



UNIVERSITÉ DES  
**MASCAREIGNES**

SAVOIR, C'EST POUVOIR

**Université des Mascareignes  
Faculté des Technologies de l'Information et de la Communication  
Département d'Informatique Appliquée  
Programmation Orientée Objet**

**Soumis à M. Khadimoullah Ramoth**

**26/02/24**

**Vibahakarsingh Beeharry, [yhbeehary@student.udm.ac.mu](mailto:yhbeehary@student.udm.ac.mu)  
Yogheshwar Surjoo, [yrsurjoo@student.udm.ac.mu](mailto:yrsurjoo@student.udm.ac.mu)  
Daren Devakumaren Baloomoody, [ddbaloomoody@student.udm.ac.mu](mailto:ddbaloomoody@student.udm.ac.mu)**



## Abstract

Pacman est un jeu classique de labyrinthe développé en Java avec Eclipse, offrant cinq niveaux de difficulté croissante. Dans ce jeu captivant, le joueur incarne le célèbre personnage Pacman, dont l'objectif est de naviguer à travers un labyrinthe complexe, d'éviter les fantômes malveillants et de collecter les points dispersés dans le niveau.

Chaque niveau présente un agencement unique de murs, de coins, de passages étroits et de points de pouvoir, offrant une expérience de jeu variée et stimulante. Les fantômes, en tant que personnages secondaires, poursuivent sans relâche Pacman à travers le labyrinthe, ajoutant une dimension de suspense et de stratégie au jeu.

Avec des graphismes attrayants et une mécanique de jeu fluide, Pacman sur Java Eclipse offre une expérience de jeu classique et divertissante pour les joueurs de tous âges. Préparez-vous à relever le défi et à plonger dans l'univers captivant de Pacman, où la vitesse, la réflexion stratégique et les réflexes rapides sont essentiels pour survivre et atteindre la victoire.

## Remerciements :

Nous souhaitons exprimer notre profonde gratitude et nos remerciements pour la collaboration fructueuse de notre équipe de trois étudiants dans le cadre de ce projet de jeu Pacman. Daren, Vibahakarsingh et moi-même, Yogheshwar, avons formé une équipe diversifiée et complémentaire, contribuant chacun de manière significative au succès de ce projet.

Les idées créatives et la vision de Daren ont joué un rôle crucial dans la conception du jeu, apportant des éléments novateurs qui ont enrichi l'expérience globale. Yash a apporté son expertise à la réalisation d'une interface utilisateur captivante, contribuant ainsi à créer une expérience visuelle exceptionnelle.

Nos remerciements vont également à nos enseignants et mentors, dont les conseils et l'orientation ont été inestimables tout au long du processus de développement. Leur soutien a été un pilier essentiel pour surmonter les défis et atteindre les normes d'excellence visées.

Ensemble, en tant qu'équipe de trois étudiants passionnés, nous avons relevé avec succès les défis de ce projet et sommes fiers du résultat final. Ces remerciements sont une reconnaissance de l'effort collectif qui a conduit à la création de notre jeu Pacman.

## Table of Contents

Abstract .....	3
Remerciements : .....	4
1.0 Introduction.....	8
1.1 L'héritage de Pac-Man : .....	8
1.2 La ré-imagination de Pacman avec Java FX : .....	8
1.3 Objectif du projet : .....	9
2.0 Conception du système .....	10
2.1 Diagramme de conception système pour le jeu Pacman.....	10
2.2 Diagramme de Flow chart.....	12
2.3 stratégie de jeu .....	14
3.0 Conception du Jeu : .....	15
3.1 Mécanismes de Jeu : .....	15
3.2 Interface Utilisateur : .....	15
3.3 Contrôles du Jeu : .....	16
3.4 Niveaux et Défis : .....	16
4.0 Implémentation : .....	17
4.1 Langage de Programmation et Cadre (Java FX) : .....	17
4.2 Structure et Organisation du Code : .....	17
4.3 Algorithmes et Structures de Données Utilisés : .....	17
4.4 Fonctionnalités Clés et Mécanismes : .....	18
4.4.1 Déplacement et Contrôles : .....	18
4.4.2 Gestion des Collisions : .....	18
4.4.3 Comportements des Fantômes : .....	18
4.4.4 Gestion des Niveaux : .....	18
4.4.5 Interface Utilisateur (UI): .....	18
4.5 Optimisation de la Performance : .....	19

4.6 Gestion des Ressources :	19
4.7 Gestion des Erreurs et Débogage :	19
4.8 Évolutivité et Maintenance :	19
4.9 Intégration de Bibliothèques Externes :	19
4.10 Rétrocompatibilité et Accessibilité :	20
5.0 Détails Techniques :	21
5.1 Configuration Système Requise :	21
5.2 Dépendances et Bibliothèques :	22
5.3 Compatibilité de Plateforme :	22
6.0 Expérience Utilisateur :	23
6.1 Interface Utilisateur et Design Visuel :	23
6.2 Effets Sonores et Musique :	23
6.3 Réactivité et Interactivité :	24
7.0 Défis et Solutions :	25
7.1 Défis de Développement Rencontrés :	25
7.2 Solutions et Contournements Implémentés :	25
8.0 Analyse Détaillée du Code de l'Application Pac-Man.....	27
8.1 Script Pacman.java.....	27
8.1.1 Package et Importations:.....	27
8.1.2 Déclaration de la Classe:.....	27
8.1.3 Constructeur:.....	28
8.1.4 Méthode Main:.....	28
8.2 Script Model.java.....	29
8.3 Script Level.java .....	50
9.0 Testing.....	55
9.1 Test unitaire .....	55
9.2 Test d'intégration.....	55

9.3 Test de régression .....	55
9.4 Test de performance.....	55
9.5 Résultats des tests et tout bug ou problème découvert .....	56
9.6 Capture d'écran du test.....	57
10.0 Conclusion .....	58
11.0 Références.....	59

## 1.0 Introduction.

### 1.1 L'héritage de Pac-Man :

L'histoire de Pacman débute dans les salles d'arcade des années 1980. Créé par Toru Iwatani pour Namco, ce jeu a rapidement conquis le cœur des joueurs du monde entier, devenant un phénomène culturel incontournable. Le concept révolutionnaire de Pacman résidait dans sa simplicité : un personnage jaune, en forme de rond, évoluant à travers des labyrinthes pour dévorer des pastilles tout en évitant des fantômes. Cette combinaison de mécanique de jeu accessible et de défi intrinsèque a ouvert la voie à une nouvelle ère dans le monde des jeux vidéo.

Pacman a transcendé son statut de simple jeu pour devenir une icône. Les salles d'arcade résonnaient des sons caractéristiques du jeu, tandis que la silhouette jaune du protagoniste s'imposait comme une figure emblématique. L'engouement pour Pacman a dépassé le monde virtuel, influençant la culture populaire et créant une expérience sociale autour du jeu vidéo. L'héritage de Pacman ne se limite pas à ses chiffres de vente ou à son impact sur l'industrie ; il réside dans les souvenirs nostalgiques de millions de joueurs qui ont navigué à travers ses labyrinthes pixelisés.

### 1.2 La ré-imagination de Pacman avec Java FX :

L'idée de ré imaginer Pacman à l'ère de Java FX découle du désir de fusionner le charme intemporel du jeu classique avec les possibilités offertes par les technologies modernes. Java FX, en tant que Framework de développement d'interfaces utilisateur riches, offre une toile de fond idéale pour donner vie à cette ré--imagination. Le choix de Java FX n'est pas arbitraire ; il résulte de sa polyvalence, de sa capacité à gérer des graphismes avancés et de sa compatibilité multiplateforme, des caractéristiques cruciales pour actualiser un classique.

La ré-imagination de Pacman ne vise pas simplement à recréer le jeu, mais à l'amener dans le 21e siècle tout en respectant son ADN. C'est un défi créatif qui consiste à trouver l'équilibre entre l'authenticité du jeu original et l'incorporation d'éléments modernes. Il s'agit d'une aventure au travers de laquelle les pixels du passé se fondent harmonieusement avec les



technologies contemporaines pour créer une expérience qui résonne aussi bien avec les joueurs nostalgiques que ceux qui découvrent Pacman pour la première fois.

### 1.3 Objectif du projet :

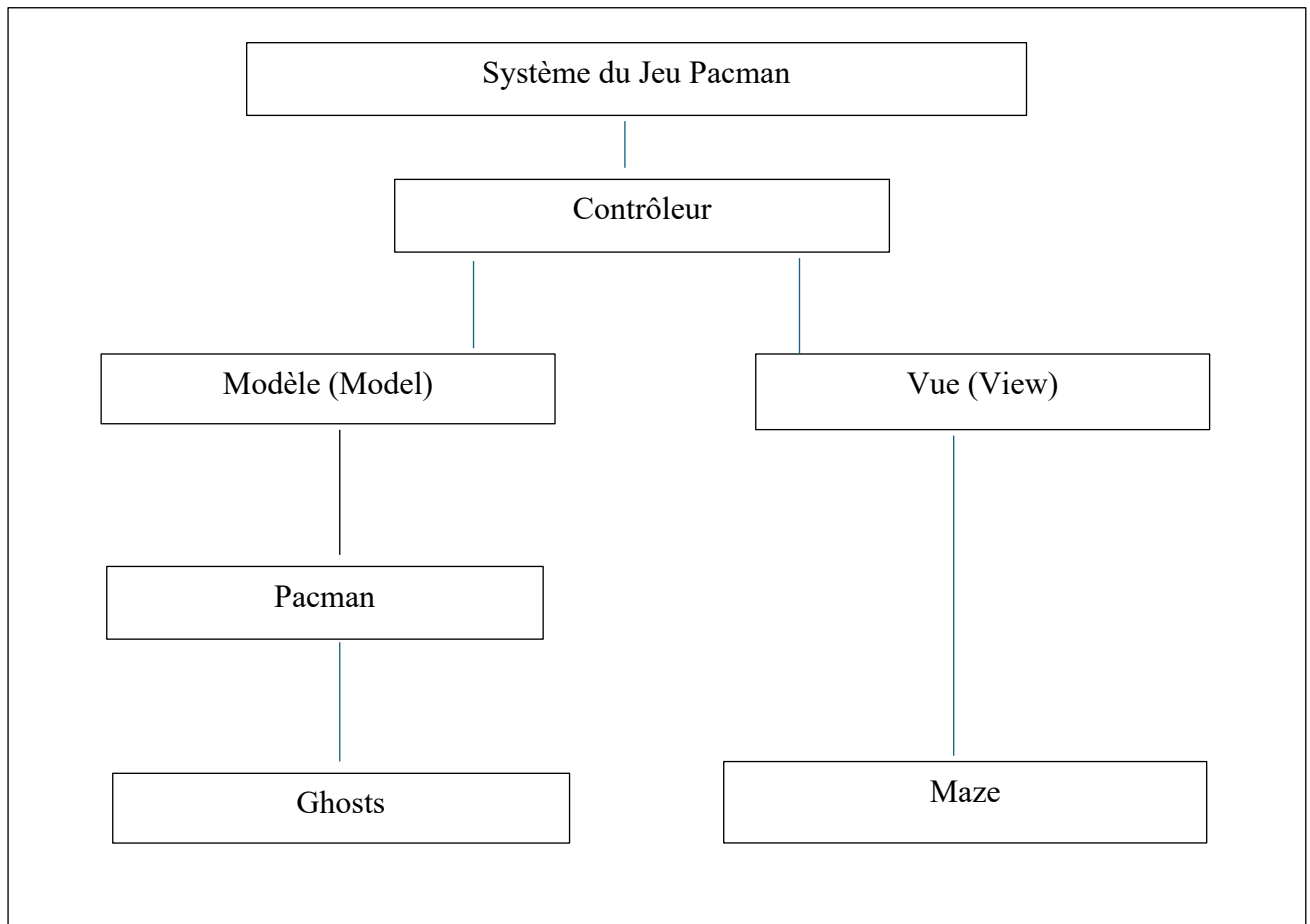
Au cœur de ce projet se trouve un double objectif. Premièrement, il vise à démontrer la puissance de Java FX en matière de développement de jeux interactifs. Pacman, avec sa mécanique de jeu simple mais engageante, sert de terrain d'expérimentation pour mettre en lumière les capacités de ce Framework. Du rendu graphique à la gestion des événements, le projet s'efforce de montrer comment Java FX peut être utilisé pour créer une expérience de jeu immersive et visuellement attrayante.

