# Document de gestion de projet

Jeu du Mancala

Antonin Huaut

# Conception

Le jeu du Mancala est découpé en deux applications : le serveur et le client. Les deux projets sont des projets Maven, permettant de simplifier la gestion des dépendances et de génération de la cible. Les projets utilisent le gestionnaire de versions nommé Git, accessible [ici](https://gitlab.ecole.ensicaen.fr/huaut/Mancala).

## Serveur

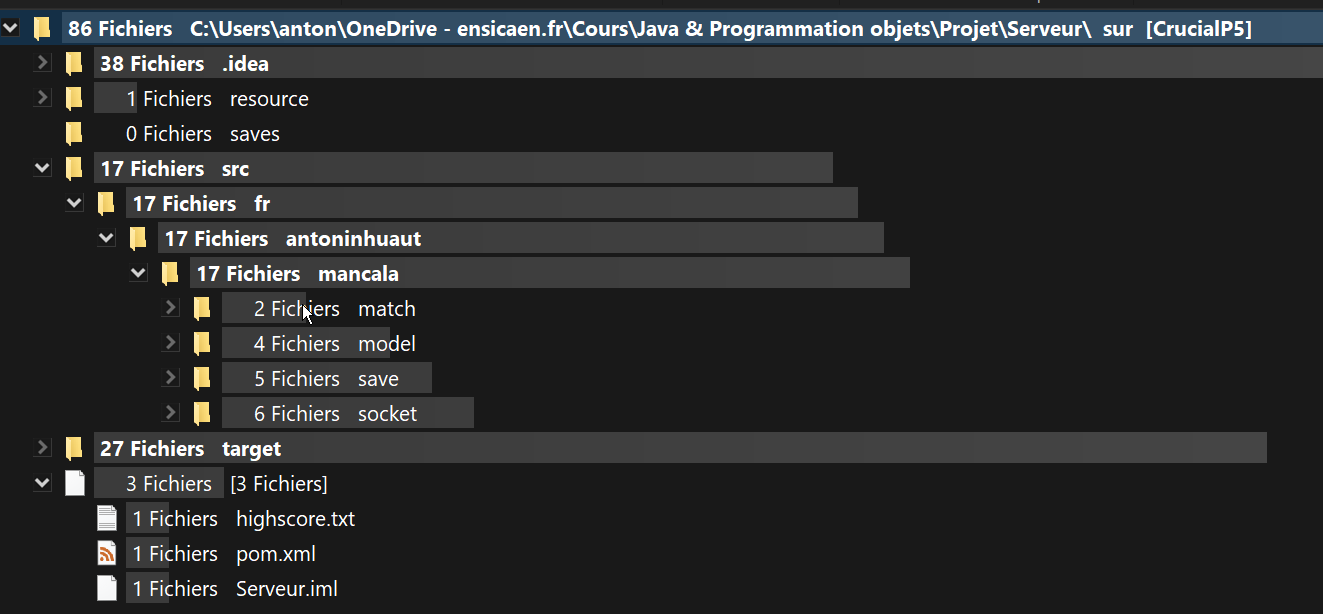


Image  : Architecture du serveur

Le serveur étant un projet Maven possède un fichier pom.xml qui détaille notamment le build et les dépendances. Le projet dépend de Gson, une bibliothèque JSON qui est utilisée pour gérer les sauvegardes (sera détaillé plus tard), de Log4J (core & api) car l’application intègre un système de log et enfin de sonar-maven qui est un runner sonar pour Maven.

On retrouve un dossier « resource » comportant le fichier de configuration pour log4j et le dossier « src » qui contient le code.

Celui-ci est structuré en plusieurs parties :

* Socket/ : Ce sont toutes les classes qui ont pour principal fonction de traiter des sockets. Il y a le système de Session (pour gérer plusieurs parties en même temps), le Joueur (et la communication entre Serveur ⬄ Client), la classe principale de l’application qui crée la socket serveur et deux énumérations qui correspondent aux différents messages que le serveur ou le client peut envoyer à l’autre.
* Save/ : Cela correspond au système de sauvegarde et de gestion des scores récents.
* Model/ : Contient des classes modèles
* Match/ : Contient les classes de déroulement du jeu (Match, Round, Move)

Le serveur peut gérer plusieurs sessions de jeu en parallèle, gère la déconnexion de joueurs et leur reconnexion. Si une session n’a plus de joueur, elle sera supprimée.

