INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Projet Informatique **Pac-Man**

Mathieu Reslou, Alice Launet, Grégoire De Lambertye

Table des matières

1 Présentation du projet

Pour ce projet notre choix s'est assez rapidement tourné vers les jeux vidéos. Il nous fallait trouver un projet motivant, et dont la réalisation était techniquement intéressante. Nous avons choisi de réaliser un Pac-Man, jeu iconique et populaire, car c'est un jeu connu de tous, et suffisamment complexe. L'objectif de notre projet était de réaliser le « Pac Man » le plus similaire possible au premier jeu d'arcade. C'est-à-dire avec un plateau de jeu constitué d'un labyrinthe dans lequel évoluent le « Pac Man » et 4 fantômes. Le but du jeu est d'attraper toutes les billes blanches et jaunes du plateau sans rencontrer les fantômes qui se déplacent dans le labyrinthe. Pour réaliser ce programme nous devions donc concevoir :

- Une partie graphique riche, avec le dessin du labyrinthe et des différents personnages.
- L'intégration d'une bande son, pour rendre le jeu vivant et attractif.
- La création des algorithmes de déplacement de chacun des fantômes, et des interactions entre les personnages et leur environnement.
- Un menu, permettant d'apporter du réalisme et d'accéder à plusieurs fonctionnalités (son, couleur du Pac-Man, scores ...).

Nous avons ainsi pu mettre en pratique différentes notions de nos cours d'informatique. Notamment les IHM, l'orienté objet et l'héritage, et enfin le parcours de graphe et l'optimisation. Bien sûr, il a aussi été très instructif de se documenter sur de nouvelles techniques, pour les intégrer au projet, par exemple pour l'ajout des bandes son ou l'utilisation de nouveaux Listeners.

2 Présentation de l'algorithme

2.1 Introduction

Pour réaliser ce projet nous avons utilisé le langage de programmation java étudié en cours. Nous avons organisé notre programme en 4 packages distincts :

- un package IHM contenant l'affichage et les écouteurs.
- un package personnage pour le Pac-Man et les fantômes.
- un package chemin qui contient les classes pour le calcul des trajectoires.
- un package musique qui permet de gérer les bruits et musiques du jeu.

La classe **Labyrinthe** permet de relier ces différentes parties et les articulent pendant une partie de jeu. Enfin la classe **FenetreMenu** sert de menu, elle permet d'accéder à toutes les fonctionnalités du jeu (options de son et d'affichage, scores, crédits et lancement d'une partie). Elle se compose de :

- un bouton permettant de lancer le jeu.
- un bouton score, qui affiche le score de la dernière partie, et le meilleur score des parties réalisées sur la session en cours.
- un menu option qui permet de changer la couleur du Pac-Man et de régler le son de la musique (on/off et volume).
- un bouton crédits, affichant des informations importantes concernant la création du jeu.
- un bouton pour quitter le menu et le jeu.

2.2 Diagramme UML

`https://drive.google.com/file/d/1rtNX9y0BTWwuIkLyn5ZSH0dofUffMlB0/view?usp=sharing`

Notre programme est structuré en 4 packages : ihm, chemin, personnage et musique. Ces 4 packages s'articulent autour des classes **Labyrinthe** et **FenetreMenu**. La classe permettant de lancer le tout est

JeuPac-Man. Notre diagramme reprend cette organisation et montre les différentes interactions entre les classes, tout en respectant les sous-groupes auquel elles appartiennent.

2.3 Déroulement d'une partie

Après avoir exécuté **JeuPac-Man** et fait ses choix dans les options du menu (et pourquoi pas s'être renseigné dans les crédits) le bouton "jouer" permet de démarrer une partie. **FenetreMenu** va alors créer un nouveau labyrinthe et lancer la méthode *run()* qui contient l'algorithme de jeu.

Lors de son instanciation, un labyrinthe créé un affichage et tous les personnages du jeu. La classe **Affichage** ouvre une fenêtre et lui associe un KeyListener. **Affichage** contient un attribut de la classe **Map**, qui contient le coeur de l'IHM. Via un "paint" elle transforme un tableau de chiffres en dessin.

