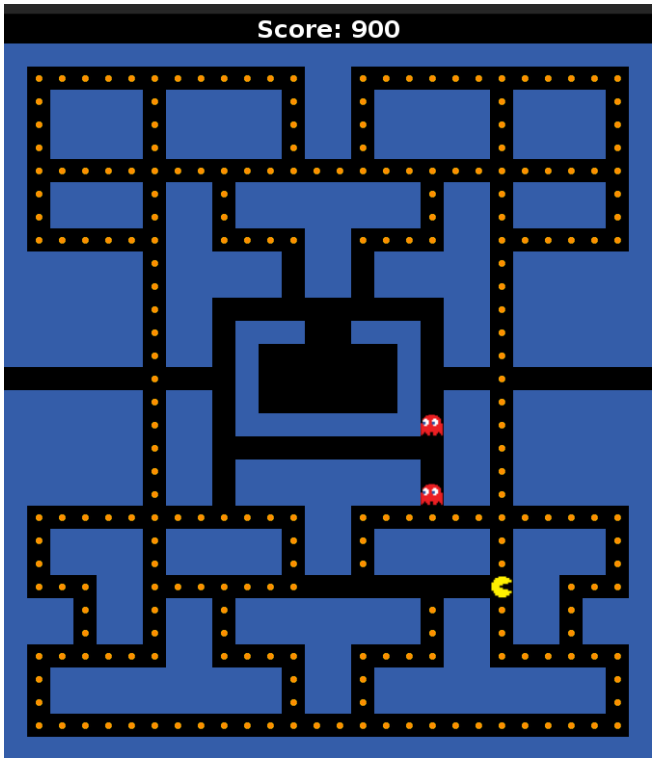
Antoine Poinsot *p1607390*
Kennedy Niamke *p1715528*

Projet Algorithmique Et Programmation Orientée Objet : PACMAN



Outils

Nous avons utilisé le JDK 11 et aucune librairie externe.

Organisation

Nous avons créé un repo Git sur l'instance GitLab de Antoine (https://gitlab.darosior.ninja).
Cela nous a permis de centraliser notre code, nos discussions sur l'architecture et objectifs (issues), ainsi que nos reviews (merge requests).

Nous avons également échangé par mail et téléphone.

Architecture

Nous avons respecté une architecture MVC et le code (dans le dossier « **src/** » depuis la racine du repo) est regroupé dans un package « **pacman** » et structuré comme suit :

- Le dossier « **src/pacman/controller** » regroupant les différentes classes du controleur.
- Le dossier « **src/pacman/model** » regroupant les différentes classes du modèle, la plus grosse partie contenant la logique du jeu.
- Le dosser « **src/pacman/view** » regroupant les différentes classes de la vue. Swing a été utilisé pour l'interface graphique.
- La classe « **src/pacman/Main.java** » qui ne fait qu'instancier le modèle et le controleur.

Modèle

La classe principale du modèle est « **Game.java** », qui implémente la logique de Pacman (« **Pacman.java** ») et des fantômes (« **Ghost.java** ») sur la map (« **pacmanMap.java** »).

Cette dernière est une interface à la grille, représentée comme un tableau d'entiers à deux dimensions utilisé comme une bitmap. Ce choix a été fait car permettait une gestion simple des collisions (le premier bit (en partant du moins signifiant) indique la présence d'un mur, le deuxième d'un point à manger, le troisième la présence d'un fantôme, le quatrième de Pacman).

La classe de Pacman et celle des fantômes héritent de la même classe abstraite (« **Entity.java** ») car partagent des mêmes attributs. Elles gèrent principalement la direction de ces entités, qui peut être modifiée par le controleur pour Pacman, et qui est dictée par un algorithme de recherche de chemin au-travers de la map pour les fantômes (qui sera détaillé plus loin).

La boucle principale de jeu est, comme dans le code initialement donné, démarrée dans un thread. A noter que le « tick » de cette boucle peut être altéré en fonction du temps d'exécution de la logique (notamment la recherche de chemin pour les fantômes) afin que le temps d'arrêt soit tout le temps de la même longueur.

Vue

La classe principale de la vue est « **pacmanView.java** », qui hérite d'une Jframe et met à jour le contenu d'une grille d'éléments graphique en consultant le modèle (ou plus exactement, en réagissant à des évènements).

La grille contient des instances de la classes « **rotatableJLabel.java** », une classe héritant de Jlabel qui permet d'orienter l'image de fond afin de pouvoir tourner le rendu graphique des entités en fonction de la direction qu'elles prennent.

Nous avons choisi des images « old school », à la fois pour des goûts esthétiques et parce qu'ils s'inscrivaient bien dans le gameplay un peu ralenti par les performances de Swing (et non du modèle!).

