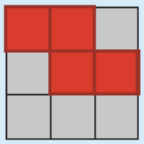
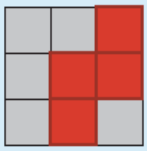
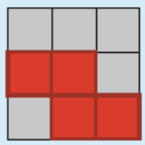
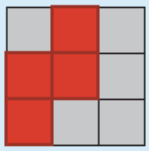
Voice les différentes actions qui peuvent être réaliser dans Tetris.

▼ **Déplacement :**

- **RIGHT** : translate le tétrimino vers la droite.
- **LEFT** : translate le tétrimino vers la gauche.
- **SOFTDROP** : accéléré la chute du tétrimino.
- **HARDROP** : pose directement le tétrimino dans la colonne et la position dans laquelle elle est lors de la chute.

▼ **ROTATE** :

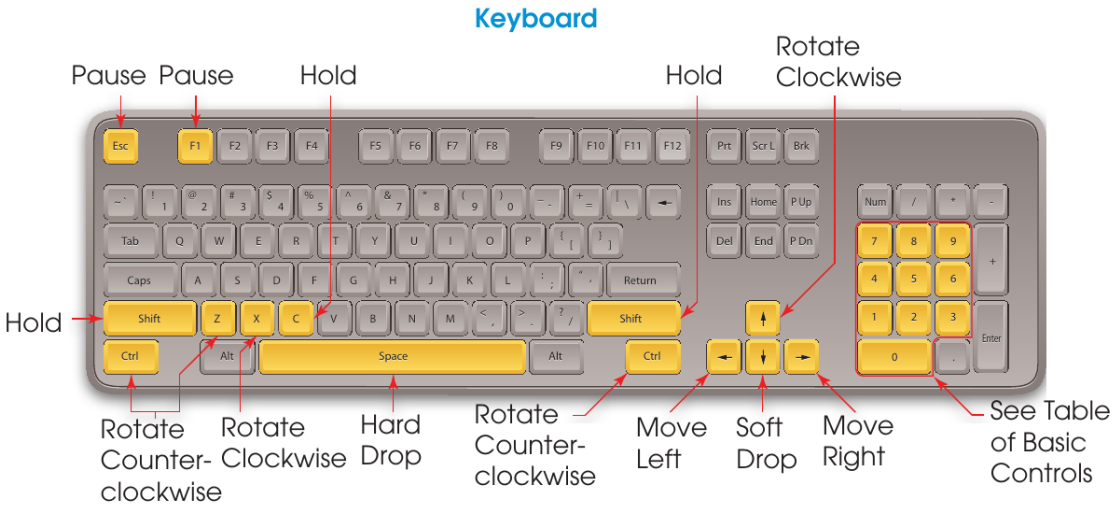
Fait tourner le tétriminos dans le sens horaire ou antihoraire.

| Tetrimino | North | East | South | West |
|---|---|---|---|---|
| O-Tetrimino |  |  |  |  |
| I-Tetrimino |  |  |  |  |
| T-Tetrimino |  |  |  |  |
| L-Tetrimino |  |  |  |  |
| J-Tetrimino |  |  |  |  |
| S-Tetrimino |  |  |  |  |
| Z-Tetrimino |  |  |  |  |

▼ **HOLD** :

Permet de garder la pièce tombante en réserver. Cette action peut être réalisée une seule fois par chute c'est à dire que lorsqu'on a mis de côté une pièce il faut au moins placer une pièce pour à nouveau pouvoir utiliser cette action. Si une pièce est déjà en réserve, cette action va switcher la pièce tombante avec celle en réserve.

## Bouton de contrôle




Voici les inputs qui sont acceptés par le jeu lier à l'action qu'ils réalisent. Ils sont pris en compte uniquement durant une partie de jeu.

