

Project JVC - Chess 3D

Tuna Acikbas & Olivier Pillods

May 2023

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | Modélisation et Implémentations | 3 |
| 2.1 | Choix des technologies et des structures | 3 |
| 2.2 | Modélisation du jeu d'échec (standard) | 3 |
| 2.3 | Règle variante et pièces féeriques | 4 |
| 3 | Diagramme de Classes | 7 |
| 3.1 | Contrôleur | 7 |
| 3.2 | Modèle | 8 |
| 3.3 | Vue | 9 |
| 4 | Algorithmes | 10 |
| 4.1 | Les algorithmes de déplacement et d'attaques | 10 |
| 4.2 | Les mouvements spéciaux | 10 |
| 4.3 | Les affichages | 11 |
| 5 | Répartition du Travail | 12 |
| 5.1 | Choix de départ | 12 |
| 5.2 | Méthodes appliquées | 12 |
| 5.3 | GitLab | 12 |
| 6 | Implémentations bonus (Olivier) | 13 |
| 6.1 | Classes et fonctionnalités visuelles | 13 |
| 6.2 | Importation de fichiers .obj | 14 |
| 6.3 | Création et export des pièces avec Blender | 14 |
| 6.4 | Cimetière | 17 |
| 6.5 | Retour en arrière | 17 |
| 6.6 | Animations | 17 |

1 Introduction

Olivier : "Ayant de l'expérience et des compétences avancées dans le domaine de la création de jeu-vidéo et en modélisation 3D, j'ai souhaité réaliser ce projet en 3D, tout en restant sur les technologies Java et JavaFX. Il y a une section supplémentaire à la fin de ce rapport, qui développe les implémentations hors barème, relatif au visuel. Le temps de travail que j'ai dépensé pour ces fonctionnalités est en dehors de la répartition du travail à deux, c'était juste pour la passion."

Voici quelques captures d'écran de notre jeu d'échecs :



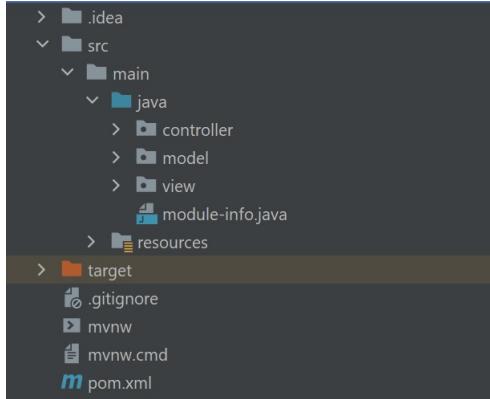
2 Modélisation et Implémentations

2.1 Choix des technologies et des structures

Nous utilisons l'environnement de développement intégré (IDE) IntelliJ, avec le framework Maven. Le code a été réalisé intégralement en Java, avec JavaFX pour l'interface graphique.



La structuration de ce projet a été fait en se basant sur le principe du modèle MVC (Model-View-Controller). Nous avons un dépôt Git sur le GitLab de l'université, ce qui nous a permis de travailler en parallèle. Ci-dessous une capture d'écran du squelette de notre arborescence avec Maven et le modèle MVC :



2.2 Modélisation du jeu d'échec (standard)

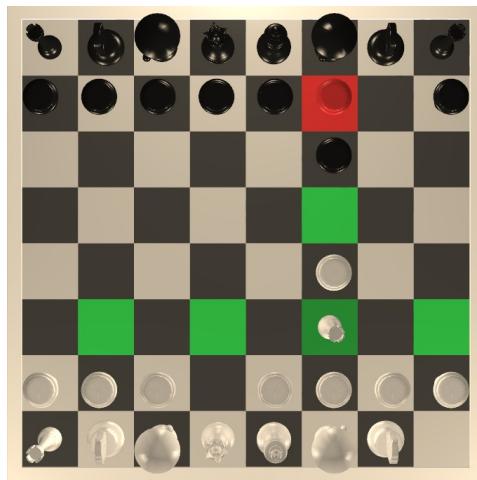
Nous avons imaginé les différents modèles :