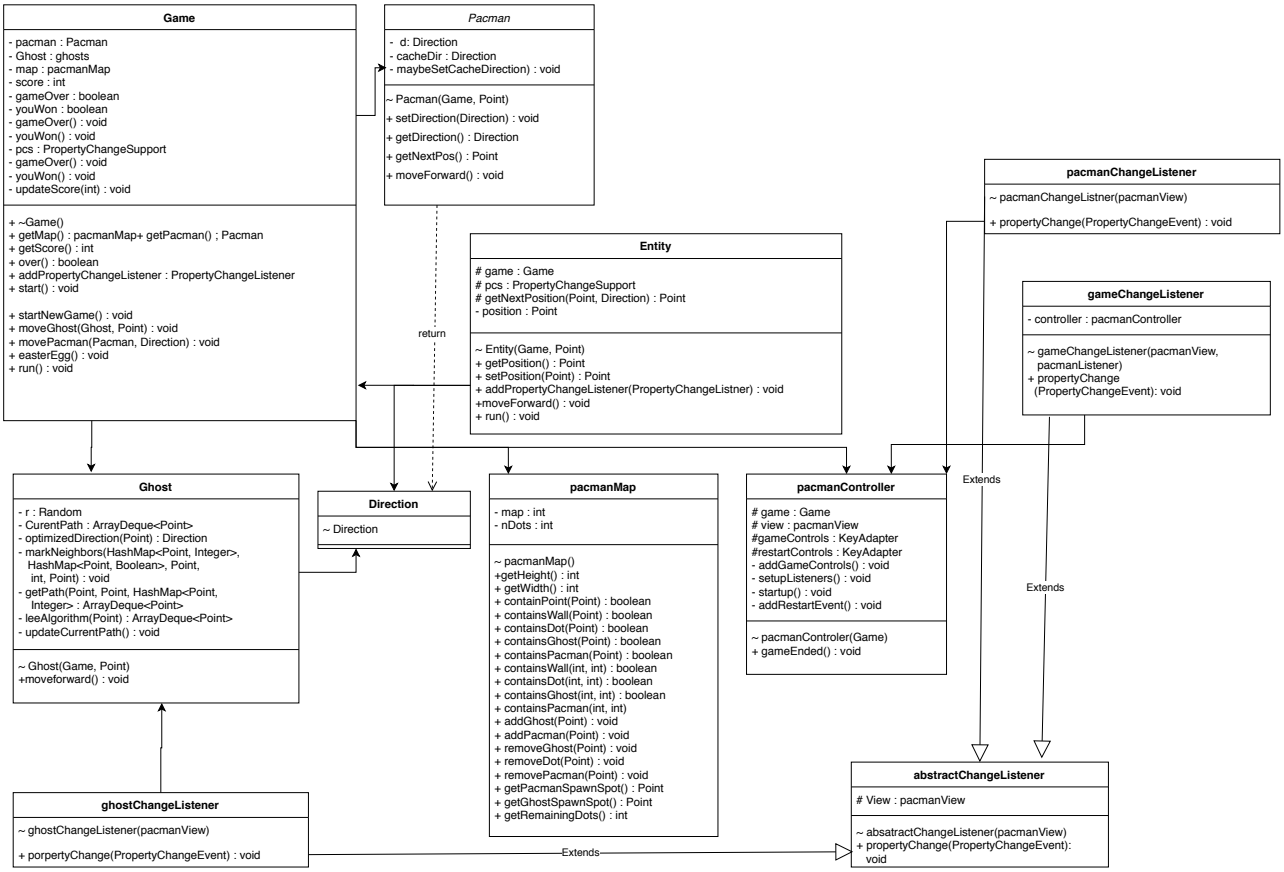
Contrôleur

La classe principale du contrôleur est « **pacmanController.java** », qui ajoute et gère les évènements de l'utilisateur et du modèle.

Ces évènements venant du modèle sont gérés par des classes héritants d'une classe abstraite « **abstractChangeListener.java** », qui hérite elle-même de « **PropertyChangeListener.java** ». Il y en a une pour la logique du jeu (« **gameChangeListener.java** »), une pour Pacman (« **pacmanChangeListener.java** ») et une pour les fantômes (« **ghostChangeListener.java** »).

Cette architecture a été préférée à celle utilisée dans le code fourni initialement, qui utilisait des **Observable**, car ces derniers étaient obsolètes depuis le JDK9 (2017 https://en.wikipedia.org/wiki/Java_version_history) et parce que la gestion assez naturelle des évènements permettait de mieux segmenter les parties modèle, vue, et contrôleur.
Aussi parce que c'est ce que la doc recommendait, pour être tout à fait honnête.