- Déplacement « **LEFT** » : Flèche de gauche, 4 pavée numérique
- Déplacement « **RIGHT** » : Flèche de droite, 6 pavée numérique
- Déplacement « **SOFTDROP** » : Flèche du bas, 2 pavée numérique
- Déplacement « **HARDROP** » : Espace, 8 pavée numérique
- **ROTATE** « horaire » : Flèche du haut, X, 3 et 5 pavée numérique
- **ROTATE** « antihoraire » : CTRL, 1 et 9 pavée numérique
- **HOLD** : SHIFT, C, 0 pavée numérique

# Compilation et execution

## Compiler avec Maven

```
$ mvn clean package
```

## Executer les tests unitaires

```
$ mvn verify
```

## Exécuter le serveur

```
$ java -jar src/server/target/tetris.server-2.0.jar
```

Le port par défault est le `6969`

## Exécuter le client

```
$ java -jar src/client/target/tetris.client-2.0.jar
```

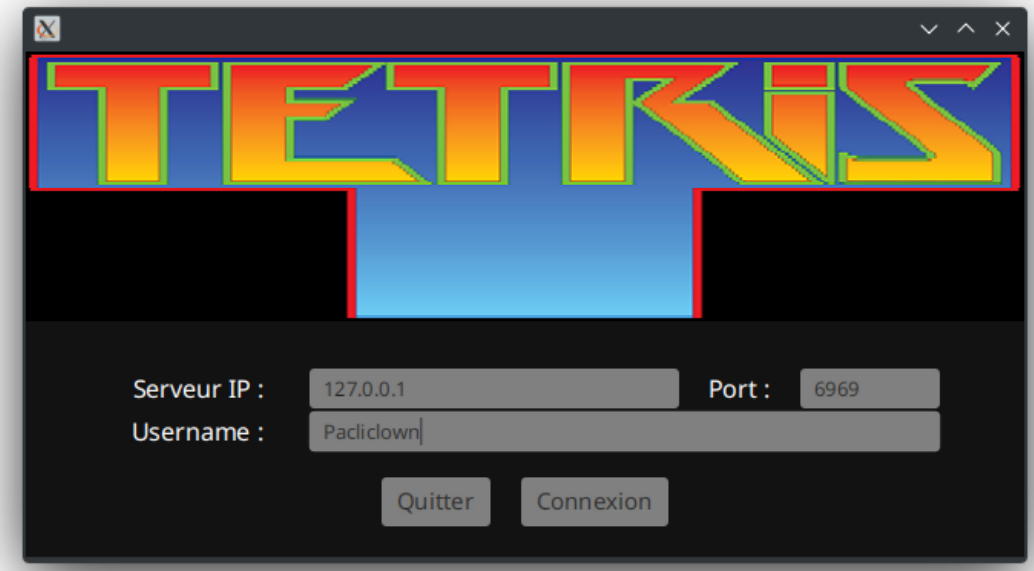
### Avec des parametres

```
$ java -jar src/client/target/tetris.client-2.0.jar --host=127.0.0.1 --port=6969 --use
rname=Pacliclown
```