- Une Pièce, qui serait abstraite.
- Le Pion, la Tour, le Cavalier, le Fou, la Reine, le Roi, qui hériterait de Pièce, chacun ayant des mouvements spécifiques.
- Une classe Position, qui gère des coordonnées (x, y) d'un plateau de 8x8.
- Une classe Plateau (Board), qui contient un tableau statique de 64 Pièces (donc chaque élément est soit instancié, soit *null*).
- Des classes dans le contrôleur et la vue, qui utiliseront ces modèles.

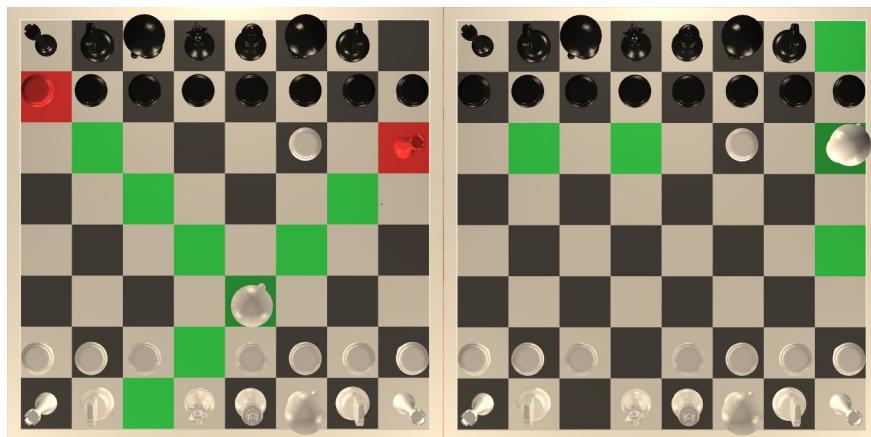
2.3 Règle variante et pièces féeriques

Nous avons inventé et mis en place 4 pièces féeriques qui s'inspirent des pièces existantes :

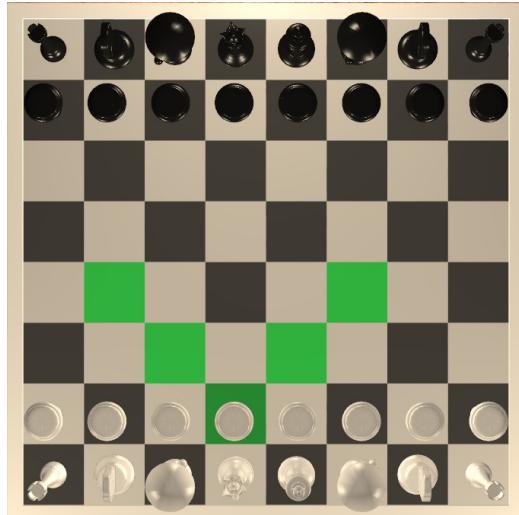
- *Nouvelle range, 'de pas' 2* : Le Géant (Giant) es basé sur le déplacement de la Tour (Rook). Cependant, il peut seulement se déplacer sur les cases qui ont le même couleur que celui de sa case de départ, mais en contrepartie, il n'est pas bloqué par les pièces. (Au lieu d'inventer une range maximale ou minimale, nous avons inventé une range 'de pas' 2).