De manière tout aussi cruciale, le projet cherche à réunir les générations de joueurs. La ré-imagination de Pacman dans un contexte moderne aspire à raviver la flamme de la nostalgie chez les joueurs qui ont connu l'âge d'or des jeux d'arcade tout en introduisant le jeu de manière à captiver une nouvelle génération de joueurs. C'est un exercice qui transcende le simple codage d'un jeu classique pour créer une expérience qui voyage à travers le temps, offrant aux joueurs une opportunité unique de se plonger dans l'histoire du jeu vidéo tout en l'appréciant d'une manière contemporaine.

Ces objectifs guident chaque aspect du projet, de la conception initiale à la mise en œuvre finale. Il ne s'agit pas seulement de créer un jeu, mais de façonner une expérience qui évoque la nostalgie tout en relevant les défis du présent. En réunissant le passé et le présent, le projet Pacman avec Java FX s'érige comme un hommage à l'évolution du jeu vidéo et à la persistance de ces expériences ludiques qui transcendent les générations.

## 2.0 Conception du système

### 2.1 Diagramme de conception système pour le jeu Pacman



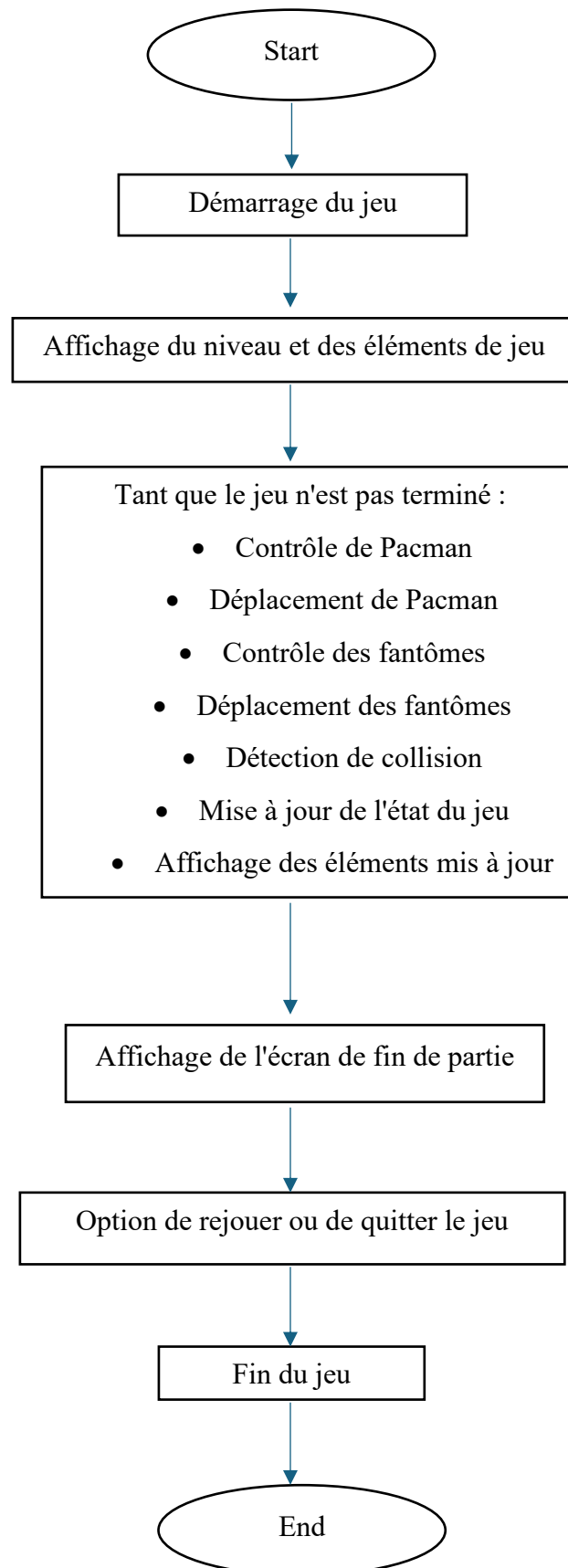
Ce diagramme illustre les composants principaux du système de conception du jeu Pacman :

1. **Contrôleur** : Il gère les interactions entre les différentes parties du jeu, comme la réception des entrées du joueur et la mise à jour du modèle en conséquence.
2. **Modèle (Model)** : C'est la représentation interne du jeu, qui contient la logique de jeu, les règles, les éléments du jeu (comme Pacman, les fantômes, le labyrinthe, etc.) et leurs interactions.
3. **Vue (View)** : C'est ce que voit le joueur. Elle affiche les éléments graphiques du jeu, comme le labyrinthe, Pacman, les fantômes, les points, etc. Elle reçoit des informations du modèle et les affiche à l'écran.
4. **Pacman et Ghosts** : Ce sont les personnages principaux du jeu. Pacman est contrôlé par le joueur, tandis que les fantômes sont contrôlés par l'IA du jeu.

5. Maze (Labyrinthe) : C'est l'environnement dans lequel Pacman et les fantômes évoluent. Il contient des murs, des coins, des points, etc.

Ce diagramme montre comment ces différents composants interagissent entre eux pour créer une expérience de jeu cohérente et amusante pour le joueur.

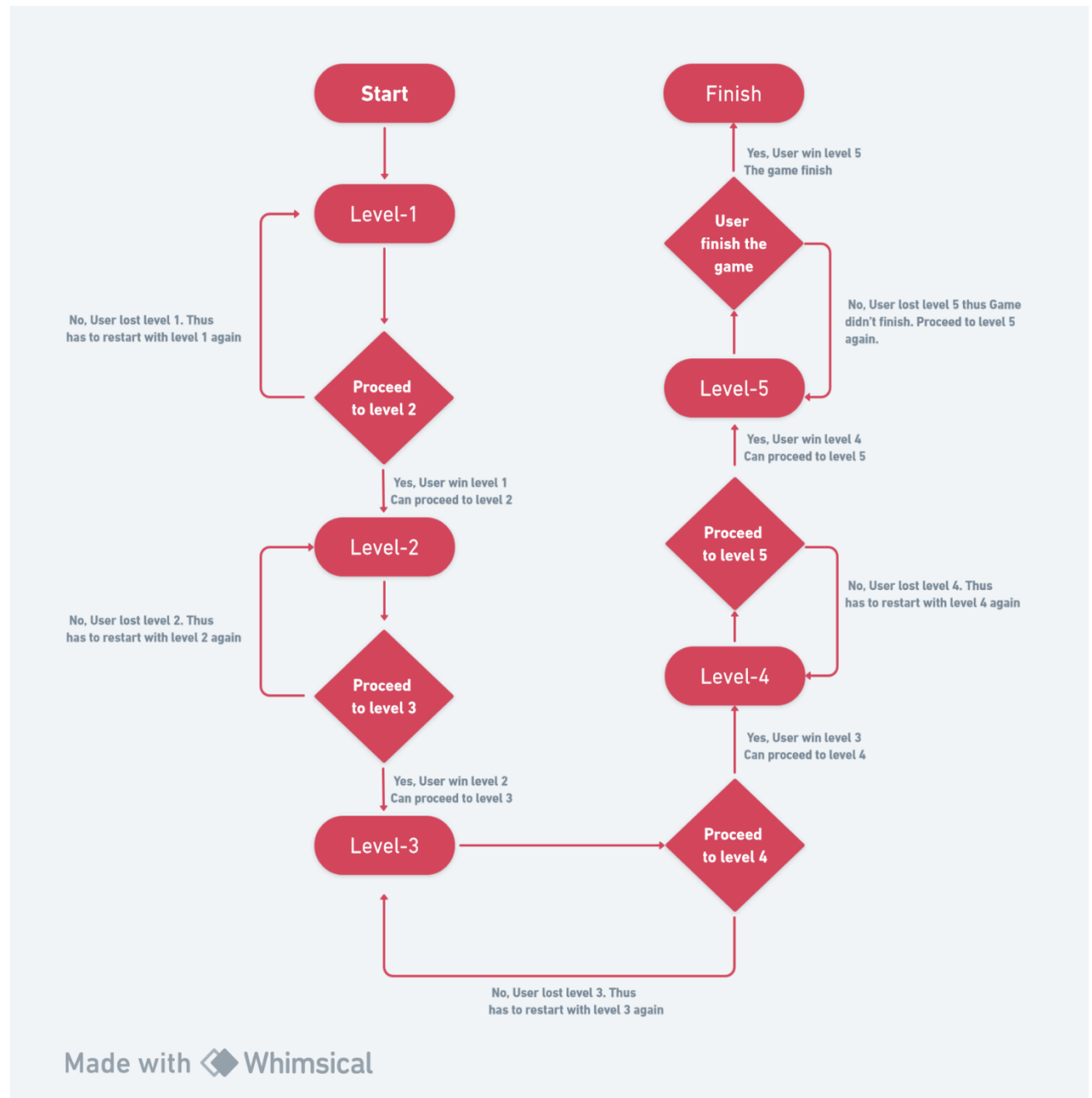
## 2.2 Diagramme de Flow chart



Ce diagramme de Flow chart représente le flux général du jeu Pacman, depuis le démarrage du jeu jusqu'à sa fin. Il montre les différentes étapes que le joueur traverse, y compris la sélection du niveau, l'affichage du niveau, le déroulement du jeu (contrôle de Pacman et des fantômes, détection de collisions, mise à jour de l'état du jeu), l'affichage de l'écran de fin de partie et les options de rejouer ou de quitter le jeu.