Une hard limite de session est présente à 200 sessions en parallèle (pour éviter des attaques). La réception des messages des clients est sécurisée : un message invalide éjectera le client. Un message incorrect ou incomplet sera ignoré. Les actions des clients sont contrôlées pour éviter la triche.

## Client

Le Client est plus complexe que le serveur, notamment car il intègre la composante graphique.

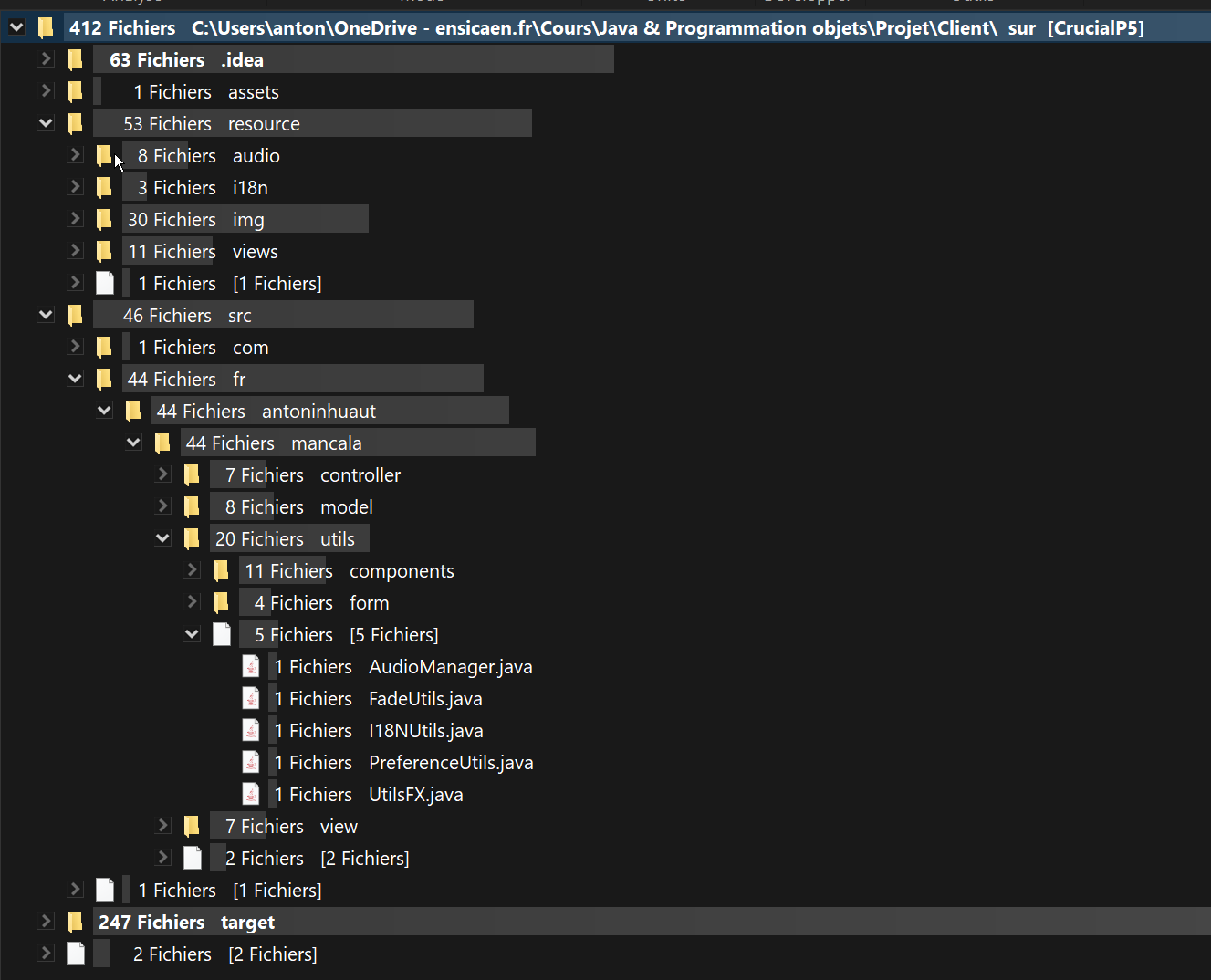


Image  : Architecture du client

Le client est aussi un projet Maven, il possède un pom.xml qui contient les dépendances et le build.  
Dépendances :

* JavaFX (base, controls, graphics, fxml et media)
* Log4J, tout comme le serveur, il intègre un système de logging

Et du build :

* Pour créer un runnable de développement
* Pour récupérer et copier toutes les dépendances dans un dossier
* Pour générer un exécutable jar (sans les dépendances)
* Pour générer un installateur pour Windows comportant un JRE allégé + les dépendances + l’application

Au niveau dossier, il y a un dossier « assets » qui sert pour la génération de l’installateur pour Windows. Le dossier « resource » possède tous les sons et les musiques de l’application, les messages de l’internationalisation (anglais / français), les différentes images de l’interface graphique, les fichiers FXML des vues et le fichier de configuration de Log4J.

Et enfin le dossier « src » contenant les sources :

* Le package com.jdotsoft.jarloader contient une classe permettant le chargement dynamique de bibliothèques installées par l’installateur Windows.
* Le fichier module-info.java détaillant les modules utilisés pour la JVM[[1]](#footnote-1)