La méthode *run()* qui est le squelette du jeu commence par jouer une musique d'intro. Puis, dans une boucle "while", les personnages se déplacent, interagissent entre eux et avec le plateau.

— Pac-Man se déplace en récupérant la dernière touche reçue par le KeyListener. La méthode *déplacementPersonnage()* simule la case où le personnage souhaite se rendre puis exécute un déplacement en fonction de ce qui se trouve dans cette case. Si c'est un élément infranchissable, comme un mur, le personnage va persévérer dans son mouvement précédent.

— Les fantômes héritent de la classe **Fantome**, leur déplacement varie en fonction des phases de jeu. Celles-ci correspondent aux comportements que les fantômes adoptent en fonction de Pac-Man.

En temps normal, les fantômes ont des comportement différents :

— Le fantôme cyan se déplace de manière aléatoire partout sur la carte. Pour qu'il ait un déplacement fluide, il ne change de direction qu'aux intersections et ne fait jamais demi-tour.

— Le fantôme rouge cherche toujour le chemin le plus court entre Pac-Man et lui. Pour trouver son chemin il se sert des méthodes du package "chemin".

— Le fantôme orange à un comportement similaire au fantôme rouge mais lorsque le joueur tape une certaine combinaison de direction, il va chercher à fuir Pac-Man pendant quelques déplacements. Cela permet de le rendre moins prévisible.

— Le Fantôme rose est considéré comme le plus "intelligent" : il cherche à se placer sur la case où Pac-Man se rend. C'est à dire qu'il cherche le chemin le plus court entre sa position et le premier mur que devrais rencontrer Pac-Man s'il continue dans son déplacement. Il se place ainsi en embuscade.

Lorsque Pac-Man mange une grosse bille, les fantômes deviennent bleus. Ils tentent de fuir Pac-Man, qui a maintenant le pouvoir de les manger. Pour ce faire, ils recherchent le chemin le plus court jusqu'à Pac-Man, et prennent systématiquement une autre direction que celle trouvée (quand cela leur est possible). Pour un déplacement plus fluide, ils ne calculent cette direction que tous les 4 déplacements. Au bout de quelques instants, les fantômes deviennent blancs pour signaler au joueur que son pouvoir va bientôt s'arrêter.

Si le joueur rencontre un fantôme, en temps normal il perd une vie et tous les personnages sont téléportés à leur position initiale. Si Pac-Man avait le pouvoir de manger les fantômes, le fantôme rencontré rejoint sa position initiale par le plus court chemin et on ne voit plus que ses yeux. Il n'est plus possible d'interagir avec lui.

Le niveau se termine lorsque Pac-Man a mangé toute les billes du plateau. Les personnages sont alors téléportés à leur emplacement initial et le plateau se recouvre à nouveau de billes. D'un niveau à l'autre, la vitesse de déplacement des personnages augmente de 20%. La partie se termine lorsque le joueur à été mangé 3 fois par un fantôme et n'a plus de vie.

ZOOM SUR Le lissage de l'affichage : Pour avoir une expérience de jeu plus fluide, il fallait dessiner les personnages à plusieurs endroits rapidement et donner ainsi une impression de mouvement. Or nous ne disposions que d'un tableau 2D. L'idée que nous avons eu a été de dessiner un personnage en amont de sa position. Imaginons que Pac-Man soit en position [20,23] sur notre grille. Pour signaler un personnage sur le plateau, la case qui correspond à sa position contient un code dont le dernier chiffre définit son "indice de déplacement" ou la direction dans laquelle il se déplace. Grâce à cet indice nous pouvons deviner la case sur laquelle se trouvait le personnage avant de se déplacer en [20,23]. S'il venait du nord, nous le dessinerons sur la case supérieure et regardant vers le bas puis un petit peu plus bas et ainsi de suite. Pour chaque tableau envoyé à l'affichage, la fenêtre va se mettre à jour et dessiner 6 fois tout le plateau. Le deuxième problème lié au lissage de l'affichage à été de gérer la différence de vitesse relative entre Pac-Man et les fantômes, qui sont deux fois plus lents que ce dernier. pour cela dans la boucle "while" de la méthode *run()* de **Labyrinthe**, Pac-Man change de position 2 fois alors que les fantômes ne se déplacent que d'une case. Afin d'éviter que leur déplacement soient saccadés, un indice signale à la méthode *paintComponent()* si les 3 premières positions des fantômes doivent être dessinées ou les trois dernières.