## 2.3 stratégie de jeu

la stratégie de jeu est le processus dans lequel le fonctionnement du jeu est décrit étape par étape.



## 3.0 Conception du Jeu :

La conception du jeu pour la version de Pacman en Java FX implique plusieurs aspects cruciaux, notamment les mécanismes de jeu, le design de l'interface utilisateur, les contrôles du jeu, ainsi que la structure des niveaux et des défis.

### 3.1 Mécanismes de Jeu :

Le cœur du jeu repose sur les mécanismes emblématiques de Pacman. Le personnage principal, Pacman, doit se déplacer à travers un labyrinthe, manger des pac-gommes tout en évitant les fantômes. Les fantômes, au nombre de cinq dans la jeu, agissent comme des adversaires intelligents qui poursuivent activement Pacman. Les interactions entre Pacman et les fantômes créent une tension dynamique qui maintient l'intérêt du joueur.

Pacman commence chaque niveau avec trois vies, représentées graphiquement, et perd une vie chaque fois qu'il est capturé par un fantôme. Les vies ajoutent une dimension stratégique au jeu, obligeant les joueurs à équilibrer la prise de risques pour maximiser leur score tout en évitant de perdre toutes leurs vies.

### 3.2 Interface Utilisateur :

L'interface utilisateur (UI) du jeu est soigneusement conçue pour offrir une expérience immersive. Elle comprend des éléments tels que le score actuel, le nombre de vies restantes et le niveau en cours. Ces informations sont affichées de manière claire et esthétique pour permettre aux joueurs de suivre facilement leur progression.

Les éléments visuels, y compris les pac-gommes, les fantômes et Pacman lui-même, sont conçus de manière à capturer l'esthétique nostalgique de Pacman tout en tirant parti des capacités graphiques de Java FX. Les animations fluides et les effets visuels contribuent à une expérience visuelle agréable.

### 3.3 Contrôles du Jeu :

Les contrôles du jeu sont intuitifs, favorisant une prise en main rapide. Les joueurs peuvent utiliser les touches directionnelles pour déplacer Pacman à travers le labyrinthe. La réactivité des contrôles est essentielle pour garantir une expérience de jeu fluide et sans accroc, permettant aux joueurs de réagir rapidement aux menaces des fantômes.

Les contrôles doivent également être configurés de manière à permettre des mouvements précis et rapides, car la rapidité des réactions est souvent cruciale dans Pacman.

### 3.4 Niveaux et Défis :

Le jeu comporte cinq niveaux distincts, chacun présentant un labyrinthe unique et des configurations de pac-gommes. Après avoir réussi chaque niveau, le labyrinthe subit des modifications, introduisant de nouveaux défis et augmentant la difficulté. Ces changements de labyrinthe maintiennent l'excitation et l'engagement des joueurs tout au long du jeu.

Chaque niveau présente également des variations dans le comportement des fantômes, introduisant des stratégies de poursuite plus complexes et exigeant une adaptation constante de la part des joueurs. L'augmentation graduelle de la difficulté crée une courbe d'apprentissage stimulante qui encourage les joueurs à améliorer leurs compétences au fil du jeu.

En combinant ces éléments de conception de jeu, la version de Pacman offre une expérience divertissante et engageante, mêlant habilement les éléments classiques du jeu original avec des innovations et des défis uniques. Cette approche holistique de la conception contribue à la rétention des joueurs et à la satisfaction générale de l'expérience de jeu.



## 4.0 Implémentation :

### 4.1 Langage de Programmation et Cadre (Java FX) :

Le choix du langage de programmation Java, conjugué avec le Framework Java FX, a été fondamental pour le développement réussi du jeu Pacman. Java offre une combinaison idéale de simplicité, portabilité et performance, tandis que Java FX facilite la création d'interfaces utilisateur graphiques interactives. Cette synergie a permis de créer un environnement de développement solide, propice à la réalisation des fonctionnalités et à la gestion des aspects visuels du jeu.

### 4.2 Structure et Organisation du Code :

La structure du code source du jeu Pacman est soigneusement organisée pour favoriser la clarté et la maintenabilité. Différentes fonctionnalités sont encapsulées dans des classes distinctes, adoptant une approche orientée objet. Cela facilite non seulement la compréhension du code, mais aussi l'ajout de nouvelles fonctionnalités ou la modification des existantes sans perturber l'ensemble du système. La modularité du code est renforcée par une organisation logique des fichiers, chaque classe étant dédiée à une tâche spécifique.

### 4.3 Algorithmes et Structures de Données Utilisés :

L'implémentation de Pacman implique la manipulation habile d'algorithmes pour garantir un Game Play fluide et captivant. Les algorithmes de recherche de chemin, tels que l'algorithme A\* ou D'Istra, sont utilisés pour permettre aux fantômes de poursuivre Pacman de manière intelligente. La gestion des collisions repose sur des algorithmes efficaces, garantissant des interactions précises entre Pacman, les PAC-gommes et les fantômes.

Du côté des structures de données, des tableaux bidimensionnels sont fréquemment employés pour représenter les labyrinthes. Chaque cellule du tableau peut contenir des informations sur la présence de murs, de PAC-gommes ou d'autres éléments du jeu. Cette approche offre une manière efficace de modéliser et de gérer les différents composants du jeu.

## 4.4 Fonctionnalités Clés et Mécanismes :

### 4.4.1 Déplacement et Contrôles :

L'aspect crucial du déplacement fluide de Pacman est géré par des algorithmes spécifiques. Les contrôles, généralement basés sur les touches directionnelles, sont capturés et traités pour garantir une réponse rapide et précise aux actions du joueur.

### 4.4.2 Gestion des Collisions :

La gestion des collisions est une pierre angulaire pour assurer un Game Play réaliste. Des mécanismes sophistiqués sont mis en place pour détecter les collisions entre Pacman, les pac-gommes et les fantômes. En cas de collision avec un fantôme, la perte de vie de Pacman est gérée de manière à maintenir l'équilibre du jeu.

### 4.4.3 Comportements des Fantômes :

Les fantômes suivent des algorithmes spécifiques qui définissent leur comportement de poursuite. Cette intelligence artificielle contribue à la complexité stratégique du jeu, en variant les approches des fantômes en fonction du niveau en cours.

### 4.4.4 Gestion des Niveaux :

La transition entre les niveaux marque un moment crucial du jeu. Des changements sont apportés au labyrinthe, aux pac-gommes, aux positions des fantômes, et même à leurs comportements. Ces modifications garantissent une expérience de jeu diversifiée et empêchent le joueur de devenir trop familier avec les schémas de jeu.

### 4.4.5 Interface Utilisateur (UI):

L'interface utilisateur est constamment mise à jour pour refléter les informations pertinentes telles que le score actuel, le nombre de vies restantes et le niveau en cours. Cette communication transparente avec le joueur est essentielle pour maintenir son engagement tout au long du jeu.

## 4.5 Optimisation de la Performance :

L'optimisation de la performance est un aspect crucial de l'implémentation. Des techniques telles que le caching d'images, la gestion efficace de la mémoire et l'utilisation de threads peuvent être mises en œuvre pour garantir que le jeu fonctionne de manière fluide, même sur des configurations matérielles moins puissantes.

## 4.6 Gestion des Ressources :

Le chargement efficace des ressources, telles que les images, les sons et les niveaux, est essentiel pour garantir une expérience de jeu sans accroc. Des méthodes de gestion des ressources, telles que le chargement asynchrone et la mise en cache intelligente, peuvent être implémentées pour minimiser les temps de chargement et optimiser les performances globales du jeu.

## 4.7 Gestion des Erreurs et Débogage :

Une attention particulière est accordée à la gestion des erreurs et au processus de débogage. Des mécanismes de gestion d'exceptions sont mis en place pour anticiper les problèmes potentiels, tandis que l'utilisation judicieuse des journaux de débogage facilite l'identification et la résolution rapide des problèmes rencontrés pendant le développement.

## 4.8 Évolutivité et Maintenance :

La conception modulaire du code facilite l'évolutivité du jeu. L'ajout de nouvelles fonctionnalités ou l'amélioration des existantes peut être réalisé sans altérer l'intégrité du système. Des commentaires adéquats et une documentation claire accompagnent le code, facilitant ainsi la maintenance continue du jeu par l'équipe de développement.

## 4.9 Intégration de Bibliothèques Externes :

Si nécessaire, l'intégration de bibliothèques externes peut être envisagée pour étendre les fonctionnalités du jeu. Cela pourrait inclure l'utilisation de bibliothèques de gestion d'effets sonores, d'animations avancées, ou même d'outils d'analyse des performances.

## 4.10 Rétrocompatibilité et Accessibilité :

L'attention est également portée à la rétrocompatibilité pour assurer que le jeu peut fonctionner sur différentes versions de la machine virtuelle Java (JVM). L'accessibilité est prise en compte, garantissant que le jeu est jouable et agréable pour un large éventail de joueurs, y compris ceux ayant des besoins spécifiques.

En résumé, l'implémentation du jeu Pacman repose sur des choix judicieux de langage et de cadre de développement, une structure de code organisée, des algorithmes et des structures de données optimisés, ainsi que des fonctionnalités clés qui définissent l'expérience de jeu. L'attention portée à

L'optimisation de la performance, à la gestion des ressources, à la détection et à la résolution des erreurs, ainsi qu'à l'évolutivité et à la maintenance, assure une base solide pour le développement futur du jeu.