Diagramme des classes



Fonctionnalités et extensions

Modèle

- **Labyrinthe original** : de toutes les sources d'informations sur Pacman que nous avons consultées cette disposition des murs et points semblait être la plus idiomatique. Nous l'avons reproduite à l'identique (hormis la grille pour les fantômes).
- **Fantômes multiples** : simple avec une grille type bitmap, les chevauchements ne posent pas problème.
- **Score** : selon ces mêmes sources, une boule semble valoir 10 points.
- **Mémoire de direction** : une direction pour Pacman peut être pré-choisie.
- **Lissage** du temps d'arrêt dans la boucle principale.
- Effet de « **wraparound** ».
- **Fantômes intelligents** : un algorithme de recherche de chemin (très documenté dans le code, dans la classe « **Ghost.java** ») a été utilisé pour modifier la direction des fantômes en direction de Pacman. Il s'agit de l'algorithme de Lee (https://en.wikipedia.org/wiki/Lee_algorithm) qui présente une forte précision mais également une forte complexité théorique. Le temps d'exécution est toutefois très acceptable en pratique (et sur un graphe aussi petit) et complètement compensé par le lissage opéré dans la boucle principale.
    - Le chemin n'est mis à jour que tous les 5 tours (randomisé, donc en moyenne) de boucle afin de laisser sa chance au joueur.
    - Un effet intéressant de l'algorithme est que si Pacman bouge les fantômes ont tendance à le prendre en sandwich.

Vue

- **Image pivotante** : les images des fantômes et de Pacman tournent en fonction de leur direction.
- Ecrans de **Game Over** et de victoire.

Contrôleur

- **Redémarrage** d'une nouvelle partie.
- Les mises à jour sont **évènementielles**.
- Un **easter egg** est présent, au cas où les enseignants n'arriveraient pas à accéder à l'écran de victoire :-P.

Captures d'écran

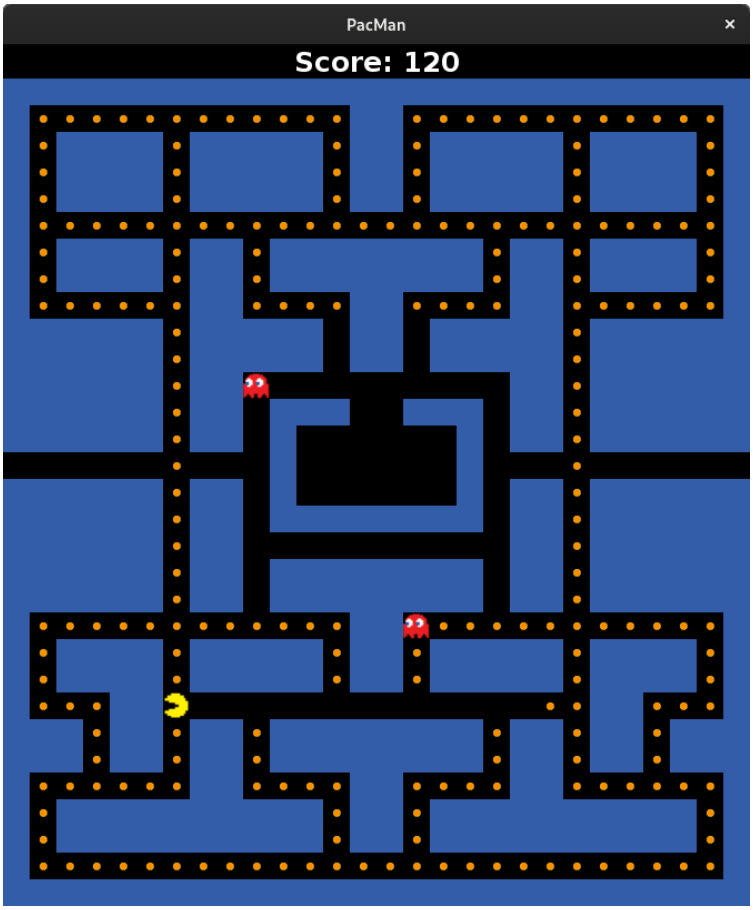


Figure 1: Les fantômes foncent sur Pacman, chacun par un chemin différent : ils vont bientôt le prendre en sandwich !

