



# PROJET JAVA

## Sujet 1



GAMO Christian | SOUCCOUCHETTY Darlène | ZHENG Lisa-Marie

L3 MIAGE - Année 2021/2022



# PRÉSENTATION DU PROJET

## Pourquoi avoir réalisé ce projet ?

Dans le cadre de notre enseignement en programmation orientée objet, nous avions pour objectif de réaliser une application en langage Java, en trinôme. Parmi les deux thématiques qui nous furent proposées, nous avons choisi la première : le jeu vidéo.

Cette thématique nous a particulièrement inspirée tous les trois, car nous sommes à l'unanimité adeptes de jeux vidéos, en particulier de style rétro. C'est ainsi que nous avons décidé de réaliser notre propre version d'un jeu d'arcade emblématique des années 80-90 : Pac-Man.

## Avec quels moyens avons-nous réalisé ce projet ?

Pour réaliser ce projet, nous avons réinvesti de nombreuses notions et acquis de notre cours de programmation orientée objet (principe d'encapsulation, polymorphisme, héritage etc...), dont nous allons parler plus en profondeur au niveau de la partie relative à la structure du code. Néanmoins, certaines précisions quant à nos choix pour la conception et la programmation de l'application ont été apportés au niveau de notre documentation JavaDoc.

Concernant l'aspect pratique de la programmation de l'application, nous avons utilisé l'environnement de développement Eclipse, ainsi qu'une extension disponible sur le Marketplace de l'IDE : Saros, qui nous a permis de coder en mode collaboratif sur une même version de notre projet. Dans le cas où nous codions pas simultanément, nous utilisions Git pour pouvoir échanger nos codes en tirant profit du principe des branches logicielles (commit/push/merge) offert par l'outil.

Enfin, pour rendre notre projet plus interactif et immersif, les sons originaux du jeu Pac-Man et une interface graphique ont été implémentés. Nous avons donc utilisé en complément d'autres librairies non disponibles dans la bibliothèque standard Java à savoir javax.swing et javax.sound.

## Quel est le principe de notre application ?

Au sein de notre version du jeu Pac-man, le joueur incarne le personnage jaune et rond Pac-man et contrôle ses déplacements à travers le labyrinthe pour que ce dernier puisse manger toutes les gommes disponibles sur le plateau de jeu. Toutefois, durant toute la durée de la partie, Pac-Man se fait poursuivre par 4 fantômes : Blinky (rouge) ; Inky (bleu) ; Pinky (rose) ; Clyde (orange) qui sont ses ennemis et dont le but est d'éliminer Pac-Man.

Grâce aux 4 super gommes réparties à travers le labyrinthe Pac-Man peut cependant manger ses ennemis et les neutraliser pour une durée déterminée. Si Pac-Man touche un fantôme en dehors du mode super forme (sans avoir préalablement mangé une super gomme pour effrayer les fantômes), ce dernier perd une vie. Si il perd ses 3 vies en touchant des fantômes à 3 reprises avant d'avoir mangé toutes les gommes disponibles sur le plateau, ce dernier meurt et la partie est terminée.

Si Pac-Man arrive à manger toutes les gommes disponibles sur le plateau, le nombre de vies dont il dispose est réinitialisé à 3, puis le jeu recommence mais la difficulté du jeu augmente puisqu'à chaque victoire, les fantômes deviennent plus rapides.

En somme , le principe de notre jeu reprend les mêmes codes que la version classique du jeu. Les seuls éléments qui diffèrent sont l'absence de fruits rapportant des points bonus et l'absence de niveaux supplémentaires.

Le seul moyen pour que le niveau soit différent est de modifier le fichier niveau.txt, lu par le programme, en changeant ses valeurs selon la forme du labyrinthe désirée.



# STRUCTURE DU CODE

Pour la structure du jeu, nous avons fait en sorte de séparer le côté "front-end" de l'application du côté "Back-end" de l'application. Ainsi, chaque package contenant les différentes classes de notre jeu joue un rôle bien délimité au sein de notre application.

Les packages permettent justement de contrôler en partie la visibilité que chaque classe, méthode et variables de notre projet ont par rapport d'autres, ce qui permet à notre code de respecter une certaine logique en termes d'architecture logicielle, donc en somme les principes de conception SOLID.

D'autres conventions de programmation relatives à la structure de notre code ont été scrupuleusement respectées pour faciliter la maintenance du code notamment le fait de nommer nos classes avec une lettre majuscule, et nos packages et méthodes en minuscules, de ne pas directement créer nos classe au niveau du package source par défaut etc...