* Le package fr.antoninhuaut.mancala contient le code de l’application
  + A la racine se trouve le fichier d’entré de l’application et la classe initialisant l’application JavaFX.
  + Controller/ : Contient tous les contrôleurs des vues
    - Le sous-dossier global/ est spécial, il contient tous les contrôleurs de vue « fixes » (barre de titre, fenêtre, menu, barre d’outils) et les classes abstraites. C’est aussi ici qu’est géré le changement de la vue variable (vue de connexion, vue du plateau, …).
  + Model/ : Contient toutes les classes modèles de l’application
  + Utils/ : Contient de nombreuses classes utilitaires ou composants graphiques
    - Components/ :
      * Alert : Alert JavaFX personnalisées
      * Dialog : Dialog JavaFX personnalisées
      * SwitchButton : Composant JavaFX représentant un bouton swichable 
    - Form/ : Contient les classes de conversion et vérification du formulaire de connexion (Validateur d’entier, retrait des espaces, …)
    - AudioManager : Singleton gérant la partie audio de l’application
    - I18NUtils : Singleton gérant l’internationalisation de l’application
    - PreferenceUtils : Singleton pour sauvegarder des préférences utilisateurs (langue, pseudo, ip & port du serveur, paramètres son/plateau)
    - UtilsFX : Classe utilitaire pour l’interface graphique, notamment pour l’effet « fade » lors de la minimisation de l’application
  + View/ : Contient les vues de l’application
    - Le sous-dossier global/ est spécial, il contient les vues « fixes » et les classes abstraites

# Fonctionnalités

Interface graphique :

