****

**Rapport**

**Projet Encadré – POO en Java**

**Pac-Man**

Filière Informatique par apprentissage

Années 2019 - 2020



**Maxence BESSY**

**Thomas BECHET**

Sommaire

[1 – Fonctionnalités de l’application 3](#_Toc41844201)

[2 – Architecture du projet 4](#_Toc41844202)

[3 – Répartition des rôles 4](#_Toc41844203)

[4 – Problématiques 4](#_Toc41844204)

# 1 – Fonctionnalités de l’application

Afin de répondre au sujet proposé, nous nous sommes basés avant tout sur les règles de base du jeu Pac-Man sorti en 1980. Celles-ci sont décrites à la page suivante <https://fr.wikipedia.org/wiki/Pac-Man>. Cependant, pour des raisons d’originalité et de différenciation avec les autres équipes nous avons préféré compléter les points flous du jeu avec nos propres règles. De plus, le choix de l’architecture choisie nous a forcé à s’orienté vers une solution alternative. Cependant, le jeu de manière générale reste cohérent avec ce que tous utilisateurs non spécialistes s’attendraient du jeu Pac-Man.

Les fonctionnalités décrites dans cette partie regroupent à la fois les fonctionnalités du jeu en lui-même et les fonctionnalités plus générales de notre application.

**Fonctionnalités de base**

* Choix de la carte au lancement de la partie
* Détection de fin de partie avec le score
* IA basique pour les fantômes
* Libération des fantômes en fonction de leur couleur
* Téléportation à l’autre bout de la carte quand il y a une sortie de carte
* Possibilité de manger les Pacgums et Fruits
* Gestion des Super-Pacgums
* Réapparition du Pac-Man et des fantômes
* Gestion des menus
* Contrôle du Pac-Man

Point à préciser, l’IA des fantômes n’est pas différenciée selon le type de fantôme (bleu, rouge, rose et violet).

**Affichage**

* Compteur de début
* Déplacement fluide pour les entités
* Animations pour les entités
* Animation de réapparition pour les entités
* Animation pour le mode Super-Pacgum
* Affichage du score
* Affichage du nombre de vie restant
* Affichage des murs avec des courbures si nécessaire
* Redimensionnement automatique de la carte en fonction de la taille de la fenêtre

**Multijoueur**

* Création d’une partie avec un nombre de joueur choisi
* Possibilité de rejoindre une partie
* Utilisation du protocole TCP
* Choix du port et de la carte à charger
* Déconnexion automatique du joueur en cas d’échec
* Attente du nombre de joueur requis
* Synchronisation de l’affichage
* Affichage des scores des autres joueurs

**Editeur de cartes**

* Charger une carte
* Créer une nouvelle carte
* Sauvegarder une carte avec un nom
* Possibilité de supprimer une carte
* Placement des différentes tuiles de la carte
* Placement des différentes entités sur la carte
* Possibilité de jouer sur les cartes éditées
* Affichage de la grille pour le placement des tuiles

Nous utilisons un format de carte personnalisé. Ces fichiers sont accessibles sous la forme de fichier textes. Il est donc possible d’éditer manuellement les cartes même si cela n’est pas recommandé. D’ailleurs, il n’y a pas de vérification sur la lecture des cartes.