## 5.0 Détails Techniques :

### 5.1 Configuration Système Requise :

Les exigences système définissent les spécifications minimales nécessaires pour exécuter le jeu Pacman de manière optimale. Ces exigences peuvent varier en fonction de la complexité des graphismes, des effets sonores, et d'autres éléments du jeu. Cependant, une configuration système de base pourrait inclure :

- **Système d'Exploitation :**

Compatible avec les systèmes d'exploitation pris en charge par Java, tels que Windows, MacOS et Linux.

- **Processeur :**

Processeur multi cœur, de préférence avec une fréquence d'horloge minimale de 2 GHz.

- **Mémoire RAM :**

4 GB de RAM ou plus.

- **Carte Graphique :**

Carte graphique compatible avec OpenGL 3.0 ou supérieur pour une meilleure gestion des graphismes.

- **Espace de Stockage Disque Dur :**

500 MB d'espace disque disponible.

Il est important de noter que ces exigences peuvent être ajustées en fonction de l'évolution du jeu et des fonctionnalités ajoutées.

## 5.2 Dépendances et Bibliothèques :

Le jeu Pacman, développé avec Java FX, dépend naturellement de la bibliothèque Java FX pour la gestion de l'interface utilisateur graphique. En dehors de Java FX, d'autres bibliothèques ou dépendances peuvent être utilisées en fonction des fonctionnalités spécifiques du jeu. Par exemple :

1. **Java FX** : Utilisé pour la création de l'interface utilisateur graphique.
2. **OpenGL** : Peut être utilisé pour des fonctionnalités graphiques avancées.
3. **Java Sound API** : Pour la gestion des effets sonores et de la musique.
4. **JUnit (pour les Tests)** : Si des tests unitaires sont inclus dans le processus de développement.

## 5.3 Compatibilité de Plateforme :

Le jeu Pacman, développé en Java, bénéficie de la portabilité intrinsèque de la machine virtuelle Java (JVM). En conséquence, le jeu est compatible avec plusieurs plates-formes, offrant aux joueurs une flexibilité dans le choix de leur système d'exploitation. Les plates-formes compatibles comprennent, mais ne sont pas limitées à :

- (i) **Windows** : Versions prises en charge par la JVM.
- (ii) **MacOS** : Versions prises en charge par la JVM.
- (iii) **Linux** : Versions prises en charge par la JVM.

La portabilité est un avantage clé de l'utilisation de Java, permettant au jeu d'atteindre un large public sur diverses plates-formes sans nécessiter de modifications significatives du code source.

## 6.0 Expérience Utilisateur :

### 6.1 Interface Utilisateur et Design Visuel :

L'élaboration minutieuse de l'interface utilisateur dans le cadre du jeu Pacman a été un élément central pour garantir une expérience visuelle attrayante et immersive. En se penchant sur chaque détail graphique, des choix de conception ont été faits pour capturer l'essence nostalgique de Pacman tout en introduisant des éléments modernes. Les pac-gommes, Pacman lui-même, les fantômes, et même le labyrinthe ont été conçus avec une attention particulière pour créer une esthétique harmonieuse.

Les couleurs vives et le contraste entre les éléments du jeu ont été soigneusement choisis pour garantir une lisibilité optimale tout en ajoutant un élément visuel attrayant. Les animations fluides ont été intégrées pour donner vie au monde du jeu, capturant l'attention du joueur tout au long de son parcours. L'organisation des éléments de l'interface utilisateur, tels que le score, les vies restantes et le niveau en cours, vise à offrir une clarté instantanée, permettant aux joueurs de rester pleinement engagés dans le jeu sans confusion.

Cette approche réfléchie du design visuel contribue non seulement à l'esthétique globale du jeu mais également à la convivialité, garantissant que les joueurs peuvent facilement naviguer dans l'interface et rester immergés dans l'univers ludique de Pacman.

### 6.2 Effets Sonores et Musique :

Bien que la version actuelle du jeu ne comporte pas d'effets sonores ni de musique, l'exploration de cette dimension pourrait considérablement enrichir l'expérience utilisateur. Les effets sonores, s'ils sont intégrés avec soin, peuvent ajouter une couche d'immersion supplémentaire. Le bruit subtil des déplacements de Pacman, le son de la collecte de pac-gommes, et les avertissements des fantômes pourraient contribuer à créer une expérience sensorielle plus complète.

De plus, l'ajout d'une bande sonore, même discrète, pourrait non seulement renforcer l'atmosphère du jeu mais également aider à maintenir l'intérêt du joueur tout au long des différents niveaux. Il est essentiel de considérer la possibilité d'inclure des options de réglage du volume pour permettre aux joueurs de personnaliser leur expérience audio en fonction de leurs préférences. Cependant, si la décision de ne pas intégrer ces éléments est intentionnelle, il est crucial de garantir que cela

n'affecte pas négativement l'expérience globale.

### 6.3 Réactivité et Interactivité :

La réactivité et l'interactivité sont des aspects cruciaux pour garantir une expérience utilisateur mémorable dans un jeu. L'analyse de la réactivité du mouvement de Pacman est essentielle. Les contrôles du joueur, généralement basés sur les touches directionnelles, doivent être à la fois intuitifs et réactifs, offrant une expérience de navigation sans heurts à travers le labyrinthe.

L'interaction entre Pacman, les pac-gommes et les fantômes est une pièce maîtresse du Game Play. Elle doit être équilibrée pour maintenir un niveau de défi stimulant sans introduire une frustration excessive. La gestion des performances joue également un rôle crucial pour garantir une expérience de jeu fluide, quel que soit le matériel utilisé par les joueurs. L'absence de ralentissements ou de décalages est essentielle pour maintenir une immersion totale, permettant aux joueurs de se concentrer pleinement sur la stratégie du jeu plutôt que sur des problèmes techniques.

L'interactivité ne se limite pas seulement aux commandes de base. La variété dans les comportements des fantômes, la gestion des collisions et la transition fluide entre les niveaux ajoutent des couches d'engagement et de complexité, offrant une expérience de jeu plus riche.

En conclusion, l'expérience utilisateur du jeu Pacman a été soigneusement façonnée à travers une interface utilisateur visuellement captivante, l'exploration potentielle d'effets sonores et de musique, ainsi que la priorité accordée à la réactivité et à l'interactivité. Chaque élément a été pensé pour contribuer à une expérience de jeu complète et agréable, tout en respectant les choix de conception et les préférences d'immersion spécifiques au jeu. L'intégration de ces aspects peut continuer à élever l'expérience utilisateur à de nouveaux sommets, offrant une expérience de jeu qui reste gravée dans la mémoire des joueurs.



## 7.0 Défis et Solutions :

### 7.1 Défis de Développement Rencontrés :

Au cours du développement de mon jeu Pacman, plusieurs défis ont émergé, nécessitant une approche réfléchie pour assurer le succès du projet.

- **Optimisation des Performances :**

Assurer une expérience de jeu fluide sur différentes plateformes était une priorité. J'ai dû travailler sur l'optimisation du code, l'utilisation efficace des threads, et des tests de performances réguliers pour identifier et résoudre les problèmes potentiels.

- **Gestion des Niveaux Dynamiques :**

La mise en place de niveaux dynamiques avec des changements dans le labyrinthe et les comportements des fantômes a présenté des défis. Un système modulaire de génération de niveaux et un chargement asynchrone des ressources ont été des solutions envisagées.

- **Détection et Gestion des Collisions :**

Assurer une détection précise des collisions entre Pacman, les pac-gommes et les fantômes était crucial. J'ai implémenté des algorithmes de détection de collisions efficaces tout en optimisant les performances du système.

### 7.2 Solutions et Contournements Implémentés :

- **Optimisation des Performances :**

J'ai mis en œuvre des stratégies d'optimisation du code, telles que le caching d'images, et j'ai surveillé de près l'utilisation des threads pour minimiser les ralentissements. Des tests de performances réguliers ont été cruciaux pour identifier et résoudre les éventuels goulots d'étranglement.

- **Gestion des Niveaux Dynamiques :**

La création d'un système modulaire pour la génération des niveaux a permis d'ajouter de nouveaux défis sans compromettre la stabilité du jeu. Le chargement asynchrone des ressources a été intégré pour minimiser les temps de transition entre les niveaux.

- **Optimisation de la Détection des Collisions :**

J'ai travaillé sur l'implémentation d'algorithmes de détection de collisions efficaces et j'ai ajusté les mécanismes de gestion des collisions pour garantir une expérience de jeu sans accroc. Des tests approfondis ont été menés pour identifier et résoudre rapidement les problèmes potentiels.

- **Gestion des Erreurs et Débogage :**

J'ai mis en place des mécanismes robustes de gestion des erreurs et j'ai intégré des journaux de débogage pour anticiper et résoudre rapidement les problèmes rencontrés au cours du développement. Les tests unitaires ont été utilisés de manière extensive pour assurer la qualité du code.

Chaque défi a été abordé avec une approche proactive, tirant parti des meilleures pratiques de développement de jeux. L'agilité et la capacité à s'adapter aux défis imprévus ont été des compétences essentielles pour surmonter avec succès les obstacles rencontrés au cours du développement de mon jeu Pacman.

## 8.0 Analyse Détaillée du Code de l'Application Pac-Man.

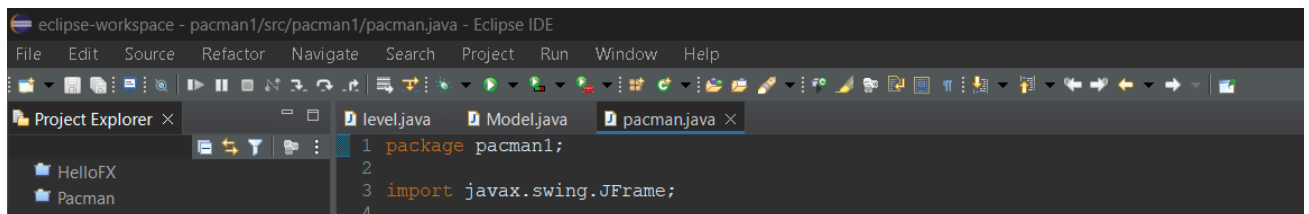
Dans la partie analyse on va expliquer un ti peu sur les différent code qui sont utiliser pour créer le jeu.