ZOOM SUR le son : Pour réaliser toute la bande son du jeu, nous avons décidé d'utiliser une classe trouvée sur internet : **SimpleAudioPlayer** ORACLE et AFFILIATES, *Java™ Platform Standard Ed. 8*. Cette dernière, nous permet d'intégrer des "clips" que l'on peut jouer et arrêter à notre guise. Il a alors fallu placer les bandes sons aux bons endroits et les activer aux bons moments en utilisant les différentes méthodes de **Labyrinthe**. La bande son est répartie sur deux audio players : un pour la musique de fond et un pour tout le reste. La complexité de la mise en place de ces audios réside dans le fait que l'audioPlayer ne peut contenir qu'une seule musique en même temps. Sinon les musiques se superposent et on perd le contrôle sur la première musique. Nous devons donc lancer une musique, puis l'arrêter et enfin recréer un audioPlayer du même nom avec une musique différente et utiliser la méthode *play()* pour la jouer. Ensuite, nous avons ajouté des méthodes dans la classe **SimpleAudioPlayer**, pour pouvoir modifier le volume et pouvoir couper le son. Ces méthodes interagissent avec le contrôle du son de l'ordinateur.

ZOOM SUR la recherche du chemin le plus court : Pour chercher le chemin le plus court entre deux points (noeuds) de notre labyrinthe, nous nous sommes inspirés de l'algorithme A*. Le principe est le suivant :
On se place à un noeud initial (coût nul par défaut). L'algorithme va chercher, parmi les noeuds voisins du noeud considérés, celui à l'heuristique (somme du coût propre* au noeud, et de la distance qui le sépare du but à atteindre) la plus basse. Pour ce faire, il aura :

- placé chaque noeud voisin du noeud considéré dans la liste nommée ouvert (représentant l'espace accessible à partir du noeud).
- trié ensuite cette liste (grâce à méthode compareTo() de l'interface comparable), de telle sorte à avoir le noeud à l'heuristique la plus basse en première position.
- choisi le premier noeud

Il retire ensuite ce premier noeud de la liste Ouvert, l'ajoute à une deuxième liste nommée Chemin et recommence ce processus à partir du nouveau noeud jusqu'à ce que le premier noeud soit le noeud d'arrivée. L'algorithme maintient aussi la liste Fermé, correspondant aux noeuds déjà vérifiés, pour éviter de les considérer à nouveau. À la fin, c'est la liste Chemin qui contient le chemin recherché. (*le coût propre de chaque noeud correspond au nombre de case minimum le séparant du point de départ, en suivant les chemins possible.)