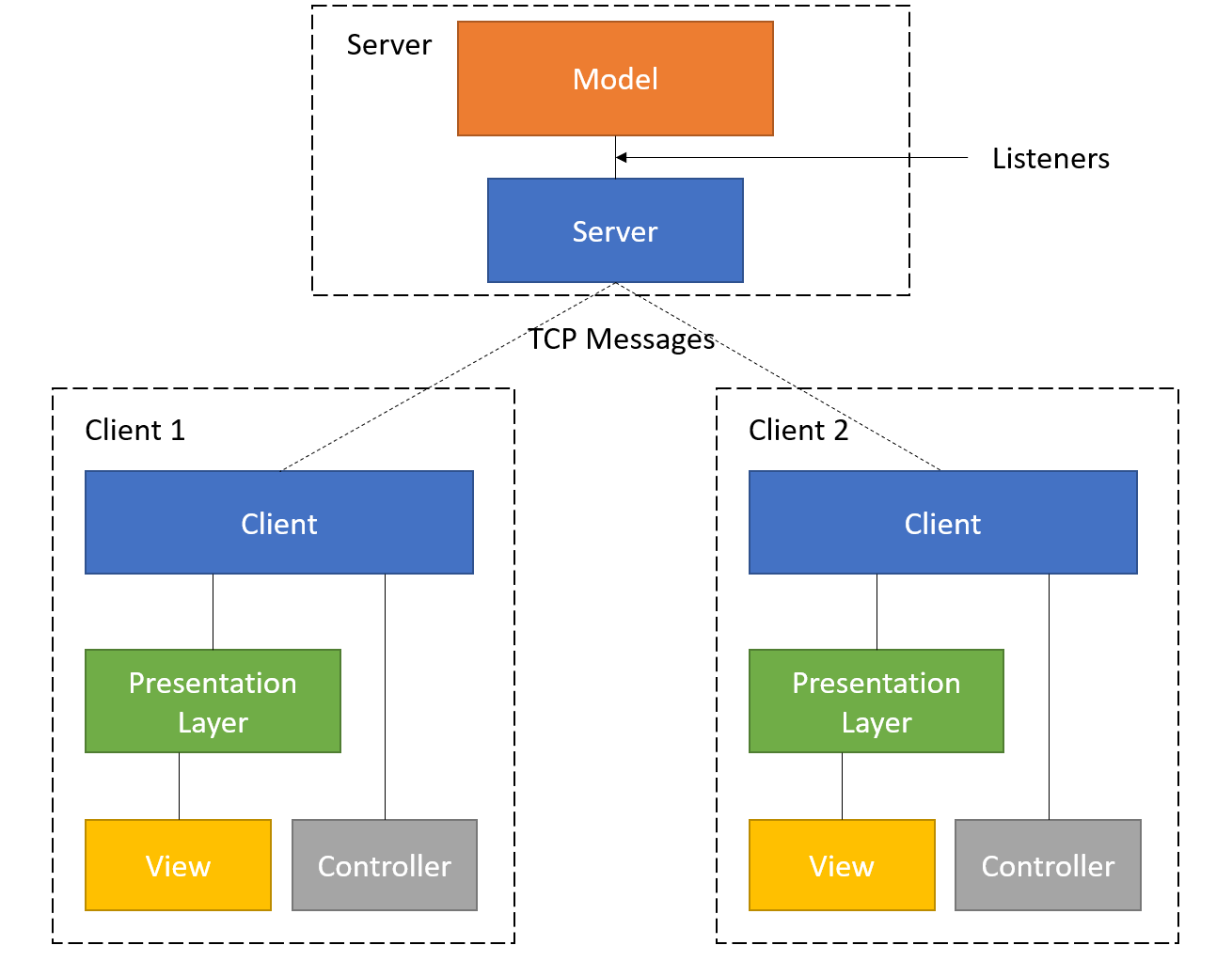
# 2 – Architecture du projet

Le développement de l’application c’est fait en deux temps. Tout d’abord, nous n’étions pas sûr de toutes les fonctionnalités au départ. De ce fait nous sommes parties sur le modèle MVC de manière stricte. La partie modèle du jeu à donc pu être totalement isolée du reste de l’architecture. La vue quant à elle dans une autre partie. Le choix d’implémenter le multijoueur nous a forcé à revoir l’architecture générale.

La principale problématique était celle de la réplication des données sur le réseau. En cherchant les différentes solutions, nous avons décidé de choisir la plus simple. En isolant, les interactions entre un éventuel client et serveur, on remarque que le client n’a que l’information de la direction de son Pac-Man à envoyer. Pour tout le reste, tous les clients sont à l’écoute du serveur. C’est pourquoi nous avons choisi de n’avoir qu’un seul modèle du jeu et les données liées à l’affichage sont répliquées sur le réseau.