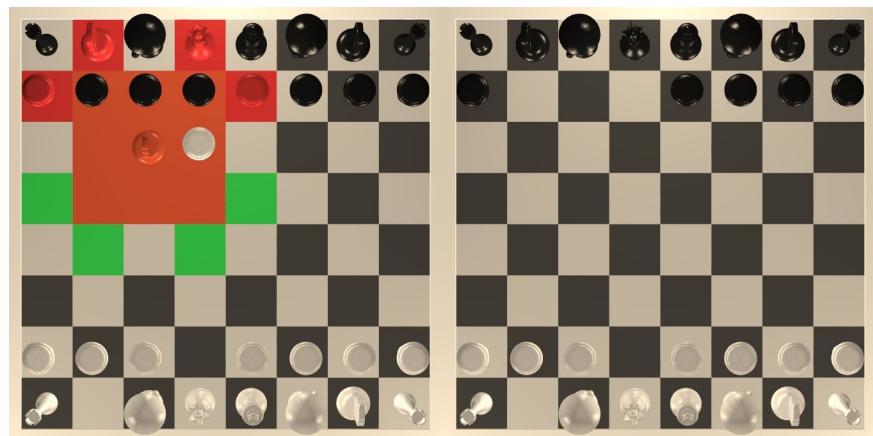
- *Nouveau déplacement, le vol de mouvements* : Le Magicien vole les mouvements de la pièce adverse qu'il mange. Il possède initialement le déplacement du fou (Bishop).



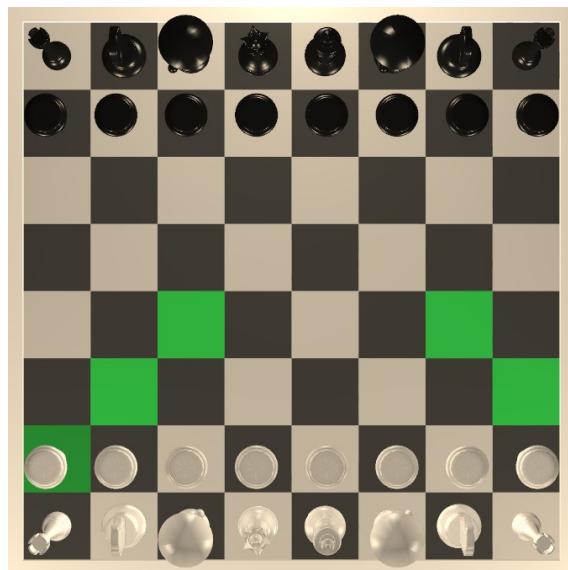
- *Nouveau déplacement, le pion inversé* : Le Nwap qui possède les mouvements d'un Pion qui avance en diagonale et qui mange en avant. (La promotion et la prise en passant fonctionnent aussi pour le Nwap).



- *Nouvelle attaque, l'explosion* : Le Kamikaze possède les mouvements du chevalier. Ce dernier peut se suicider en détruisant toutes les pièces (alliés et ennemis) autour de lui dans une range de 1 case.

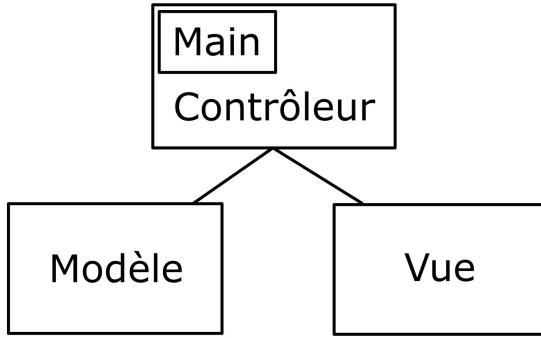


Pour le choix d'une règle qui sera utilisé durant le jeu variant, nous avons décidé de modifier les déplacements qui étaient possibles. C'est ce qui nous a amené d'imiter les mouvements du serpent du jeu (Snake). Les pièces peuvent directement aller d'un côté du plateau de jeu à un autre. (Pour que cela ne soit pas trop "extrême", nous avons implémenté cette règle pour les bords droite-gauche et gauche-droite uniquement)



3 Diagramme de Classes

Dans cette section, nous allons aborder le diagramme de classe de notre projet et voir quelques points précis.



