# Rapport - Arkanoid

David de Meester de Ravestein

August 6, 2025

**Introduction**

Rapport dans le cadre du projet du cours INFO-F202. Reprend les informations essentielles liées au développement du jeu Arkanoid.

## Table des matières

# 1 Environnement et Prérequis

Pour pouvoir profiter du jeu, il faut être sur Linux, avoir GCC ou Clang et Cmake d'installés. Cmake est un environnemnt de build qui est vachement pratique pour les projets de taille moyenne.

# 2 Compilation et execution

Après avoir unzip le projet dans un dossier, dirigez vous vers ce dossier.
Voici les commandes à taper pour compiler et executer le jeu;

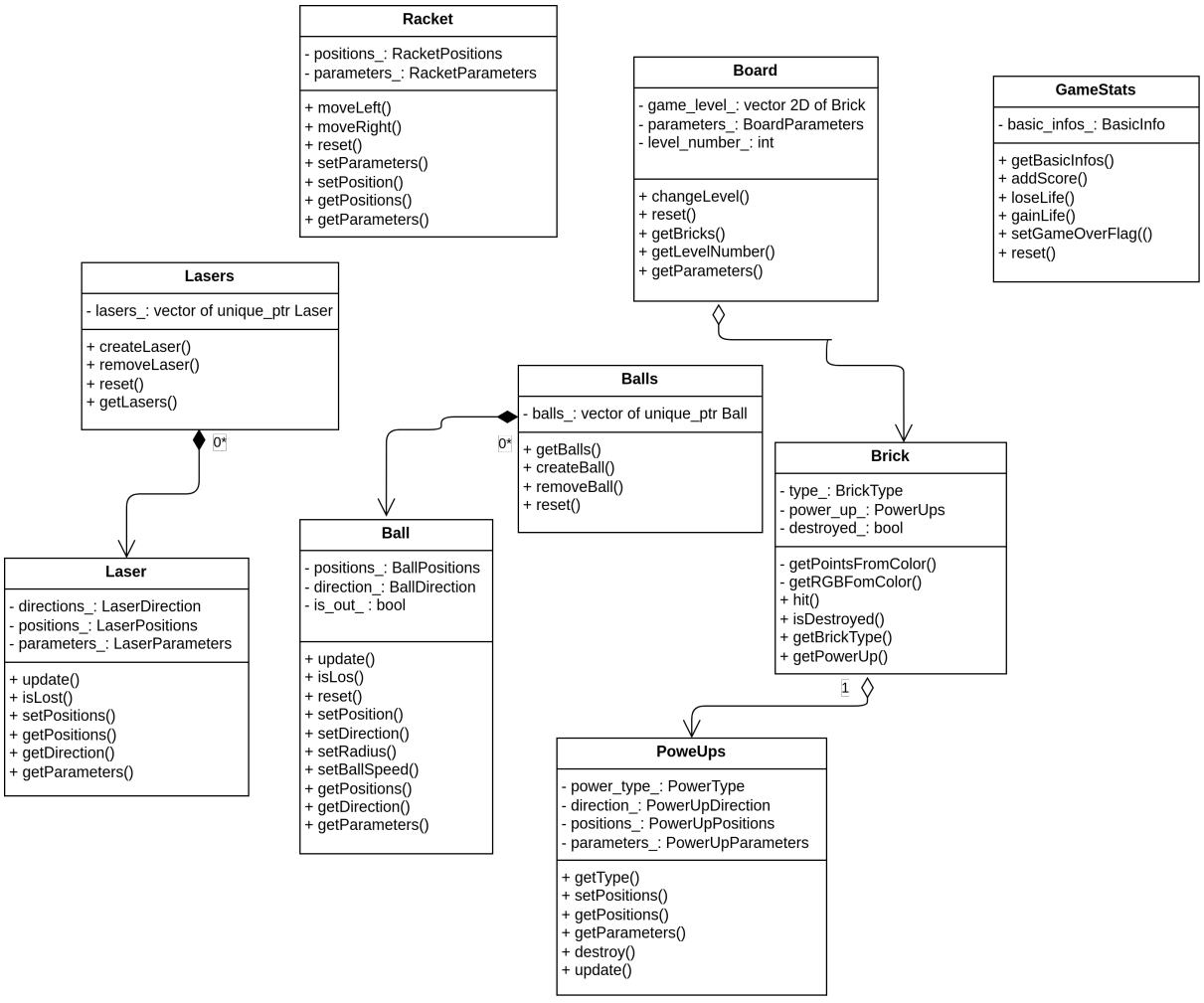
```
mkdir build && cd build
cmake ..
cmake --build .
./ARKANOID
```