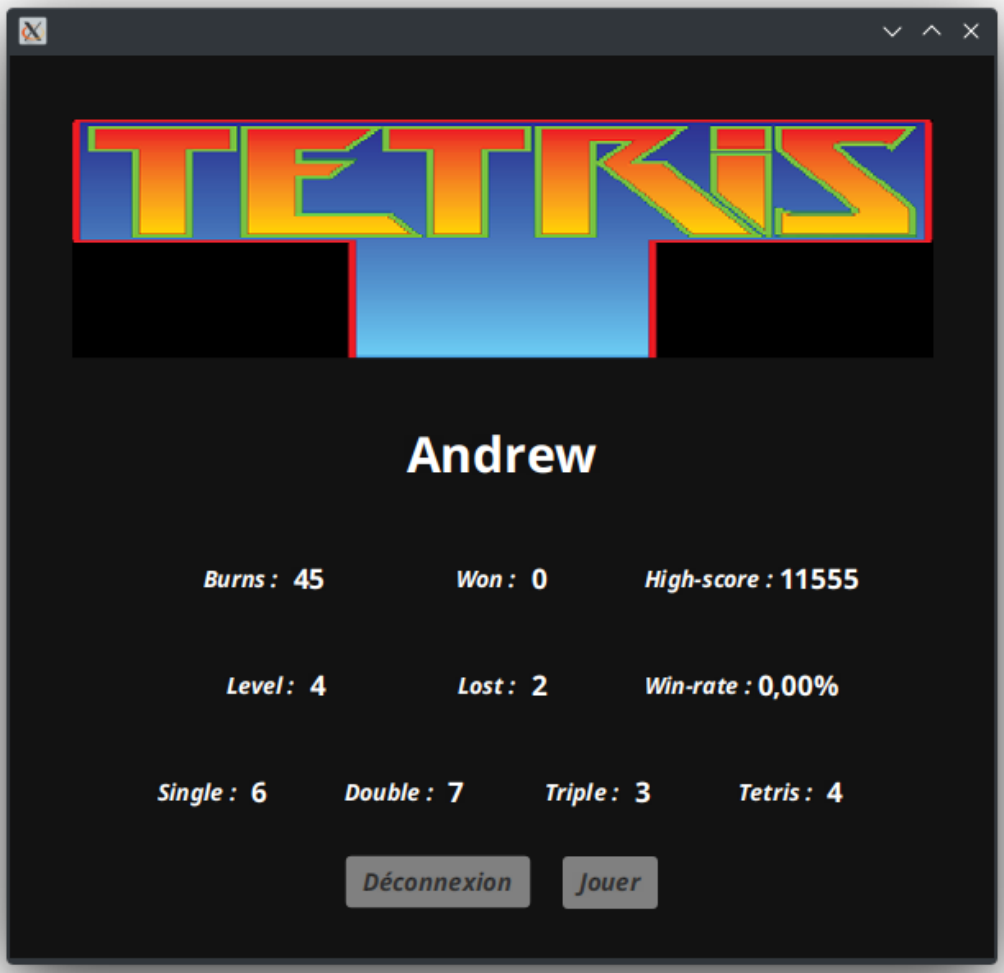
# Implémentations

## Affichage (View)

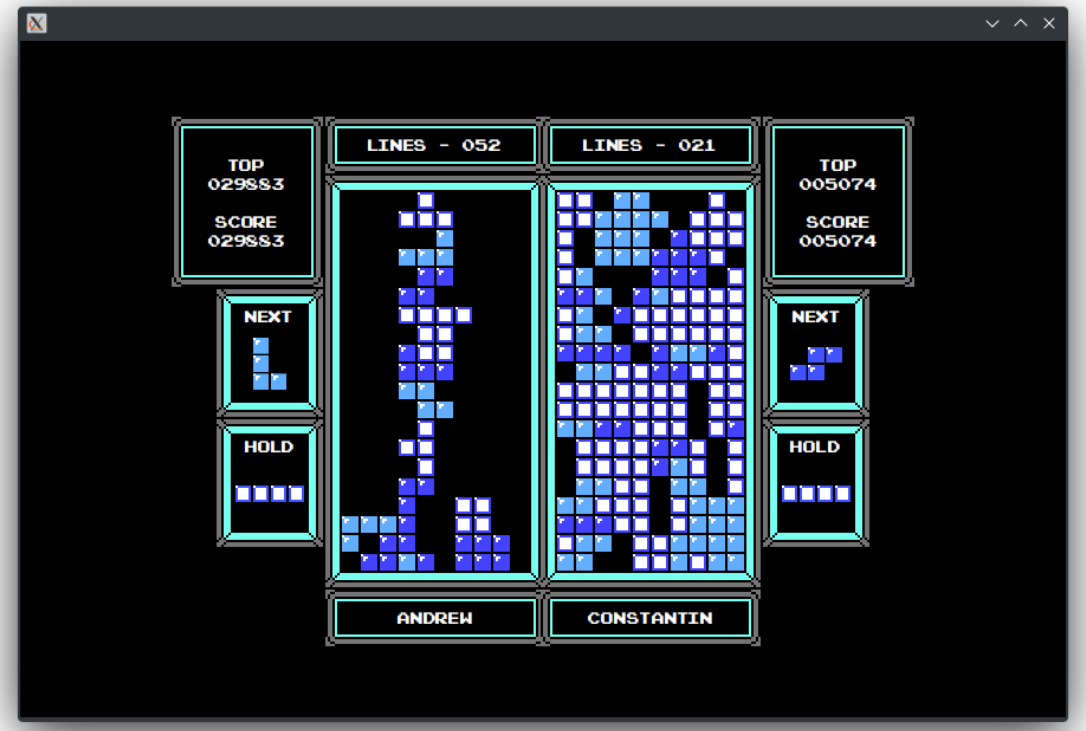
### *Écran de connexion*




***Ecran du Menu***


TETRIS
Andrew
Burns : 45
Won : 0
High-score : 11555
Level : 4
Lost : 2
Win-rate : 0,00%
Single : 6
Double : 7
Triple : 3
Tetris : 4
Déconnexion
Jouer

***Ecran de jeu***


TOP
029883
SCORE
029883
LINES - 052
LINES - 021
TOP
005074
SCORE
005074
NEXT
HOLD
NEXT
HOLD
ANDREW
CONSTANTIN

💡 **Description des classes :**

Le module « *client* » regroupe tout ce qui est en rapport avec l'application côté client. C'est principalement toutes les classes de vue ainsi que les contrôleurs associés. Il communique avec le serveur par le biais de « *message* » qui mettent à jours les informations de chaque côté. Tous cela est diviser en différents package.