En ce qui concerne la réplication des données, nous ne voulions pas spécialiser le modèle du jeu pour le réseau, c’est pourquoi nous utilisons une couche supplémentaire entre la vue et la partie client. Cette couche est appelée *Presentation Layer* et est chargée de simplement formater/déformater les données pour les afficher et les envoyer à la vue.

On peut remarquer qu’avec ce modèle la vue peut quand même écouter directement le modèle (notamment pour le solo). Cependant, pour une question de temps, nous avons préféré ne pas ajouter de cas particuliers. C’est pour cela que, dans le cas du solo, un serveur avec un seul joueur est créé.



Pour la réplication des messages TCP sur le réseau nous avons dû utiliser notre propre protocole d’échange. Ne connaissant pas suffisamment les méthodes optimisées d’échange, nous sommes parties sur un formatage simple et extensible se basant sur l’association d’une clé et d’une valeur.

Chaque message dispose d’un mot clé servant d’entête pour le client. Le client connaissant les mots-clés, il peut rediriger le message (sous la forme d’une chaîne de caractères) et construire le bon message pour l’envoyer au client (Presentation Layer). Les messages peuvent alors contenir des données avec des types concrets.

Voilà des exemples de messages envoyés aux clients :

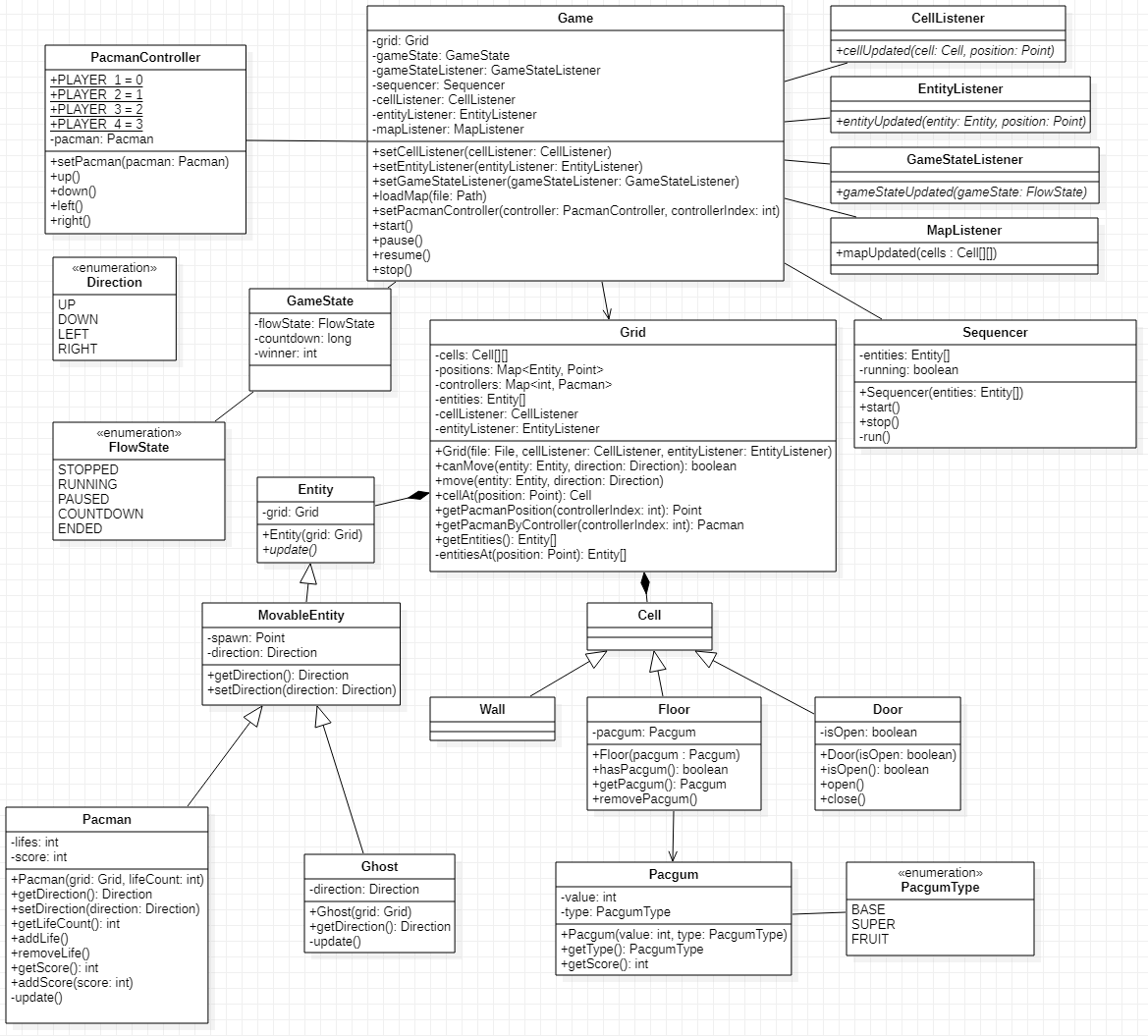
door@x=10;y=8;color=3

ghost@id=0;x=11;y=9;direction=left;ismoving=true;speed=300;state=alive;spawn\_x=9;spawn\_y=9;isPanic=false;panicTime=0;ghostId=0

ghost@id=1;x=6;y=9;direction=left;ismoving=true;speed=300;state=alive;spawn\_x=10;spawn\_y=7;isPanic=false;panicTime=0;ghostId=3

ghost@id=2;x=10;y=9;direction=right;ismoving=true;speed=300;state=alive;spawn\_x=10;spawn\_y=9;isPanic=false;panicTime=0;ghostId=2

pacman@id=3;x=5;y=15;direction=left;ismoving=false;speed=300;state=alive;spawn\_x=10;spawn\_y=15;lifes=3;score=50;hero=false;timeHero=0;controllerId=0