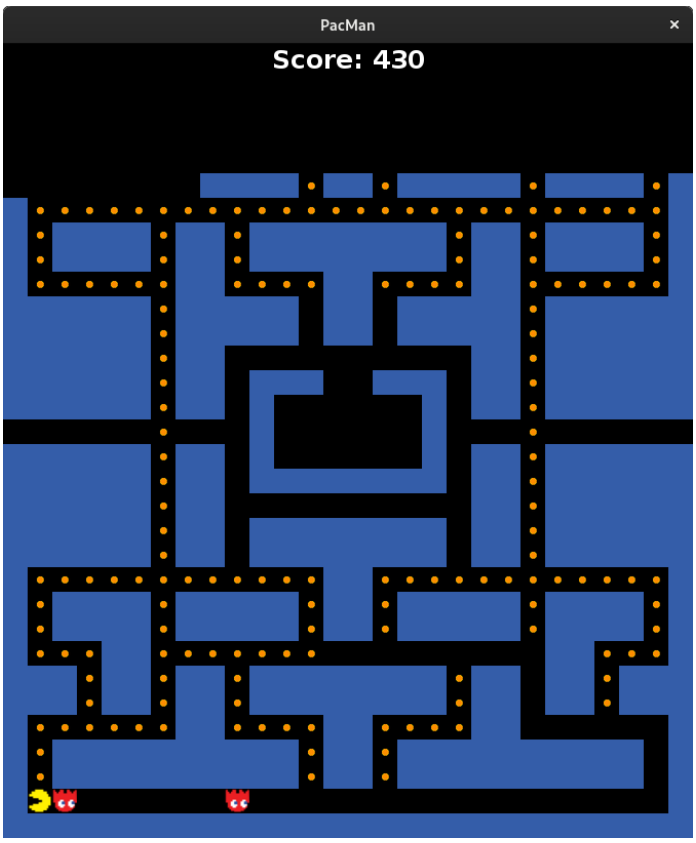


Figure 2: Un fantôme vient de rencontrer Pacman, l'écran commence à se noircir....

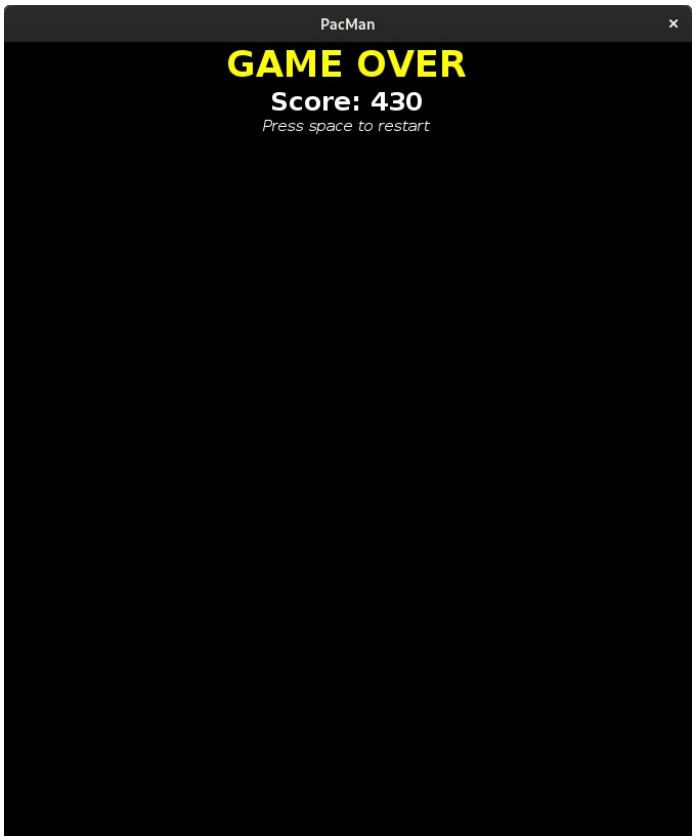


Figure 3: Le fantôme a eu Pacman, Game Over !

Améliorations possibles

Nous aurions aimé pouvoir tout implémenter, mais voici une liste de certaines caractéristiques notables que nous avions en tête et n'avons pas implémenté :

- Ajustement de la difficulté (baisser la mise à jour du chemin).
- Donner plusieurs vies à Pacman.
- Ajouter les « energizer »

Contributions respectives

Kennedy a fait la plus grande partie du diagramme de classes (toutefois Antoine a fait la partie des listeners pour le contrôleur), et Antoine la plus grande partie du code (toutefois Kennedy a fait la fonctionnalité de redémarrage d'une partie).
Pour autant, ces indications ne concernent que le premier jet et ne prennent pas en compte le travail collaboratif de review des nouvelles fonctionnalités ajoutées et ne donnent par conséquent pas une bonne impression du travail fourni par chacun (Antoine auraient fait tout le code et Kennedy tout le diagramme, ce n'est pas le cas dans les faits).