* Interface avec plusieurs vues stylisées
  + Barre de titre personnalisée :
    - Déplacer la fenêtre comme une application classique
    - Minimiser la fenêtre (ou via l’icône) avec une animation de « Fade »
    - Quitter l’application
  + Menu personnalisée (détaillé dans partie correspondant au jeu)
  + Barre d’outils personnalisée :
    - Changement de langue : Anglais ⬄ Français
    - Retourner à l’accueil (et quitter sa partie)
    - Pop-up d’information (auteur du programme, bouton vers le Gitlab)
* Internationalisation de l’interface
  + Langues supportées : Anglais & Français (mais support pour en rajouter)
  + Fichiers langues externalisés du code, présent dans le dossier « resource »
  + Possibilité de changer de langue dans n’importe quelle vue avec effet immédiat
* Deux vues variables majeures :
  + Vue de connexion :
    - Possibilité de rentrer son pseudo, l’ip et le port d’un serveur
    - Vérification des champs (pseudo sans espace, port = un entier, …)
    - Spinner durant la connexion au serveur, message si échec de connexion
  + Vue du jeu :
    - Vue d’attente : Affichée si la session auquel le joueur est connecté ne comporte pas 2 joueurs. (Ou si un joueur quitte une session, en attendant sa reconnexion ou qu’un autre joueur se connecte)
* Choix d’ergonomie :
  + Menu et options accessibles en un clic
  + Menu toujours présent pour interagir avec la partie rapidement

Jeu du Mancala :

* Règles implémentées
  + Les parties du Mancala comportent deux joueurs.
  + Il y a 12 trous, 6 par joueur, avec 4 graines dans chaque au démarrage d’une partie.
  + Le premier joueur qui joue est tiré au hasard.
  + Un joueur qui joue prend l’ensemble des graines d’un de ses trous et distribue les graines, une par trou, dans le sens inverse des aiguilles d’une montre.
  + Si la dernière graine est dans le camp adversaire et qu’il y a 2 ou 3 graines dans ce trou, il récupère ces graines, et on regarde la case précédente de la même façon jusqu’à être dans son camp OU qu’il n’y a pas 2/3 graines.
  + Si on fait un tour complet ou plus, on ne remet pas de graine dans le trou qu’on vient de jouer.
  + Si l’adversaire n’a plus de graine et qu’on peut jouer un coup qui lui en redonne, le joueur est forcé de faire se coup. Sinon la partie s’arrête et il capture les graines restantes.
  + Si un coup doit affamer un adversaire, alors celui-ci est jouable mais aucune capture n’est faîte.
  + La partie s’arrête quand un joueur a au moins 25 graines (ou arrêt match ou abandon ou voir autres règles).
  + Quand il reste 10 graines ou moins sur le plateau, le joueur qui a la main peut proposer l’abandon à l’adversaire. S’il accepte, les graines restantes sont partagées.
  + S’il reste moins de 6 graines sur le plateau, la partie est nulle. (Sauf si un joueur a au moins 25 graines).
* Menu du jeu
  + Charger une partie : Le joueur doit entrer le code de la session, par défaut, le champ est prérempli avec celui de sa session. Il peut le changer pour entrer celui d’une autre session pour récupérer une précédente sauvegarde. La partie chargée est alors exactement au même stade qu’à la sauvegarde (qui doit jouer, score, graines dans les trous, manches gagnées par joueur, …). Seuls les pseudos des joueurs sont remplacés par ceux de la session en cours.
  + Sauvegarder une partie : La partie en cours est sauvegardée du côté du serveur. Un pop-up est affiché à l’utilisateur qui lui indique son code de session, c’est le code qui permettra de recharger la sauvegarde plus tard. S’il recharge la sauvegarde dans la même session, il n’aura pas besoin d’entrer le code.
  + Abandonner partie : Le joueur abandonne le round en cours (mais pas toute la partie !). L’adversaire capture toutes les graines restantes.
  + Nouveau match : Un nouveau match est relancé, les scores et le nombre de round gagné sont remis à 0. De même pour le plateau.
  + Arrêter le match : Termine le match en cours.
  + Annuler le dernier coup : Le joueur annule son dernier coup, il ne peut le faire que lorsque c’est au tour de l’adversaire de jouer. (Donc quand il vient juste de jouer)

Général :

* Préférence todo
* Menu outils todo

# Problèmes rencontrés

Jdk 11.0.11 / 11.0.9

Installateur Windows

# Idées d’améliorations

# Remarques

1. Pour générer un JRE allégé pour l’installateur, j’utilise l’outils jdeps et jlink (intégré à Java 9) pour lister les modules Java utilisés par mon projet, permettant de générer un JRE ne contenant que le strict nécessaire. [↑](#footnote-ref-1)