Comme expliqué plus haut, la partie modèle est totalement isolée. Il est possible de l’exécuter indépendamment. Les interactions avec le modèle se font grâce aux *listeners* et le *pacman controller*. Les *listeners* permettent de recevoir les mises à jour du modèle et le *controller* permet de choisir la direction du Pac-Man contrôlé. Le jeu utilise souvent le principe de réflexion en java pour identifier la spécialisation des classes en temps réel. Celui-ci est notamment utilisé pour la spécialisation des tuiles (Wall, Floor, Door).

# 3 – Répartition des rôles

Pour la répartition des rôles nous avons choisi de déterminer les tâches à faire avant de nous les répartir.

Il y avait deux grandes parties, l’analyse puis le développement. Deux parties qu’il fallait faire dans l’ordre chronologique, ce qui permet de travailler à deux dessus.

Pour l’analyse nous avons réalisé un diagramme de classes, l’outil le plus important pour la suite, nous l’avons fait ensemble afin de réfléchir à la structure de notre application ensemble pour qu’aucun de nous deux ne subisse le raisonnement de l’autre.

Pour la partie développement il fallait un « squelette » pour notre application, thomas a donc créé la base du modèle et de la vue pour qu’ils puissent fonctionner ensemble tandis que Maxence a créé les différentes classes présentes sur notre diagramme de classes. Lorsque nous avions un Pac-Man capable de se déplacer sur un fond noir Maxence à réaliser tous les spritesheets du Pacman ainsi que des fantômes. Alors que Thomas s’est occuper de les animer côté Vue. A l’inverse pour la carte, Thomas a réalisé la spritesheet et l’affichage des mur sur la carte tandis que Maxence faisait la carte sur un document texte afin de l’interpréter au démarrage de la carte.

Notre application était maintenant une application ou un Pacman pouvait se déplacer dans sur une carte facilement éditable grâce au fichier texte. Il nous manquait donc les fantômes, les Points, l’overlay du jeu puis les extensions. Notre jeu était suffisamment « grand » pour que nous puissions travailler chacun de notre côté sur une fonctionnalité. Maxence à fait la gestion des fantômes en particulier l’ « ia » ou du moins les déplacements des fantômes en fonction de la position du joueur mais aussi en fonction de la génération aléatoire de nombre. Pendant que thomas réalisait un éditeur de carte.