# 3 Tâches réalisées

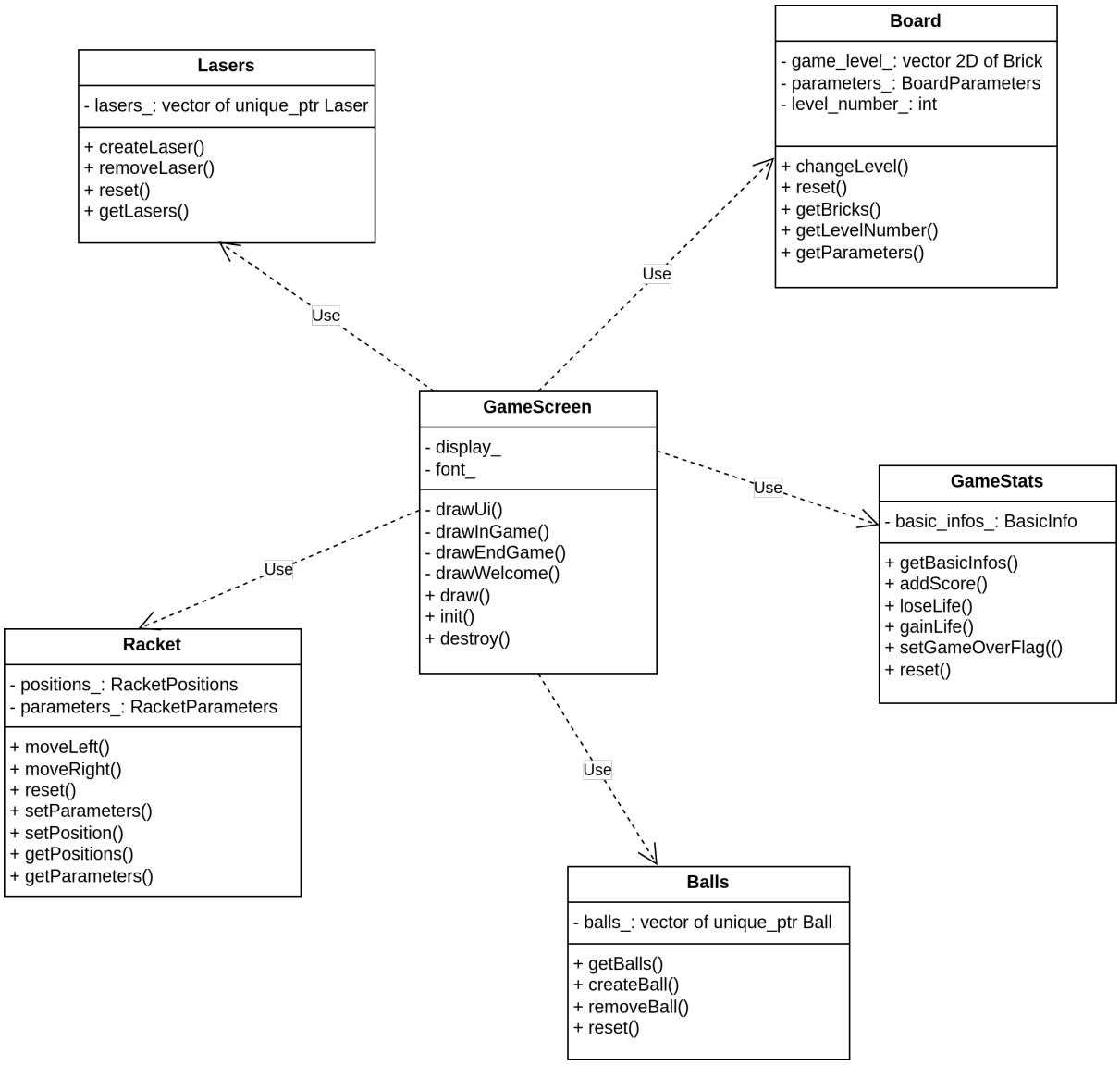
- Toutes les tâches de bases ont été accomplies.
- Toutes les tâches additionnelles ont été accomplies.