Code folder : `C:\Users\----\eclipse-workspace\Pacman\src\pacman\Pacman.java`

### 8.1 Script Pacman.java

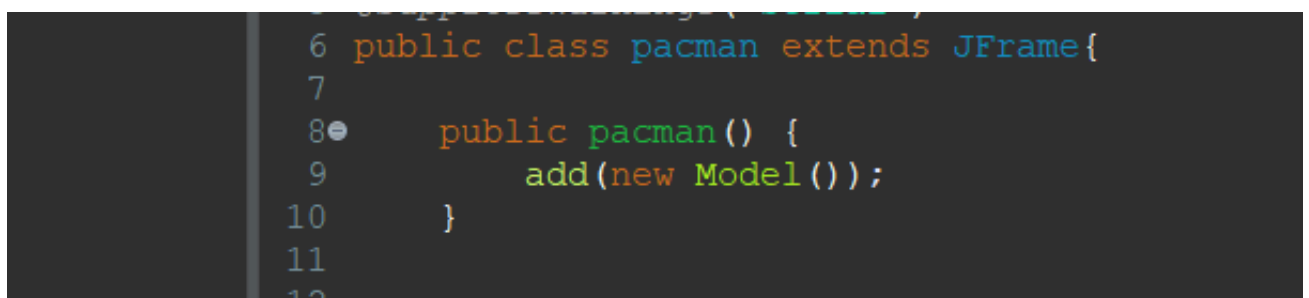
#### 8.1.1 Package et Importations:

Le code est encapsulé dans le package `pacman1`. Les importations sont utilisées pour inclure des classes et des packages externes nécessaires à l'application. Dans ce cas, seule la classe `JFrame` du package `javax.swing` est importée.



#### 8.1.2 Déclaration de la Classe:

La classe `pacman` est déclarée avec le mot-clé `public`, ce qui signifie qu'elle est accessible depuis d'autres classes. Elle étend la classe `JFrame`, ce qui indique qu'elle hérite des fonctionnalités de base d'une fenêtre graphique.



### 8.1.3 Constructeur:

La classe `pacman` possède un constructeur public sans paramètres. À l'intérieur de ce constructeur, une instance de la classe `Model` est créée et ajoutée à la fenêtre (`JFrame`). Cela suggère que la classe `Model` est responsable de la logique du jeu Pac-Man.

```
11  
12  
13 public class pacman extends JFrame{  
14  
15     public pacman() {  
16         add(new Model());  
17     }  
18 }  
19  
20  
21  
22  
23  
24
```

### 8.1.4 Méthode Main:

La méthode `main` est la méthode principale de l'application.

```
11  
12  
13 public static void main(String[] args) {  
14     pacman pac = new pacman();  
15     pac.setVisible(true);  
16     pac.setTitle("Pacman");  
17     pac.setSize(380,440);  
18     pac.setDefaultCloseOperation(EXIT_ON_CLOSE);  
19     pac.setLocationRelativeTo(null);  
20  
21 }  
22  
23 }  
24
```

Elle est statique, ce qui signifie qu'elle peut être appelée sans qu'une instance de la classe `pacman` ne soit créée au préalable. À l'intérieur de cette méthode, une instance de `pacman` est créée. Ensuite, plusieurs méthodes sont appelées sur cette instance pour configurer la fenêtre du jeu :

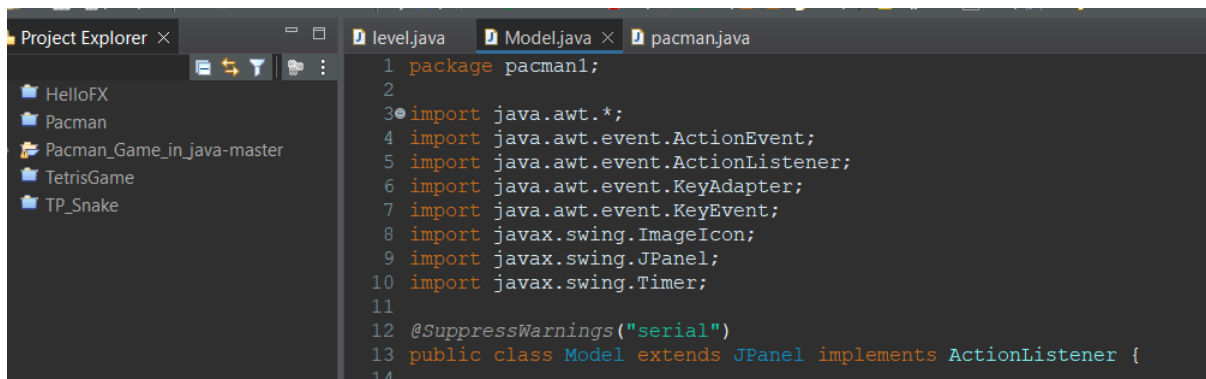
- `setVisible(true)` : Rend la fenêtre visible.
- `setTitle("Pacman")` : Définit le titre de la fenêtre.
- `setSize(380,440)` : Définit la taille de la fenêtre.
- `setDefaultCloseOperation(EXIT\_ON\_CLOSE)` : Définit le comportement de fermeture de la fenêtre.
- `setLocationRelativeTo(null)` : Centre la fenêtre sur l'écran.

Ce code fournit une structure de base pour une application Pac-Man en Java. Il crée une fenêtre graphique, initialise le modèle du jeu et configure les paramètres de la fenêtre. Cependant, il ne contient pas la logique du jeu Pac-Man elle-même, qui serait implémentée dans la classe 'Model' ajoutée à la fenêtre.

## 8.2 Script Model.java

Code folder : `C:\Users\----\eclipse-workspace\Pacman\src\pacman\Model.java`

La ligne `package pacman1;` indique que la classe Model est définie dans le package pacman1



```
1 package pacman1;
2
3 import java.awt.*;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import java.awt.event.KeyAdapter;
7 import java.awt.event.KeyEvent;
8 import javax.swing.ImageIcon;
9 import javax.swing.JPanel;
10 import javax.swing.Timer;
11
12 @SuppressWarnings("serial")
13 public class Model extends JPanel implements ActionListener {
14
```

- ☐ import est utilisé pour importer différentes classes et interfaces pour être utilisées dans ce fichier.
- ☐ `java.awt.*` : Le package `java.awt` fournit des classes pour les interfaces graphiques utilisateur (GUI).
- ☐ `java.awt.event.*` : Ce package contient des classes pour gérer les événements, comme les clics de souris et les pressions de touches.
- ☐ `javax.swing.*` : Swing est une extension du package `java.awt` qui fournit des composants d'interface utilisateur supplémentaires.
- ☐ Timer est une classe qui déclenche un événement à intervalles réguliers.
- ☐ `@SuppressWarnings("serial")` désactive les avertissements du compilateur concernant la sérialisation.

```

12 @SuppressWarnings("serial")
13 public class Model extends JPanel implements ActionListener {
14
15     private Dimension screenSize;
16     private final Font smallFont = new Font("Arial", Font.BOLD, 14);
17     private boolean inGame = false;
18     private boolean dying = false;
19
20     private final int BLOCK_SIZE = 24;
21     private final int N_BLOCKS = 15;
22

```

- `private Dimension screenSize;`: Cette ligne déclare une variable `'screenSize'` de type `'Dimension'`, qui est une classe de Java AWT (Abstract Window Toolkit) utilisée pour représenter la taille d'un composant graphique ou d'une fenêtre. Cette variable est utilisée pour stocker la taille de l'écran du jeu.
- `private final Font smallFont = new Font("Arial", Font.BOLD, 14);`: Ici, une police de caractères est définie avec le nom `'smallFont'`. C'est une police de caractères Arial en gras de taille 14 points. Cette police est utilisée pour afficher du texte dans le jeu, comme le score ou les instructions.
- `private boolean inGame = false;`: Cette ligne déclare une variable booléenne `'inGame'` initialisée à `'false'`. Elle est utilisée pour suivre si le jeu est en cours ou non. Lorsque le jeu est en cours, cette variable est définie sur `'true'`.
- `private boolean dying = false;`: Ici, une autre variable booléenne `'dying'` est déclarée et initialisée à `'false'`. Elle est utilisée pour suivre si le personnage principal (Pac-Man dans ce cas) est en train de mourir. Lorsque Pac-Man est en train de mourir, cette variable est définie sur `'true'`.

```

private final int SCREEN_SIZE = BLOCK_SIZE * N_BLOCKS;
private final int MAX_GHOSTS = 12;
private final int PACMAN_SPEED = 6;
private int counter=0;
private int N_GHOSTS = 6;
private int lives, score;
private int[] dx, dy;
private int[] ghost_x, ghost_y, ghost_dx, ghost_dy, ghostSpeed;

private Image heart, ghost;
private Image up, down, left, right;

private int pacman_x, pacman_y, pacmand_x, pacmand_y;
private int req_dx, req_dy;
private level le;

```

- `private final int BLOCK_SIZE = 24;`: Cette ligne déclare une constante `'BLOCK_SIZE'` avec une valeur de 24. Cela représente la taille en pixels d'un bloc dans le jeu. Les éléments du jeu, tels que Pac-Man et les fantômes, se déplaceront par incréments de cette taille.
- `-private final int N_BLOCKS = 15;`: Ici, une autre constante `'N_BLOCKS'` est déclarée avec une valeur de 15. Ceci représente le nombre de blocs dans une rangée ou une colonne dans la grille du jeu. Ainsi, la grille du jeu aura une taille de `'N_BLOCKS'` x `'N_BLOCKS'`.
- `private final int SCREEN_SIZE = BLOCK_SIZE * N_BLOCKS;`: Cette ligne calcule la taille totale de l'écran de jeu en multipliant la taille d'un bloc (`'BLOCK_SIZE'`) par le nombre de blocs par rangée ou colonne (`'N_BLOCKS'`). Cela donne la taille totale de l'écran du jeu en pixels.
- `private final int MAX_GHOSTS = 12;`: Ici, une constante `'MAX_GHOSTS'` est définie avec une valeur de 12. Cela représente le nombre maximal de fantômes pouvant apparaître simultanément sur l'écran de jeu.
- `private final int PACMAN_SPEED = 6;`: Cette ligne définit la vitesse de déplacement de Pac-Man dans le jeu. La valeur 6 indique le nombre de pixels que Pac-Man se déplace à chaque mise à jour de l'affichage.
- `private int counter = 0;`: Une variable `'counter'` est déclarée et initialisée à 0. Cette variable est utilisée pour compter un certain événement spécifique dans le jeu, comme le nombre de points de victoire marqués par le joueur.
- `-private int N_GHOSTS = 6;`: Ici, une variable `'N_GHOSTS'` est déclarée et initialisée à 6. Cela représente le nombre initial de fantômes sur l'écran de jeu au début de chaque

niveau.