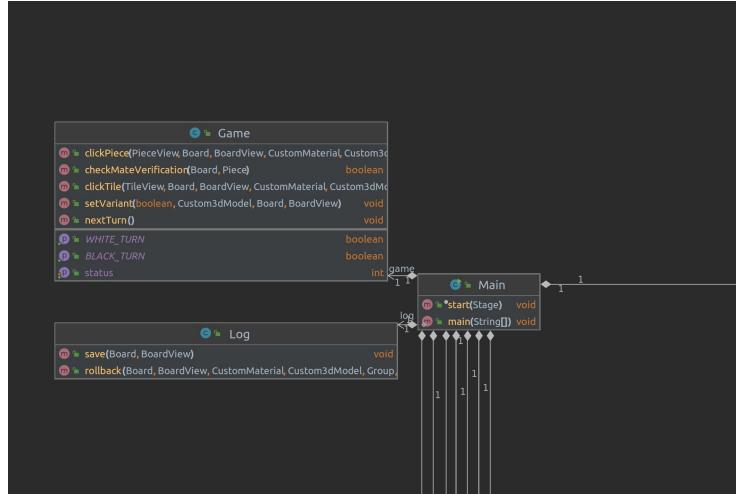
3.1 Contrôleur

Le contrôleur est composé de Trois éléments :

Main : La Classe qui permet de lancer le jeu

Game : Le Contrôleur principal qui contrôle l'action des joueurs

Log : La Classe qui s'occupe de sauvegarder une copie du plateau de jeu à la fin de chaque tour pour faire des retours en arrières