À la page suivante sont également détaillées les différentes contraintes que nous avons respectées, et qui en réalité ont été assez indispensables afin de construire une architecture logicielle compréhensible, logique et surtout adaptées à d'éventuelles modifications sans impacter la structure de nos classes et méthodes existantes.

Pour revenir au packages de notre projet, qui délimitent pour rappel les différentes parties de notre architecture logicielle, ces derniers sont au nombre de 4. De manière assez schématique nous avons :

- Le package jeu pacman affichage : qui contient toutes les classes et méthodes permettant de gérer tout ce qui est relatif à la récupération des fichiers sprites pour l'affichage du labyrinthe et des personnages du jeu.
- Le package jeu pacman entites : qui contient toutes les classes et méthodes permettant de modéliser dans notre application les différents personnages du jeu Pacman ainsi que leurs caractéristiques (algorithmes de déplacements des personnages).
- Le package jeu pacman labyrinthe : qui contient toutes les classes et méthodes permettant de modéliser le labyrinthe qui constitue en quelque sorte notre plateau de jeu et la "plateforme" d'interaction de nos personnages.
- Le package jeu pacman application : qui contient toutes les classes et méthodes qui sont au cœur de notre application puisque ce package permet de faire le lien avec tous les autres packages et contient les éléments clés pour pouvoir faire fonctionner le jeu notamment l'interface graphique, les sons, le classement ainsi que la classe jeu.

Nous avons également d'autres dossiers annexes contenant des éléments nécessaires au bon fonctionnement de notre jeu ou nécessaires pour tester la validité de nos méthodes :

- Le dossier leaderboard : qui contient un fichier texte avec la sauvegarde du classement des dix meilleurs joueurs du jeu.
- Le dossier niveaux : qui contient deux exemples de fichiers textes permettant de charger le labyrinthe du jeu.
- Le dossier polices : qui contient le fichier .ttf contenant la police par défaut des différents textes écrits qui apparaissent sur l'interface graphique.
- Le dossier sons : qui contient les différents fichiers .wav contenant les sons originaux issus du jeu Pac-Man que nous utilisons également dans notre version du jeu.
- Le dossier JUnit : qui contient les fichiers tests avec lesquels nous avons effectué nos tests pour vérifier le bon fonctionnement et la validité de nos méthodes.

Pour en savoir plus sur la structure de notre code, nous avons également inclus à notre archive un diagramme UML pour pouvoir visualiser la structure de notre projet.



# IMPLEMENTATION DES CONTRAINTES

## 1. Principe d'encapsulation, getters/setters.

Afin de respecter le principe d'encapsulation, tous les attributs de nos classes sont privés et ne sont accessibles qu'à travers des setters et des getters. Nous avons également fait le choix de définir certaines classes comme étant privées même si usuellement très peu de classes sont privées, dans le cas où leur utilisation n'était que restreinte à une classe en elle-même.

## 2. Héritage, interfaces, classes abstraites, polymorphisme.

Concernant l'héritage, cette notion nous a été particulièrement utile dans le cadre de notre projet, notamment en ce qui concerne nos personnages. Nous avons appliqué le principe de polymorphisme, par exemple lorsque chaque fantôme adopte un comportement différent par rapport à sa classe mère fantôme. Nous avons également fait appel à l'héritage pour la classe SpriteAnime qui est par extension un Sprite dont le comportement a été modifié.

Du point de vue des classes abstraites, ces dernières nous ont été utiles du point de vue des fantômes également car il n'est pas possible d'instancier un fantôme ou une entité, il est simplement possible d'instancier individuellement chaque personnage, ce qui est assez logique du point de vue de notre architecture logicielle.

Nous n'avons à proprement dit pas utilisé d'interfaces pour notre projet, nous avons seulement implémenté des interfaces déjà existantes au sein de librairies Java comme l'interface KeyListener, permettant de gérer les interactions clavier durant le jeu.

## 3. Enumérations, collections et Maps.

Concernant les énumérations, il y a les classes Direction, Mode et Case:

- L'énumérateur Direction est commun à toutes les entités du jeu, il permet au jeu de savoir dans quelles directions les entités se déplacent. Il est contenu dans la classe Entite.
- L'énumérateur Mode permet aux fantômes de changer de mode selon les événements du jeu. Ils peuvent être effrayés, en chasse, en dispersion...
- L'énumérateur Case permet l'implémentation du Labyrinthe et contient les différents objets pouvant se trouver dans une Case, par exemple une gomme ou un mur. Elle se trouve dans le package jeu\_pacman\_entite.