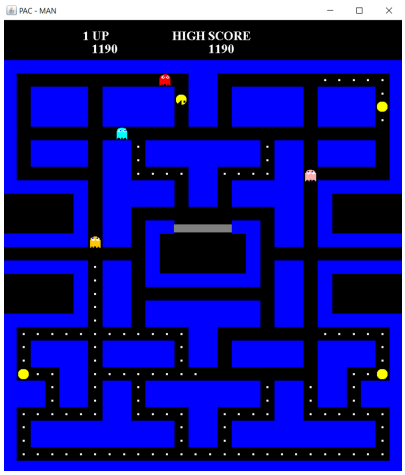

FIGURE 1 – Affichage du jeu

3 Analyse des forces et des faiblesses

Forces La partie IHM de notre projet est sa principale force. Nous avons cherché à ce que l'affichage soit le plus ressemblant possible à l'affichage du premier Pac-Man. De plus, nous l'avons rendu fluide, pour que l'expérience de jeu soit la plus agréable possible. Une chose très intéressantes que nous avons retenu dans ce projet est aussi l'incorporation de Listener dans notre programme. Nous en avons intégré plusieurs (exemples : KeyListener/WindowListener/ ChangeListener/ActionListener), de telle sorte à rendre notre programme intéractif. Encore une fois, notre projet a gagner en fluidité et en précision. Nous avons aussi fait des recherches quant à l'algorithme de recherche de chemin, pour trouver celui qui serait le mieux adapté au contexte de notre jeu.

Faiblesses L'audio player que nous avons utilisé a nécessité un code complexe pour réaliser des tâches simples. Nous aurions voulu avoir un audio Player unique et plus simple pour contrôler tout l'audio, mais la librairie intégrée à JAVA nous a limité. De plus, l'utilisation des packages est assez compliquée dans Geany. Il nous faut maintenant faire attention, à la première compilation (après téléchargement d'une nouvelle version par exemple), d'avoir bien supprimé tous les fichiers .class.

4 Avis personnel sur ce projet

Mathieu : J'ai bien aimé réaliser ce projet car j'ai enfin pu appliquer ce que l'on avait vu en cours, et ce, de manière plus ludique. Ensuite, j'ai trouvé intéressant de travailler en groupe, pour l'aspect méthode de travail et organisation. De plus, j'ai vraiment apprécié d'apprendre à utiliser de nouvelles librairies ou de nouveaux objets en cherchant sur des forums ou la javadoc Oracle JAVA™, *How to Write Window Listeners*. Finalement, je suis vraiment content d'avoir fait quelque chose d'abouti et cool.

Grégoire : J'ai trouvé ce projet très intéressant, la partie technique m'a permis de d'appliquer et de découvrir de nouvelles fonctionnalités de java dans un contexte ludique. Travailler en petit groupe nous a

permis d'apprendre à nous organiser, et à nous répartir le travail à faire. L'organisation en "mode projet" nous a permis de répartir au mieux le travail en fonction des aptitudes et des talents de chacun.

Alice : Réaliser ce projet m'a permis d'appliquer de manière concrète et agréable toutes les notions étudiées cette année. Cela m'a réellement apporté, dans le sens où je n'avais jamais eu à mener dans cette matière une réflexion sur un si grand ensemble. Mêler l'utile à l'agréable est d'autant plus motivant. En outre, mener ce projet en équipe est aussi très instructif. Cela exige l'implication de chacun, la mise en place d'une division précise des tâches, et demande une écoute attentive au sein de l'équipe. Toutefois, le terme de "mini" projet employé ne reflète pas l'ampleur notre investissement. En bref, ce projet m'a permis de m'améliorer et de réellement apprécier de faire de l'informatique.

Bibliographie

INSA Lyon (FIMI), Ressources informatiques. *[17-18] cours 4 : dessin (v2)*. URL : https://moodle.insa-lyon.fr/course/view.php?id=2544.

— *[17-18] cours 4 : dessiner*. URL : https://moodle.insa-lyon.fr/course/view.php?id=2544.

Java™, ORACLE. *The Java™ tutorials [en ligne]*. URL : https://docs.oracle.com/javase/tutorial/java/package/index.html.

Java™, Oracle. *How to Write Window Listeners*. URL : https://docs.oracle.com/javase/tutorial/uiswing/events/windowlistener.html.

— *How to Write Window Listeners*. URL : https://docs.oracle.com/javase/tutorial/uiswing/events/windowlistener.html.

Oracle et its affiliates. *Java™ Platform Standard Ed. 8*. URL : https://moodle.insa-lyon.fr/pluginfile.php/55398/mod_resource/content/1/Jdk8-api/index.html.

TimSC. *A* search algorithm [en ligne]*. URL : https://rosettacode.org/wiki/A*_search_algorithm#Java.