L’affichage des pacgum et super pacgums a été réalisé par Thomas et la gestion des points et des effets des super pacgums par Maxence. Thomas à fait la gestion de l’overlay (variable en fonction du nombre de joueur) puis il à adapter notre jeu pour du multijoueur, avec une partie server et une partie client.

Il ne restait plus que la gestion des morts et les réapparitions où Maxence a réalisé la détection de collisions et le décompte initial et Thomas à gérer toute la partie réapparition et mort du joueur.

# 4 – Limites et problématiques

Nous avons dû faire face à différentes contraintes lors de la réalisation de notre projet. Le modèle MVC à été notre première contrainte, en effet dans les jeux vidéos ce modèle n’est pas le plus optimale et ne permet pas une amélioration continue simple. L’idée de faire un Pacman multijoueur ai née après le début du développement, il a donc fallu adapter notre coder plutôt que de tout recommencer. Pour ne pas impacter le modèle général du design MVC nous avons dû rajouter une couche supplémentaire interfaçant avec la vue et modèle. De ce fait nous conservons une architecture globale respectant le modèle MVC. Le point de vu graphique à été quelque peu contraignant, en effet nous ne sommes pas graphiste et ne disposons pas d’outils pour créer des spritesheets, nous avons donc dû créer nos différentes animations à la main.