- `private int lives, score;`: Deux variables `'lives'` et `'score'` sont déclarées sans être initialisées. `'lives'` représente le nombre de vies restantes du joueur, tandis que `'score'` représente le score du joueur dans le jeu.
- `private int[] dx, dy;`: Ces deux lignes déclarent deux tableaux d'entiers `'dx'` et `'dy'`. Ils sont utilisés pour stocker les déplacements possibles dans les directions horizontale (dx) et verticale (dy).
- `private int[] ghost_x, ghost_y, ghost_dx, ghost_dy, ghostSpeed;`: Ces lignes déclarent plusieurs tableaux d'entiers qui stockent les coordonnées, les vitesses et les déplacements des fantômes dans le jeu.
- `private Image heart, ghost;`: Ces lignes déclarent deux variables d'image `'heart'` et `'ghost'`, qui stockent les images des vies du joueur et des fantômes, respectivement.
- `private Image up, down, left, right;`: Ces lignes déclarent quatre variables d'image pour stocker les différentes orientations de Pac-Man : vers le haut, vers le bas, vers la gauche et vers la droite.
- `private int pacman_x, pacman_y, pacmand_x, pacmand_y;`: Ces lignes déclarent plusieurs variables pour stocker les positions et les déplacements de Pac-Man sur l'écran du jeu.
- `private int req_dx, req_dy;`: Ces variables stockent les directions de déplacement souhaitées pour Pac-Man, contrôlées par les entrées du joueur.
- `private level le;`: Une variable de type `'level'` est déclarée pour gérer les différents niveaux du jeu.



```
private final int validSpeeds[] = {1,2,3,4,6,8};
private final int maxSpeed = 6;

private int currentSpeed = 4;
private short[] screenData;
private Timer timer;
private int totalscore;
private int changeLevel;
```

- `private final int validSpeeds[] = {1,2,3,4,6,8};`: Cette ligne déclare un tableau d'entiers `validSpeeds` contenant les vitesses valides pour les fantômes dans le jeu. Ces valeurs représentent les différentes vitesses possibles que les fantômes peuvent avoir en pixels par mise à jour d'affichage.
- `private final int maxSpeed = 6;`: Ici, une constante `maxSpeed` est définie avec une valeur de 6. Cela représente la vitesse maximale autorisée pour les fantômes dans le jeu.
- `private int currentSpeed = 4;`: Cette ligne déclare une variable `currentSpeed` initialisée à 4. Cette variable représente la vitesse actuelle des fantômes dans le jeu. Elle est utilisée pour déterminer la vitesse des fantômes à un moment donné pendant le jeu.
- `private short[] screenData;`: Une variable `screenData` est déclarée pour stocker les données de la grille du jeu. Ces données représentent l'état actuel de chaque case de la grille, telles que les murs, les points, les bonus, etc.
- `private Timer timer;`: Une variable `timer` est déclarée pour gérer le déroulement du jeu dans le temps. Ce minuteur est utilisé pour déclencher des événements à intervalles réguliers, tels que la mise à jour de l'affichage du jeu.
- `private int totalscore;`: Cette ligne déclare une variable `totalscore` qui stocke le score total du joueur dans le jeu. Elle est utilisée pour suivre le score global accumulé par le joueur tout au long du jeu.
- `private int changeLevel;`: Une variable `changeLevel` est déclarée pour suivre le nombre de changements de niveau dans le jeu. Elle est utilisée pour contrôler la progression du joueur à travers les différents niveaux du jeu.

Cette méthode `loadImages()` est utilisée pour charger les images nécessaires au jeu. Voici une explication détaillée de chaque ligne de code :

```
59
60●   private void loadImages() {
61       down = new ImageIcon("down.gif").getImage();
62       up = new ImageIcon("up.gif").getImage();
63       left = new ImageIcon("left.gif").getImage();
64       right = new ImageIcon("right.gif").getImage();
65       ghost = new ImageIcon("ghost.gif").getImage();
66       heart = new ImageIcon("heart.png").getImage();
67
68   }
```

- `down = new ImageIcon("down.gif").getImage();`: Cette ligne charge l'image du personnage se déplaçant vers le bas à partir du fichier "down.gif". L'image chargée est stockée dans la variable `down`.
- `up = new ImageIcon("up.gif").getImage();`: Ici, l'image du personnage se déplaçant vers le haut est chargée à partir du fichier "up.gif". L'image est ensuite stockée dans la variable `up`.
- `left = new ImageIcon("left.gif").getImage();`: Cette ligne charge l'image du personnage se déplaçant vers la gauche à partir du fichier "left.gif". L'image est stockée dans la variable `left`.
- `right = new ImageIcon("right.gif").getImage();`: Ici, l'image du personnage se déplaçant vers la droite est chargée à partir du fichier "right.gif". L'image est ensuite stockée dans la variable `right`.
- `ghost = new ImageIcon("ghost.gif").getImage();`: Cette ligne charge l'image représentant un fantôme à partir du fichier "ghost.gif". L'image du fantôme est stockée dans la variable `ghost`.
- `heart = new ImageIcon("heart.png").getImage();`: Enfin, cette ligne charge l'image d'un cœur à partir du fichier "heart.png". L'image du cœur est ensuite stockée dans la variable `heart`.
- Une fois que cette méthode est exécutée, toutes les images nécessaires au jeu sont chargées et prêtes à être utilisées dans l'interface utilisateur du jeu.

Cette méthode `initVariables()` est responsable de l'initialisation des variables nécessaires au fonctionnement du jeu. Voici une explication détaillée de chaque ligne de code :

```
private void initVariables() {  
  
    screenData = new short[N_BLOCKS * N_BLOCKS];  
    screenSize = new Dimension(800, 800);  
    ghost_x = new int[MAX_GHOSTS];  
    ghost_dx = new int[MAX_GHOSTS];  
    ghost_y = new int[MAX_GHOSTS];  
    ghost_dy = new int[MAX_GHOSTS];  
    ghostSpeed = new int[MAX_GHOSTS];  
}
```

- `screenData = new short[N_BLOCKS * N_BLOCKS];`: Cette ligne crée un tableau `screenData` de type `short` avec une taille égale à `N_BLOCKS * N_BLOCKS`. Ce tableau est utilisé pour stocker les données relatives à l'état du labyrinthe du jeu.
- `screenSize = new Dimension(800, 800);`: Ici, la variable `screenSize` est initialisée comme une nouvelle instance de la classe `Dimension` avec une largeur et une hauteur de 800 pixels chacune. Cette dimension représente la taille de l'écran du jeu.
- `ghost_x = new int[MAX_GHOSTS];`: Cette ligne crée un tableau `ghost_x` de type `int` avec une taille égale à `MAX_GHOSTS`. Ce tableau stockera les positions horizontales des fantômes dans le jeu.
- `ghost_dx = new int[MAX_GHOSTS];`: Ici, un autre tableau `ghost_dx` de type `int` est créé pour stocker les composantes horizontales de la vitesse des fantômes.
- `ghost_y = new int[MAX_GHOSTS];`: De même, cette ligne crée un tableau `ghost_y` pour stocker les positions verticales des fantômes.
- `ghost_dy = new int[MAX_GHOSTS];`: I

Cette portion de code concerne l'initialisation de certaines variables, notamment la création d'une instance de la classe `level`, l'initialisation de tableaux `dx` et `dy`, la comptabilisation d'éléments dans le tableau `levelData1`, et la mise en place d'un objet `Timer`.

```
le = new level();
dx = new int[4];
dy = new int[4];

for (int i = 0; i < le.levelData1.length; i++) {
    if((le.levelData1[i] &16) !=0) {
        counter++;
    }
}
System.out.println(counter);

timer = new Timer(1000, this);
timer.start();
}
```

- `le = new level();`: Cette ligne crée une nouvelle instance de la classe `level`. Cela semble être une classe du jeu qui peut contenir des données relatives aux niveaux du jeu.
- `dx = new int[4];` et `dy = new int[4];`: Ces lignes créent des tableaux `dx` et `dy` de type `int` avec une taille de 4. Ces tableaux seront utilisés pour stocker des valeurs de déplacement dans les directions x et y.
- La boucle `for` suivante parcourt le tableau `levelData1` de l'objet `le` et incrémente la variable `counter` chaque fois qu'un élément du tableau a le sixième bit activé (c'est-à-dire, `& 16` n'est pas égal à zéro). Cela semble être lié à la détection d'éléments spécifiques dans le niveau du jeu.
- `System.out.println(counter);`: Cette ligne imprime la valeur de la variable `counter` à la console. Cela pourrait être utile pour vérifier combien d'éléments spécifiques ont été détectés dans le niveau.
- `timer = new Timer(1000, this);`: Ici, un nouvel objet `Timer` est créé avec une durée de 1000 millisecondes (1 seconde) et l'objet actuel (`this`) est spécifié comme l'objet qui recevra les événements de minuterie.
- `timer.start();`: Cette ligne démarre effectivement le timer, déclenchant les événements de minuterie à intervalles réguliers.

En résumé, cette partie du code effectue des initialisations importantes pour la gestion des niveaux du jeu, la détection d'éléments spécifiques dans le niveau, et la mise en place d'un timer pour déclencher des événements à intervalles réguliers.

```

private void showIntroScreen(Graphics2D g2d) {

    String start = "Press SPACE to start, Use direction KEY to move.";
    g2d.setColor(Color.yellow);
    g2d.drawString(start, (SCREEN_SIZE)/30, 150);
}

private void drawScore(Graphics2D g) {
    g.setFont(smallFont);
    g.setColor(new Color(5, 181, 79));
    String s = "Score: " + score;
    g.drawString(s, SCREEN_SIZE / 2 + 96, SCREEN_SIZE + 16);

    for (int i = 0; i < lives; i++) {
        g.drawImage(heart, i * 28 + 8, SCREEN_SIZE + 1, this);
    }
}

```

### **Méthode `showIntroScreen(Graphics2D g2d)`**

Définition de la chaîne de démarrage (`start`) : La variable `start` est définie comme une chaîne de caractères contenant les instructions pour démarrer le jeu. Cela indique au joueur de presser la touche ESPACE pour commencer et d'utiliser les touches directionnelles pour se déplacer.

Définition de la couleur du texte : La couleur du texte est configurée en jaune en utilisant `g2d.setColor(Color.yellow)`. Cela détermine la couleur dans laquelle le texte sera affiché à l'écran.

Affichage du texte : La méthode `g2d.drawString(start, (SCREEN\_SIZE)/30, 150)` dessine la chaîne `start` à une position spécifique sur l'écran. Les coordonnées `(SCREEN\_SIZE)/30, 150` déterminent où le texte sera affiché. `(SCREEN\_SIZE)/30` représente la position horizontale et `150` la position verticale.