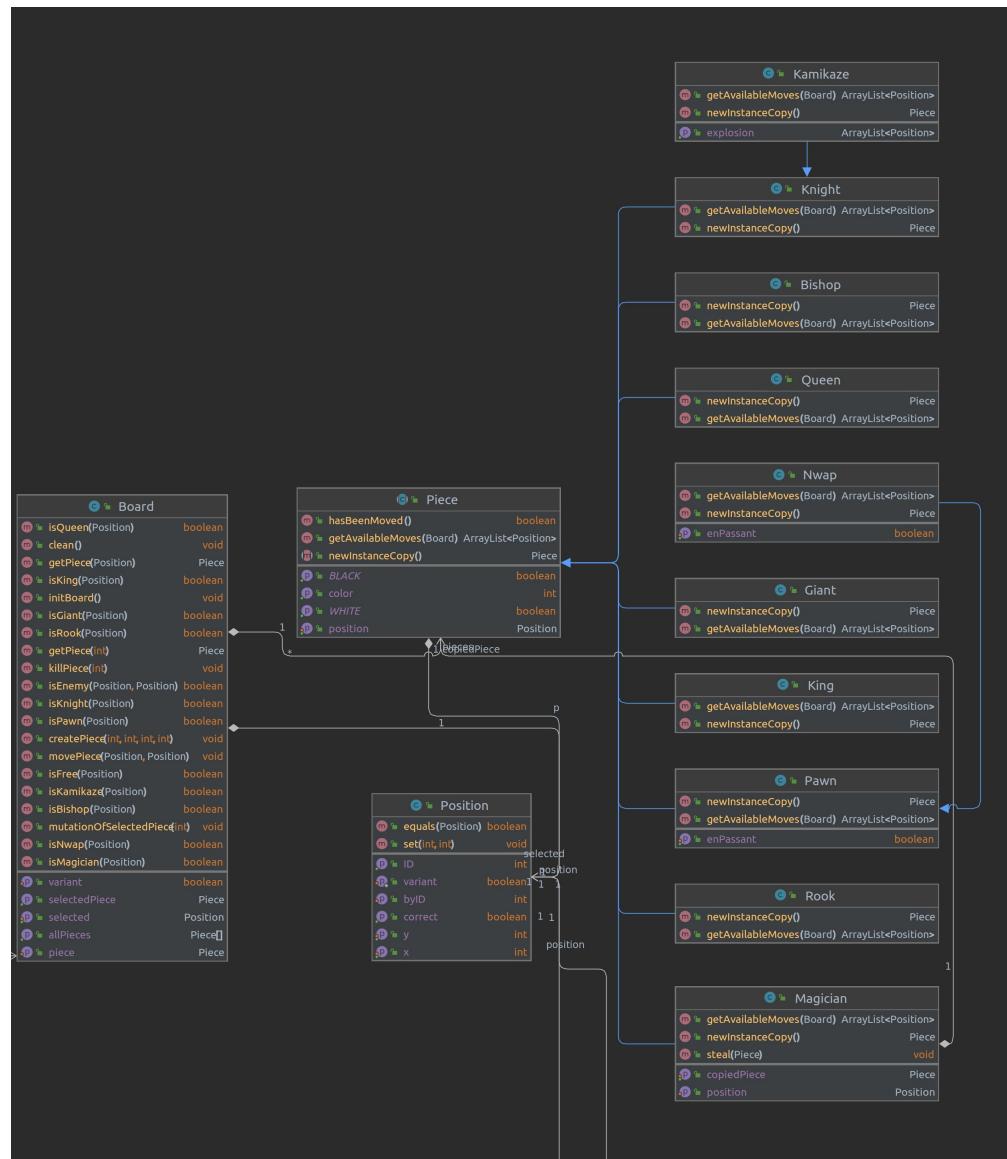


3.2 Modèle

Piece : La Classe principale dont toutes les pièces héritent leurs attributs communs.

Board : La Classe qui contient le plateau de jeu avec l'emplacement des pièces.

Position : La Classe qui gère les positions des pièces.

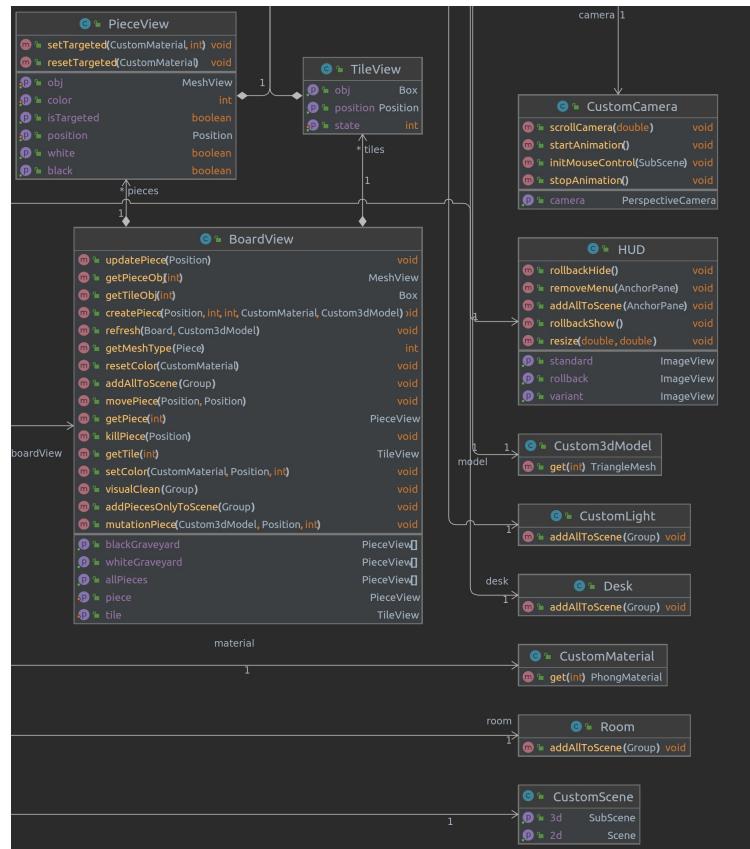


3.3 Vue

BoardView : La Classe qui contient les pièces et cases à afficher.

PieceView : La Classe qui s'occupe de la visualisation des pièces.