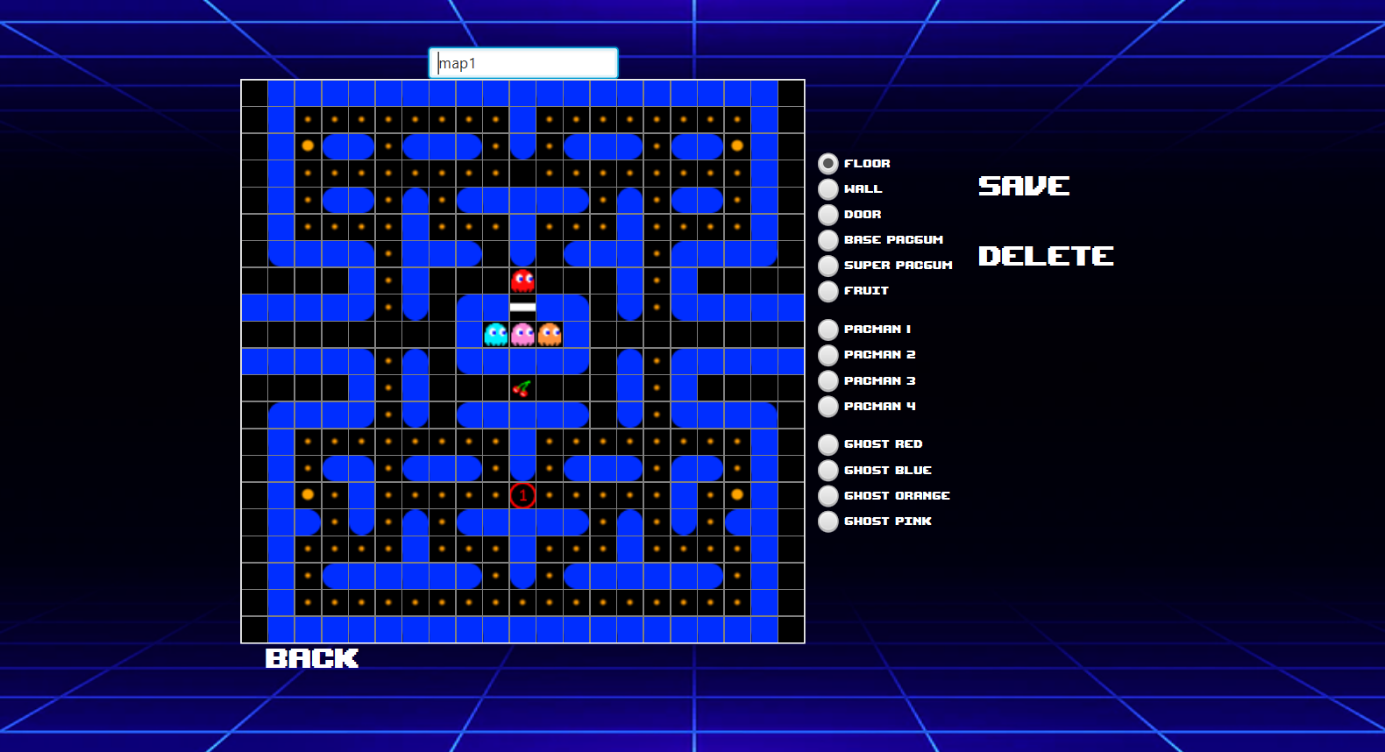
# Conclusion

Ce projet nous a appris à utiliser le modèle MVC de manière ludique, ainsi que la technologie JavaFX. De plus, cela a été l’occasion de manipuler des outils autre que du pure développement : Git, paint.net/Gimp. Grâce à la partie multi-joueurs nous avons pu réfléchir et implémenter une solution de réseau (protocole, tcp).

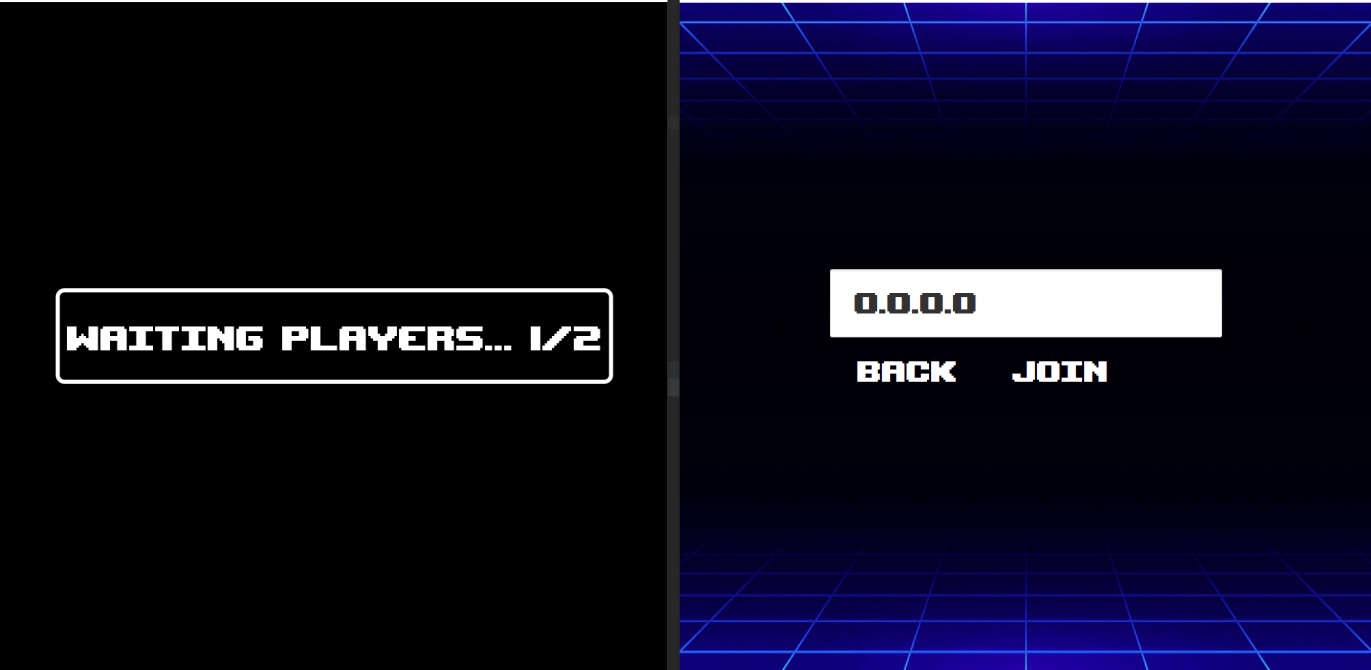
Pour finir, ce projet à été un moyen de mettre en commun nos connaissances grâce au travail d’équipe.

# Annexes

## Annexe 1 Editeur de carte



## Annexe 2 Menu multijoueur



## Annexe 3 Exemple avec player 1 et player 2