## **Méthode `drawScore(Graphics2D g)`**

Définition de la police de caractères : La police de caractères est définie comme ``smallFont``, qui est une instance de la classe ``Font``. Cela spécifie la police, le style (gras) et la taille (14) du texte qui sera affiché.

Définition de la couleur du texte : La couleur du texte est configurée en vert en utilisant ``g.setColor(new Color(5, 181, 79))``. Cela détermine la couleur dans laquelle le texte du score sera affiché.

Affichage du score : Une chaîne de caractères ``s`` est créée, contenant le texte "Score: " suivi du score actuel du joueur. Cette chaîne est affichée à une position spécifique sur l'écran à l'aide de la méthode ``g.drawString(s, SCREEN_SIZE / 2 + 96, SCREEN_SIZE + 16)``. Les coordonnées ``(SCREEN_SIZE / 2 + 96, SCREEN_SIZE + 16)`` déterminent où le texte sera affiché.

Affichage des vies restantes: Une boucle est utilisée pour afficher une image de cœur pour chaque vie restante du joueur. L'image de cœur est dessinée à des positions spécifiques sur l'écran pour chaque vie à l'aide de ``g.drawImage(heart, i * 28 + 8, SCREEN_SIZE + 1, this)``. Cela place les cœurs côte à côte à une position en-dessous du score.

Cette partie du code comporte deux méthodes importantes : ``checkMaze()`` et ``death()``. Explorons-les en détail :

### **Méthode `checkMaze()`**

- Initialisation des variables : On commence par initialiser `i` à zéro et `finished` à `true`. `i` sera utilisé pour parcourir les cases de la grille, et `finished` servira à indiquer si le labyrinthe est entièrement parcouru sans obstacle.
- Boucle de vérification : On utilise une boucle `while` pour parcourir toutes les cases de la grille. La boucle s'exécute tant que `i` est inférieur au nombre total de blocs dans le labyrinthe et que `finished` est `true`.
- Vérification des cases : À chaque itération de la boucle, on vérifie si la case actuelle de la grille (`screenData[i]`) est différente de zéro. Si c'est le cas, cela signifie qu'il y a un obstacle dans le labyrinthe et donc le labyrinthe n'est pas terminé. Dans ce cas, `finished` est mis à `false`.
- Mise à jour du score et du jeu : Après la boucle, on vérifie si le labyrinthe est terminé (c'est-à-dire si `finished` est toujours `true`). Si c'est le cas, cela signifie que le joueur a parcouru tout le labyrinthe sans rencontrer d'obstacles. On augmente alors le score de 50 points. On vérifie également s'il y a moins de fantômes que le maximum autorisé (`MAX\_GHOSTS`). Si oui, on augmente le nombre de fantômes. Ensuite, on vérifie si la vitesse actuelle (`currentSpeed`) est inférieure à la vitesse maximale (`maxSpeed`). Si oui, on augmente la vitesse. Enfin, on initialise un nouveau niveau de jeu avec la méthode `initLevel()`.

### **Méthode `death()`**

- Décrémenter des vies : À chaque fois que le joueur meurt, le nombre de vies (`lives`) est décrémenté.
- Vérification de la fin de partie : Si le nombre de vies atteint zéro, cela signifie que le joueur a perdu toutes ses vies et donc la partie est terminée. Dans ce cas, la variable `inGame` est mise à `false`.
- Continuation du niveau : Après la perte d'une vie, le jeu continue avec la méthode `continueLevel()`, où le joueur est réinitialisé à sa position de départ dans le niveau en cours.

Cette méthode gère ce qui se passe lorsque le personnage du joueur meurt dans le jeu. Voici une explication détaillée :

```
130     }
131     continueLevel();
132     }
133     lives = 1;
134     if (lives == 0) {
135         lives--;
136         break;
137     }
138     continueLevel(); }
```

- Décrémenter le nombre de vies: À chaque fois que le personnage du joueur meurt, le nombre de vies restantes est décrémenté d'une unité à l'aide de 'lives--'. Cela signifie que le joueur perd une vie à chaque fois que cette méthode est appelée.
- Vérification du nombre de vies restantes: Ensuite, il est vérifié si le joueur n'a plus de vies restantes en vérifiant si 'lives' est égal à zéro. Si c'est le cas, cela signifie que le joueur a perdu toutes ses vies.
- Mise à jour du statut du jeu : Si le joueur n'a plus de vies restantes, la variable 'inGame' est définie sur 'false'. Cela indique que le jeu est terminé et que le joueur a perdu.
- Continuation au niveau suivant : Quelle que soit la situation (que le joueur ait encore des vies restantes ou non), la méthode 'continueLevel()' est appelée. Cela permet de redémarrer le niveau actuel ou de passer au niveau suivant, selon la mise en œuvre spécifique du jeu.

En résumé, cette méthode s'occupe de mettre à jour le statut du jeu lorsqu'un joueur perd une vie, en décrémentant le nombre de vies et en vérifiant s'il en reste encore. Si le joueur n'a plus de vies, le jeu se termine, sinon, le jeu continue au niveau suivant ou redémarre le niveau actuel.

### Méthode 'moveGhosts()'

- Cette méthode est responsable du déplacement des fantômes dans le jeu. Voici une explication détaillée :
- Boucle pour chaque fantôme\*\* : La méthode itère sur chaque fantôme présent dans le jeu, de 0 à 'N\_GHOSTS - 1'.
- Vérification de la position du fantôme\*\* : Pour chaque fantôme, il est vérifié si sa position en pixels ('ghost\_x' et 'ghost\_y') est un multiple de la taille d'un bloc ('BLOCK\_SIZE'). Cela garantit que le fantôme se déplace de manière fluide d'un bloc à l'autre et ne reste pas bloqué entre deux blocs.



- Détermination de la position dans la grille de jeu\*\* : La position du fantôme dans la grille de jeu est calculée en fonction de sa position en pixels. Cela permet de déterminer la case de la grille dans laquelle se trouve actuellement le fantôme.
- Calcul des mouvements valides\*\* : En fonction de la position du fantôme dans la grille, les mouvements valides sont déterminés en vérifiant les murs et les obstacles. Les directions dans lesquelles le fantôme peut se déplacer sans rencontrer d'obstacles sont ajoutées à un tableau `dx` et `dy`.
- Choix aléatoire d'une direction\*\* : Si plusieurs directions sont valides, une direction est choisie de manière aléatoire parmi celles-ci.
- Déplacement du fantôme\*\* : La position du fantôme est mise à jour en fonction des directions choisies et de la vitesse du fantôme (`ghostSpeed`). Les nouvelles positions sont calculées en ajoutant ou en soustrayant les composantes `dx` et `dy` à la position actuelle du fantôme.
- Vérification de collision avec Pac-Man\*\* : Il est vérifié si la position actuelle de Pac-Man (`pacman\_x`, `pacman\_y`) se trouve à proximité d'un fantôme. Si tel est le cas et que le jeu est en cours (`inGame`), le booléen `dying` est défini sur `true`, ce qui signifie que Pac-Man est en train de mourir.

En résumé, cette méthode gère le déplacement des fantômes en vérifiant les collisions avec les murs et les autres éléments du jeu, et en mettant à jour leur position en conséquence. Elle vérifie également s'il y a une collision entre un fantôme et Pac-Man, ce qui déclenche le processus de mort de Pac-Man dans le jeu

### **Méthode drawGhost(Graphics2D g2d, int x, int y)**

- Cette méthode est responsable du dessin d'un fantôme à une position spécifique sur l'interface graphique du jeu.
- Elle prend en paramètres un objet `Graphics2D` pour le dessin et les coordonnées `x` et `y` où le fantôme doit être dessiné.
- En utilisant l'objet `g2d`, la méthode dessine l'image du fantôme à la position donnée.
- Cette méthode est appelée à chaque itération du jeu pour redessiner les fantômes à leurs nouvelles positions.

## **Méthode `movePacman()`**

- Cette méthode gère le déplacement du personnage principal, Pac-Man.
- Elle commence par vérifier si Pac-Man est en train de se déplacer entre deux blocs en vérifiant si ses coordonnées en pixels (`pacman\_x` et `pacman\_y`) sont des multiples de la taille d'un bloc (`BLOCK\_SIZE`).
- Ensuite, elle calcule la position actuelle de Pac-Man dans la grille de jeu en fonction de ses coordonnées en pixels.
- La méthode vérifie ensuite si Pac-Man entre en collision avec des points de nourriture en vérifiant si la valeur du point de la grille correspond à un point de nourriture.
- Si Pac-Man mange un point de nourriture, le score est mis à jour, et s'il a mangé tous les points de nourriture de la grille, le niveau est changé.
- Ensuite, la méthode vérifie si Pac-Man peut changer de direction en fonction des entrées du joueur et des murs dans la grille.
- Enfin, elle met à jour les coordonnées de Pac-Man en fonction de sa direction de déplacement actuelle (`pacmand\_x` et `pacmand\_y`) et de sa vitesse (`PACMAN\_SPEED`), puisque Pac-Man se déplace à une vitesse constante.
- Cette méthode est appelée à chaque itération du jeu pour mettre à jour la position de Pac-Man et vérifier les interactions avec les éléments de la grille.

## La méthode `drawMaze(Graphics2D g2d)` :

```
356 private void drawMaze(Graphics2D g2d) {
357
358     short i = 0;
359     int x, y;
360
361     for (y = 0; y < SCREEN_SIZE; y += BLOCK_SIZE) {
362         for (x = 0; x < SCREEN_SIZE; x += BLOCK_SIZE) {
363
364             g2d.setColor(new Color(0,72,251));
365             g2d.setStroke(new BasicStroke(5));
366
367             if ((le.levelData1[i]== 0)) {
368                 g2d.fillRect(x, y, BLOCK_SIZE, BLOCK_SIZE);
369             }
370
371             if ((screenData[i] & 1) != 0) {
372                 g2d.drawLine(x, y, x, y + BLOCK_SIZE - 1);
373             }
374
375             if ((screenData[i] & 2) != 0) {
376                 g2d.drawLine(x, y, x + BLOCK_SIZE - 1, y);
377             }
378         }
379     }
380 }
```

- Cette méthode est responsable du dessin du labyrinthe sur l'interface graphique du jeu.
- Elle utilise un double parcours `for` pour parcourir chaque case du labyrinthe.
- Pour chaque case, elle vérifie les murs qui l'entourent et dessine les lignes appropriées pour représenter les murs.
- Elle commence par initialiser les variables `i`, `x` et `y`, où `i` est l'indice de la case dans les tableaux `screenData` et `le.levelData1`, et `x` et `y` sont les coordonnées de la case à dessiner.
- Ensuite, elle utilise différentes couleurs et épaisseurs de trait pour dessiner les murs et les points de nourriture.
- Elle vérifie les valeurs des bits dans `screenData[i]` pour déterminer quels murs doivent être dessinés autour de la case.
- Si la valeur de `le.levelData1[i]` est zéro, cela signifie qu'il n'y a pas de mur, et la méthode dessine un rectangle plein pour représenter une case vide.
- Si un bit spécifique dans `screenData[i]` est défini, cela signifie qu'un mur est présent dans cette direction, et la méthode dessine une ligne pour représenter ce mur.
- Si le bit 16 est défini dans `screenData[i]`, cela signifie qu'il y a un point de nourriture à cet emplacement, et la méthode dessine un cercle pour le représenter.
- Cette méthode est appelée à chaque itération du jeu pour redessiner le labyrinthe avec les mises à jour nécessaires.