TileView : La Classe qui s'occupe de la visualisation des cases.



4 Algorithmes

Eh bé les pièces du modèle elles disent où elles vont, wouala, et le contrôleur est tout magik, et il dit à la vue "mets-leur des paillettes dans leur vie!"

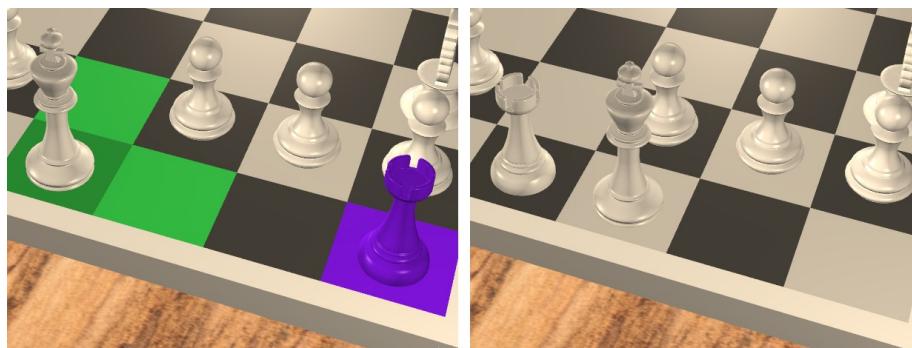
Ce projet contient différents types d'algorithmes :

4.1 Les algorithmes de déplacement et d'attaques

Les algorithmes de déplacement sont gérés par la fonction par `clickPieces`. Chaque fois que le joueur clique sur une pièce, elle fait une demande auprès de la pièce correspondante dans le modèle. Chaque pièce possède une fonction `getAvailableMoves` qui renvoie une array des mouvements possible, en argument, elle prend le plateau de jeu (board). L'array renvoyée est une liste dynamique (ArrayList) de positions, la classe Position représente un tuple x et y du plateau d'échec. La fonction `clickPieces` récupère cette array pour ensuite parcourir toutes les cases pour afficher la nature de l'action (5 actions possibles : `NORMAL`, `SELECTED`, `MOVABLE`, `ATTACK`, `SPECIAL`, `EXPLOSION`) et modifier leurs états (une couleur par état visuellement). Par la suite, L'utilisateur peut cliquer sur ces cases, qui déclencheront la fonction `clickTile` qui en vérifiant la nature de l'action va modifier les positions des pièces.

4.2 Les mouvements spéciaux

Les mouvements spéciaux (promotion, rook, en-passant) sont également gérés au sein des deux fonctions de clic. Ils sont représentés sur les cases du plateau en violet, avec le statut `SPECIAL`. Lorsque l'on clique sur celles-ci, la fonction `clickTile` teste et cherche quel mouvement spécial est à exécuter. Grâce au type de la pièce sélectionnée, et des positions de la pièce sélectionnée et de la case sélectionnée, la fonction trouvera l'action adéquate entre promotion, rook, et prise en passant.



4.3 Les affichages

Les algorithmes d'affichages sont utilisés dans trois cas. La première qui consiste à générer les assets nécessaire pour le jeu (le plateau de jeu, les pièces, la salle de jeu). Le deuxième est faite après l'appel à la fonction `clickPieces` quand les "états des cases" on était changé selon les mouvements possibles". Il existe 5 "état" possible qui sont:

- Vert foncé pour la pièce sélectionnée.
- Vert pour les déplacements.
- Rouge pour les attaques.
- Violet pour les coups spéciaux.
- Orange pour les explosions.

Le dernier algorithme d'affichage entre en jeu pendant le retour en arrière, elle récupère la Capture faite avant chaque coup et le stock dans une liste quand la fonction `RollBack` est appelé, elle replace toutes les pièces de manière à retourner un coup en arrière 6.6.