- « model » : définit les actions possibles du côté client ainsi que la gestion des communications avec le serveur. Les mise à jour du modèle sont envoyées à la vue via l'utilisation des PropertyChangeListener.
- « controller\Controller » : controller principale du client
- « view\ViewInterface » : définit les méthodes qui seront rendues visible au classes instanciant la vue.
- « view\View » : La classe principale de la vue. C'est la fenêtre principale dans laquelle les scènes sont affichées.
- « View\component\MinoView » : définit l'aspect visuelle des Minos
- « View\controllers » : contient les contrôleurs des différentes scènes ainsi que les controllers spécifique aux éléments de chaque scène

# Server

Dès qu'un client se connecte au serveur, celui-ci reçois ses statistiques. Ensuite, il a l'option de clicker sur le boutton jouer. Il sera delors placé dans une file d'attente. Dès que 2 joueurs sont près, un ***MatchUpHandler*** est créer dans une thread apart, avec les 2 joueurs en question.

▼ **MatchUpHandler**

Le MatchUpHandler va géré toute les communications entre les 2 joueurs**.** Il contient un bag generator, et un mutliplayer game.

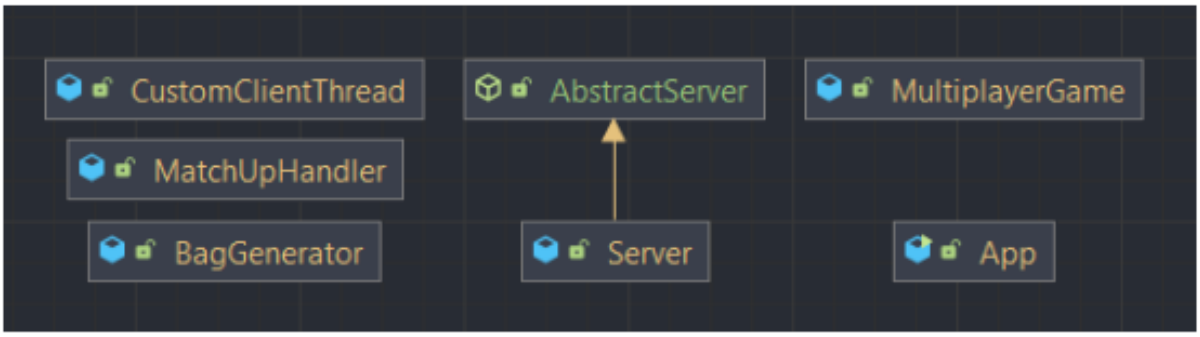
▼ **Bag generator**

Gère la distribution de tout les minos vers les 2 clients.

▼ **Mutliplayer Game**

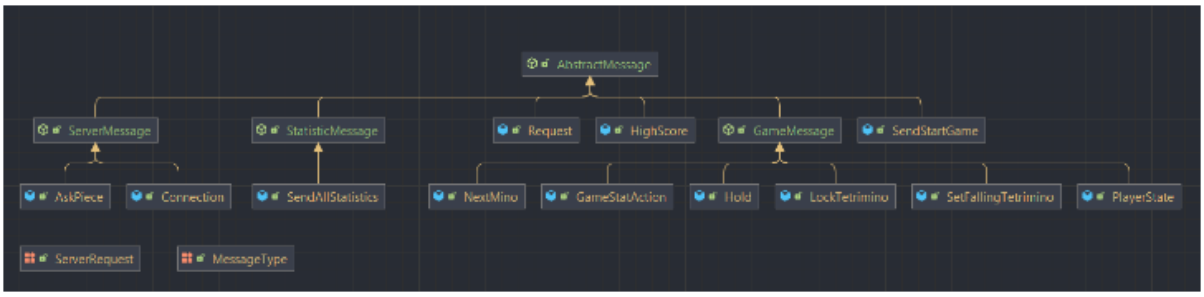
Enregistre l'état actuel constant des 2 jeux qui sont entrain de se déroulé.