# 4 Relations classes

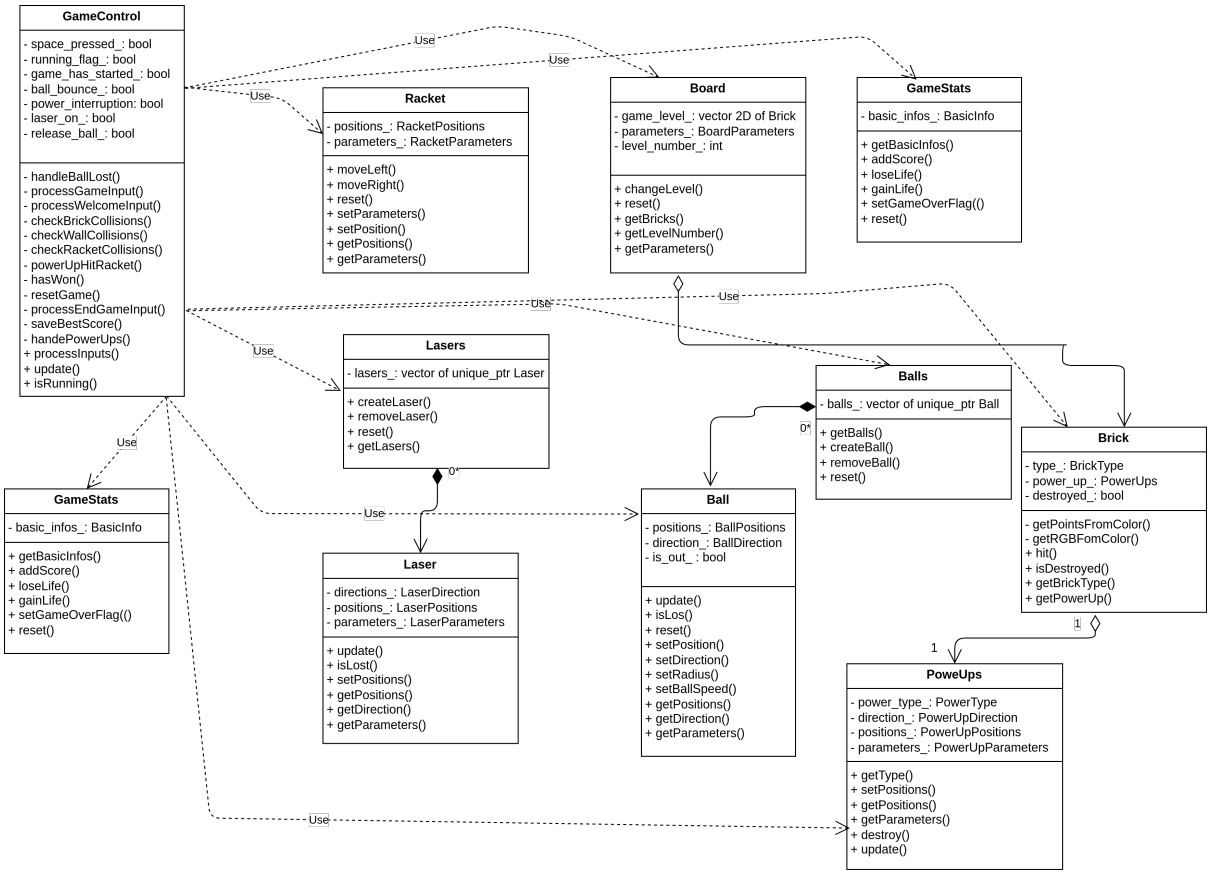
## 4.1 Modèle



## 4.2 Vue


Board
- game_level_: vector 2D of Brick
- parameters_: BoardParameters
- level_number_: int
+ changeLevel()
+ reset()
+ getBricks()
+ getLevelNumber()
+ getParameters()
Lasers
- lasers_: vector of unique_ptr Laser
+ createLaser()
+ removeLaser()
+ reset()
+ getLasers()
Use
Use
GameScreen
- display_
- font_
- drawUi()
- drawInGame()
- drawEndGame()
- drawWelcome()
+ draw()
+ init()
+ destroy()
Use
GameStats
- basic_infos_: BasicInfo
+ getBasicInfos()
+ addScore()
+ loseLife()
+ gainLife()
+ setGameOverFlag(()
+ reset()
Use
Racket
- positions_: RacketPositions
- parameters_: RacketParameters
+ moveLeft()
+ moveRight()
+ reset()
+ setParameters()
+ setPosition()
+ getPositions()
+ getParameters()
Use
Balls
- balls_: vector of unique_ptr Ball
+ getBalls()
+ createBall()
+ removeBall()
+ reset()

### 4.3 Controller




# 5 Logique de jeu

Le jeu commence par initialiser les composants Allegro, indispensables au bon fonctionnement du jeu. Ensuite vient la boucle principale, les inputs utilisateurs sont gérés grâce à Allegro et le modèle se modifie en conséquence, la raquette change de position et on met à jour le mouvement de la balle en fonction de sa direction et de sa vitesse. Tant que la variable release_ball_ est à false, la position de la balle est accrochée à la raquette. Une fois que le controller a pris les inputs et a changé le modèle, il est temps d'afficher. La vue s'en occupe grâce au fonctions draw dédiées aux différents états de jeu. Dans le controller, à chaque update on vérifie le vecteur 2D qui représente les briques, et on regarde si il y a eu collision avec la balle. On fait de même pour les collisions avec le mur, le plafond et la raquette puis on affiche. Ainsi la boucle et bouclée et rebelote jusqu'à ce que l'on change d'état de jeu et quitte le jeu, la boucle principale.

# 6 MVC

La structure de code respecte le principe Modèle, Vue, Controller. Les fichiers sont triés dans les dossiers correspondants et l'utilisation des fichers respecte la catégorie dans laquelle il s'y figure.

### 6.1 Modèle

Le Modèle s'occupe de tous les objects manipulés et affichés du jeu.

### 6.2 Vue

La Vue s'occupe de prendre les objets du modèle et de les afficher grâce à la librairie Allegro.

### 6.3 Controller

Le Controller s'occupe de prendre les inputs utilisateur et de modifier le modèle en conséquence ainsi que de mettre à jour les objets au cours du temps.

## 7 Copilot

Github Copilot a été utilisé lors du développement du jeu. Je me suis retrouvé seul à devoir construire le jeu et copilot m'a aidé à être plus aguéri lors de prise de décisions. Il m'a également été utile pour la documentation et le refactoring qui permet une maintenance et une lisibilité de bien meilleure qualité.