5 Répartition du Travail

5.1 Choix de départ

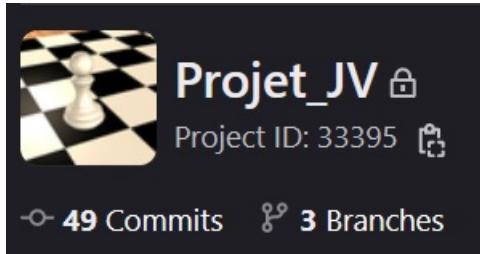
Ayant une structure suivant le modèle MVC, nous avons décidé de répartir les différents composants entre nous. Tuna s'est occupé du Modèle, Olivier s'est occupé de la Vue, et ensemble, nous avons fait le Contrôleur.

5.2 Méthodes appliquées

Nous avons fait des appels discord réguliers à la maison, ce qui nous a permis de coder à deux en partage d'écran (c'est-à-dire que l'un écrit sa partie, et l'autre regarde, donne des idées et commente), et aussi de partager régulièrement sur nos avancements et notre organisation. Nous n'avons pas choisi de rythme fixe, mais la régularité était au rendez-vous, avec des placements de deadlines intermédiaires pour différentes fonctionnalités. Au final, nous étions assez proches du système de sprints de la méthode agile.

5.3 GitLab

Nous avons crée un dépôt Git sur le GitLab de l'université pour pouvoir push / pull régulièrement l'intégralité du projet IntelliJ, et pour avoir deux branches de développement en plus de la branche 'main'.



6 Implémentations bonus (Olivier)

6.1 Classes et fonctionnalités visuelles

Pour pouvoir implémenter le jeu en 3D, j'ai eu besoin de modifier et d'ajouter des classes dans la vue. Voici une présentation de chacune des classes :

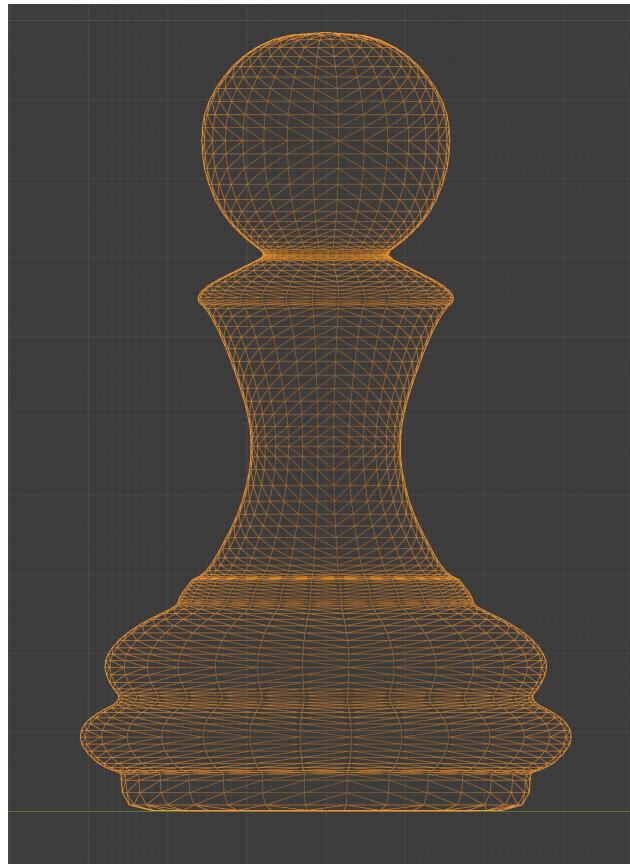
- CustomScene qui contient une scène 2D et une scène 3D (l'une est dans l'autre).
- HUD (Head Up Display, Affichage Tête Haute) qui gère les affichages sur la scène 2D, quelle que soit la taille de la fenêtre.
- CustomCamera qui gère la caméra en perspective. Elle permet un déplacement avec un glissement de souris, et un zoom avec la molette. Les déplacements sont limités pour rester "cohérent" et pour toujours regarder le plateau.
- CustomLight qui gère la lumière ambiante et quelques points de lumières proches du plateau.
- CustomMaterial qui gère l'initiation des différents shaders (\approx apparence du modèle 3d), certains avec une couleur unie (pièces, cases) et certains avec une texture (bois, moquette, mur).
- Desk et Room qui affichent une table en bois, et une pièce-studio (avec un sol en moquette beige, des murs avec de la mousse acoustique noire, et un plafond blanc).
- TileView qui contient le plateau avec les cases noires et blanches. Celles-ci sont colorables temporairement pour l'affichage des mouvements disponibles. (Vert pour les mouvements, Rouge pour les attaques, Violet pour les mouvements spéciaux, Orange pour les explosions, etc...)
- PieceView gère l'affichage d'une pièce en tant que modèle 3D. (objet 3D, position, couleur).
- BoardView contient toutes les cases du plateau (TileView), et toutes les pièces (PieceView), et contient des fonctions d'affichage des cases (couleur) et d'affichage des pièces (création, déplacement, mutation, placer au cimetière, couleur, etc...).
- Custom3dModel pour terminer, importe des fichiers .obj et transforme / convertit l'ensemble des triangles de cet objet en "TriangleMesh" (classe JavaFX), utilisables par PieceView en tant que "MeshView" (classe JavaFX qui contient un set de "TriangleMesh").