Tout les messages reçus par un client ne sont pas d'office transmis vers le MatchUpHandler. Certain sont directement géré par le client. Ceux qui sont manipuler par le MatchUpHandler se voient vois renvoyé a toutes les autres membres du match up et au multiplayer game.




Dès qu'un joueur quitte une partie ses statistiques lui sont directe réenvoyé pour les mettre à jour dans sa fenêtre de statistique.
S'il quitte avant que l'autre joueur termine de sa partie, la seul statistiques non mis-à jour est le fait qu'il gagne/perde. Dès que le deuxième joueur quitte, la statistique de gagné/perde se mets à jour chez le premier client.

## Message




Ce module définit tous les messages qu'un client peut envoyer au serveur et inversement.

▼ **ServerMessage**

Message ayant pour seul but de transmettre une information par rapport au joueur vers le seveur.

▼ **StatisticMessage**

Message qui va servir à transmettre les statistiques à chaque joueur.

▼ **GameMessage**

Message qu'a pour but de modifier le model des chaque joueurs via une method « ***execute »*** qui va prendre en paramètre un Game. Il va modifier le model en conséquence a quel message il est.

# Model

Regroupe l'ensemble des classes qui définissent les mécaniques de jeu Tetris. On y définit les formes ainsi que les comportements qu'elles peuvent avoir au sein de la matrice.


▼ **player**

Package reprenant la logique des actions et du fonctionnement d'une partie à un joueur.

- Un ManagedPlayer est un joueur qui peut fonctionner indépendamment. Un timer est utilisé pour faire descendre les pieces, et des actions

précises sont possibles.

- Un UnmanagedPlayer est un joueur qui n'est pas autonome. Pour se mettre a jour il a besoin d'un message qui le mets a jour. Ceci est utilisé pour afficher l'adversaire et pour enregistrer l'état de la partie dans le serveur

▼ **shape**

Groupe d’interface reprennent toutes les formes de chaque tétrimino suivant leurs orientations.

▼ **tetrimino**

Logique de base d’un tétrimino se trouve dans la classe abstraite « Tetrimino ».

# Base de données

Pour la base de données, on a implémenté un structure a *3 layers.* La logique, les entité et la l'intéraction avec la database.

▼ Business Logic

Une interface qui va permettre de communiquer avec la base de données.Chaque table a sa logique, ceux-ci sont tous regroupé dans ***BusinessModel***.

▼ Database

C'est ici que s'éffectue la communication directe avec la database. Une partie permet de géré la connection et les transaction. Ensuite, toutes les tables ont une classe. C'est la que les requêtes ***sql*** vont être éxécuté.

▼ Database Object (dto)

Chaque table a une entité pour permettre d'écrire une entré de la tables sous forme d'orienté objet.

▼ **Exceptions**

Permet de lever des exception propre a chaque layer.


Dans ressources, on peut trouvé les scripts pour créer les tables, les populé et les drop.


# Bibliothèques utilisées

Nous avons utilisé plusieurs bibliothèques :

- JavaFX 16
- sqlite-jdbc

# Charge de travail

- Groupe Global :
  - Choix du projet de « Tetris »

    - Reflection sur la manière d'implémenter un jeu basé sur le temps sans souffrir d'un lag du serveur
    - Réalisation du rapport.
- Gregory (54786) :
    - Réalisation complète du serveur.
    - Réalisation complète de la base de données.
    - Réalisation partielle de la communication entre le client et le serveur.
    - Réalisation partielle des différents messages.
    - Réalisation partielle de la communication entre le client et le model.
- Constantin (54439) :
    - Réalisation du prototype du v1 des « PropertyChange ».
    - Correction de warnings.
    - Refactoring de noms de classes & variables.
- Andrew (54327) :
    - Réalisation partielle de la communication entre le client et le model.
    - Réalisation partielle de la communication entre le client et le serveur.
    - Réalisation partielle des différents messages.
    - Réalisation complète de la 2ième version de la fenêtre du jeu (view).
    - Réalisation complète du model et de la logique.
    - Gestion du git et des conflits de merges.
- Thomas (54341) :
    - Réalisation partielle de la communication entre le client et le model.
    - Réalisation complètes de la 1ière version de la fenêtre de jeux.
    - Réalisation complètes du menu et de l’écran de connexion.
    - Réalisation minime du model (Setter) pour le « PropertyChange.