## 4. Design patterns.

Dans notre projet nous avons pu implémenter le design pattern Decorator avec la classe abstraite Entite, qui sera hérité par les classes Fantome et Pacman. La classe Fantôme est également une classe abstraite qui sera hérité par les différents types de fantôme. Chaque héritage permet de différencier les différentes entités et d'enrichir leurs actions.

## 5. Documentation JavaDoc, annotations et gestions des tests.

Pour réaliser la documentation JavaDoc, nous avons généré un fichier.html comportant différentes annotations permettant de mieux comprendre le fonctionnement de nos méthodes. Des fichiers test permettent également de tester la validité de nos méthodes.

## 6. Exceptions, lecture/écriture/sauvegarde de fichiers.

En ce qui concerne les exceptions, nous avons utilisé de nombreux blocs try/catch pour vérifier la validité de certaines informations nécessaires au bon fonctionnement de notre code. Nous pouvons par exemple citer le cas de du bloc try/catch au sein de la classe Audio.java qui nous permet de vérifier si le chemin absolu d'un fichier est correct, auquel cas le son fonctionne, sinon une exception est levée.

Pour la lecture, écriture et sauvegarde de fichiers nous avons fait appel à ces méthodes à plusieurs reprises pour des types de documents variés (fichiers.txt pour le leaderboard et les maps du labyrinthe, .ttf pour les polices, .wav pour les sons, et .png pour les sprites). L'aspect relatif à la sauvegarde de fichiers se trouve dans la méthode sauvegarderLeaderboard() au sein de la classe Leaderboard.java ou nous avons codé une méthode permettant de sauvegarder et de charger le fichier leaderboard.txt.

## 7. IHM : console du terminal, interface graphique, sons, gestion des entrées clavier.

L'IHM est entièrement gérée par la classe interface graphique qui prend en charge les entrées clavier en console ainsi que l'affichage de l'interface graphique et des sons du jeu. Pour se faire, nous avons utilisé les scanners ainsi que les librairies javax.sound et javax.swing afin de rendre notre application plus interactive, bien que cet aspect de l'application fut optionnel.



# CONCLUSION



En guise de conclusion, la difficulté de ce projet fut modérée si l'on s'en tenait à réaliser un projet simple, bien structuré et conforme aux consignes indiquées sur le sujet. Toutefois les opinions sont assez variable à ce propos puisque chacun d'entre nous a commencé la programmation orientée objet (Java) à un moment différent.

Le fait d'avoir privilégié les trinômes et binômes fut donc une très bonne idée selon nous pour travailler avec des personnes ayant chacun un niveau différent pour pouvoir s'entraider et progresser.

Ceux qui ont commencé cette année la programmation en Java ont donc été logiquement désavantagés pour ce projet, mais dans l'ensemble, nous trouvons qu'il fut cohérent avec les cours et TD/TP que nous avons pu suivre.

Ce qui fut d'autant plus intéressant et formateur pour nous, fut de mettre en pratique dans le cadre d'un véritable projet ces connaissances que nous avons pu acquérir dans le cadre de cet enseignement, et cela a constitué un bon entraînement en vue des examens.

Grâce à une bonne répartition des tâches et à l'utilisation d'outils de travail collaboratifs comme Git ou Saros, nous n'avons pas rencontré de grandes difficultés dans la réalisation de notre jeu. Hormis le fait que nous ayons au départ longuement hésité sur le jeu que nous allions coder, sur la structure de notre code (packages, classes etc...) et la difficulté à prendre en main les librairies javax.swing et javax.sound que nous avons du appréhender en autodidacte, le reste fut d'une difficulté normale.

D'autre part, les cours, le debugger, les points d'arrêts, les classes test et les ressources dont nous disposions sur internet en cas d'erreurs comme StackOverFlow nous ont été d'une grande utilité lorsque nous avions des erreurs à la compilation et exécution du code.

Sources externes au cours utilisées (pour l'implémentation de l'interface et des sons du jeu) :

<https://www.jmdoudoux.fr/java/dej/chap-swing.htm>  
[https://koor.fr/Java/TutorialSwing/first\\_application.wp](https://koor.fr/Java/TutorialSwing/first_application.wp)  
<https://koor.fr/Java/API/javax/sound/sampled/AudioSystem.wp>