6.2 Importation de fichiers .obj

L'importation d'un fichier modèle 3D n'existe pas en JavaFX, j'ai codé l'importation des fichiers .obj moi-même. Je me suis limité à une importation 1-1 pour faire simple et clair (1 objet = 1 fichier .obj), pas de texture et pas d'UV mapping. Je cherche deux listes de données, [1] les vertex (qui contiennent une coordonnée x,y,z) et [2] les triangles (qui contiennent trois ID de vertex). Je n'ai pas implémenté la lecture de "quads", seulement la lecture de "tris", de toute façon en 3D un polygone à quatre cotés est en fait deux triangles, il me suffit de "trianguler" le modèle 3D au préalable avant l'export.

6.3 Création et export des pièces avec Blender

J'utilise le logiciel Blender pour faire de la modélisation 3D. J'ai récupéré un set de pièces d'échecs (standard) en ligne pour avoir une base. J'ai ensuite édité les pièces pour leur rajouter du détail, avoir des arrondis précis, et les meshes toujours triangulées. J'ai exporté chaque pièce (fonctionnalité inclue dans Blender) en format OBJ Wavefront. Voici à quoi ressemble le pion avant l'export :



Blender est un outil très puissant, je me suis amusé à faire un shader de pièces noir semi-brillantes et un shader de pièces en verre. Voici quelques rendus en RayTracing des pièces :





J'ai crée les pièces variantes à partir des pièces standard. Le Nwap est un Pawn (pion) avec des mouvements inversés, visuellement c'est un pion à l'envers. Le Giant est une tour très fine et grande. Le Kamikaze est un cavalier qui cache bien son jeu (pas de changement visuel ahaha). Le Magician est un fou qui possède un chapeau asiatique et un bâton magique.



6.4 Cimetière

Lorsqu'une pièce est mangée (ou quelconque mort), je la place à côté du plateau. J'ai crée deux listes statiques de 16 pièces appelées `blackGraveyard` et `whiteGraveyard`



6.5 Retour en arrière

J'ai crée une classe Log, dans le contrôleur, qui sauvegarde l'état intégral du plateau à chaque coup effectué. Elle permet, au clic de l'icône de rollback, de revenir en arrière d'un coup. La sauvegarde fait une copie intégrale des pièces, une par une, et le retour en arrière aussi. Cela permet d'éviter des effets de bord, car les instances sont des pointeurs.



6.6 Animations

J'utilise la classe "TranslateTransition" de JavaFX pour que les déplacements (sur plateau) des pièces se font de manière transitionnelle et non par téléportation.