### la méthode `initGame()` :

```
399 private void initGame() {  
400     lives = 3;  
401     initLevel();  
402     N_GHOSTS = 6;  
403     currentSpeed = 3;  
404 }  
405
```

- Cette méthode est chargée d'initialiser les paramètres du jeu au début d'une nouvelle partie.
- Elle fixe le nombre de vies du joueur à 3 en attribuant la valeur `3` à la variable `lives`.
- Ensuite, elle initialise le niveau en appelant la méthode `initLevel()`, qui prépare le labyrinthe et place les éléments de jeu comme les points de nourriture et les fantômes.
- Elle définit également le nombre de fantômes à 6 en attribuant la valeur `6` à la variable `N\_GHOSTS`.
- Enfin, elle fixe la vitesse actuelle du jeu à 3 en attribuant la valeur `3` à la variable `currentSpeed`.
- Cette méthode est appelée au début d'une nouvelle partie pour configurer le jeu avec les paramètres initiaux.

### la méthode `initLevel()` :

```
406  
407 private void initLevel() {  
408  
409     int i;  
410     for (i = 0; i < N_BLOCKS * N_BLOCKS; i++) {  
411         screenData[i] = le.levelData1[i];  
412     }  
413  
414     continueLevel();  
415 }  
416
```

- Cette méthode est responsable de l'initialisation d'un nouveau niveau dans le jeu.
- Elle parcourt chaque case du tableau `screenData` qui représente le labyrinthe et initialise son contenu en fonction des données de niveau stockées dans `le.levelData1`.
- La boucle `for` parcourt chaque élément du tableau `screenData` et lui attribue la valeur correspondante dans `le.levelData1`.
- Une fois que toutes les cases du labyrinthe ont été initialisées avec les données du niveau en cours, la méthode `continueLevel()` est appelée pour démarrer le niveau.
- Cette méthode est utilisée pour préparer le labyrinthe avec les données spécifiques du niveau en cours, ce qui permet de démarrer un nouveau niveau avec les bonnes configurations.

## la méthode `continueLevel()` :

```
416
417 private void continueLevel() {
418
419     int dx = 1;
420     int random;
421
422     for (int i = 0; i < N_GHOSTS; i++) {
423
424         ghost_y[i] = 4 * BLOCK_SIZE; //start position
425         ghost_x[i] = 4 * BLOCK_SIZE;
426         ghost_dy[i] = 0;
427         ghost_dx[i] = dx;
428         dx = -dx;
429         random = (int) (Math.random() * (currentSpeed + 1));
430
431         if (random > currentSpeed) {
432             random = currentSpeed;
433         }
434
435         ghostSpeed[i] = validSpeeds[random];
436     }
437
438     pacman_x = 7 * BLOCK_SIZE; //start position
439     pacman_y = 11 * BLOCK_SIZE;
440     pacmand_x = 0; //reset direction move
441     pacmand_y = 0;
442     req_dx = 0; // reset direction controls
443     req_dy = 0;
444     dying = false;
445 }
446
```

- Cette méthode est chargée de poursuivre le niveau actuel après que le joueur ait perdu une vie ou qu'un nouveau niveau ait été initialisé.
- Elle initialise les positions des fantômes et de Pac-Man pour commencer le niveau.
- La position de départ de chaque fantôme est fixée à  $4 * \text{'BLOCK\_SIZE'}$ , ce qui représente une position prédéfinie dans le labyrinthe.
- Les variables `ghost\_dx` et `ghost\_dy` contrôlent la direction dans laquelle chaque fantôme se déplace initialement.
- La vitesse de chaque fantôme est déterminée aléatoirement en fonction de la vitesse actuelle du jeu.
- Les positions de départ de Pac-Man sont également initialisées.
- Les variables `pacmand\_x` et `pacmand\_y` sont réinitialisées pour arrêter tout mouvement de Pac-Man.
- Les variables `req\_dx` et `req\_dy` sont réinitialisées pour arrêter tout contrôle de direction.
- La variable `dying` est réinitialisée à `false` pour indiquer que Pac-Man n'est pas en train de mourir.

Ce code est une méthode qui dessine différents éléments sur un composant graphique dans une application en utilisant la bibliothèque Java Swing.

```
448 public void paintComponent(Graphics g) {
449     super.paintComponent(g);
450
451     Graphics2D g2d = (Graphics2D) g;
452
453     g2d.setColor(Color.black);
454     g2d.fillRect(0, 0, screenSize.width, screenSize.height);
455
456     drawMaze(g2d);
457     drawScore(g2d);
458
459     if (inGame) {
460         playGame(g2d);
461     } else {
462         showIntroScreen(g2d);
463     }
464
465     Toolkit.getDefaultToolkit().sync();
466     g2d.dispose();
467 }
```

- `public void paintComponent(Graphics g) {`: Cette méthode est une méthode de la classe qui hérite de `JComponent`, elle est utilisée pour dessiner les composants graphiques personnalisés. Elle prend un objet `Graphics` comme argument et ne retourne rien.
- `super.paintComponent(g);`: Cette ligne appelle la méthode `paintComponent` de la classe mère, ce qui permet de dessiner les composants de base de manière appropriée avant d'ajouter notre propre dessin.
- `Graphics2D g2d = (Graphics2D) g;`: Elle effectue une conversion de type de l'objet `Graphics` en un objet `Graphics2D`, ce qui permet d'utiliser des fonctionnalités avancées de dessin 2D.
- `g2d.setColor(Color.black);`: Cela définit la couleur de dessin actuelle à noir.
- `g2d.fillRect(0, 0, screenSize.width, screenSize.height);`: Cela remplit un rectangle noir qui couvre tout le composant graphique avec les dimensions de l'écran.
- `drawMaze(g2d);`: Cette méthode dessine le labyrinthe sur le composant graphique en utilisant l'objet `Graphics2D` passé en paramètre.
- `drawScore(g2d);`: Cette méthode dessine le score sur le composant graphique en utilisant l'objet `Graphics2D` passé en paramètre.
- `if (inGame) { playGame(g2d); } else { showIntroScreen(g2d); }`: Cette structure conditionnelle vérifie si le jeu est en cours (`inGame` est vrai) ou non. Si le jeu est en

cours, elle appelle la méthode `playGame()` pour dessiner les éléments du jeu, sinon elle appelle `showIntroScreen()` pour afficher l'écran d'introduction.

- `Toolkit.getDefaultToolkit().sync();`: Cette ligne synchronise l'état de l'interface utilisateur avec l'état de l'application, ce qui peut améliorer les performances.
- `g2d.dispose();`: Cette ligne libère les ressources utilisées par l'objet `Graphics2D`, ce qui est une bonne pratique pour éviter les fuites de mémoire.

Ce code déclare une classe TAdapter qui étend KeyAdapter, une classe de base pour la gestion des événements liés aux touches du clavier. Cette classe remplace la méthode keyPressed pour réagir aux événements de pression de touche.

```
class TAdapter extends KeyAdapter {

    @Override
    public void keyPressed(KeyEvent e) {
        // Récupère le code de la touche pressée
        int key = e.getKeyCode();

        // Vérifie si le jeu est en cours
        if (inGame) {
            // Si le jeu est en cours, gère les mouvements du personnage
            if (key == KeyEvent.VK_LEFT) {
                // Si la touche gauche est pressée, déplace le personnage vers la gauche
                req_dx = -1;
                req_dy = 0;
            } else if (key == KeyEvent.VK_RIGHT) {
                // Si la touche droite est pressée, déplace le personnage vers la droite
                req_dx = 1;
                req_dy = 0;
            } else if (key == KeyEvent.VK_UP) {
                // Si la touche haut est pressée, déplace le personnage vers le haut
                req_dx = 0;
                req_dy = -1;
            } else if (key == KeyEvent.VK_DOWN) {
                // Si la touche bas est pressée, déplace le personnage vers le bas
                req_dx = 0;
                req_dy = 1;
            } else if (key == KeyEvent.VK_ESCAPE && timer.isRunning()) {
                // Si la touche Échap est pressée et le jeu est en cours, arrête le jeu
                inGame = false;
            }
        } else {
            // Si le jeu n'est pas en cours, gère l'événement de pression de la touche espace
        }
    }
}
```



```

    if (key == KeyEvent.VK_SPACE) {
        // Si la touche espace est pressée, commence un nouveau jeu
        inGame = true;
        // Initialise le jeu
        initGame();
    }
}
}
}

```

Cette classe intercepte les événements de pression de touche et ajuste les variables de contrôle du mouvement (req\_dx et req\_dy) du personnage en fonction de la touche pressée. Elle vérifie également si le jeu est en cours (inGame) pour décider des actions appropriées, comme démarrer un nouveau jeu lorsque la touche espace est pressée.

Ce code représente une méthode actionPerformed qui est appelée lorsqu'un événement d'action se produit, généralement associé à des composants graphiques comme des boutons. Dans ce cas, il est probable que cette méthode soit utilisée pour redessiner la vue graphique du jeu.

```

@Override
public void actionPerformed(ActionEvent e) {
    repaint();
}
}

```

Lorsque cette méthode est invoquée, elle demande à l'élément graphique associé de se redessiner. Cela peut être utile dans un jeu pour mettre à jour l'affichage en fonction des changements d'état du jeu ou des actions des joueurs.

## 8.3 Script Level.java

Code folder : `C:\Users\----\eclipse-workspace\Pacman\src\pacman\Level.java`

La classe `levelData1`

```
1 package pacman;
2
3 public class level {
4     public final short levelData1[]={
5         19, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,
6         17, 16, 16, 16, 16, 16, 24, 16, 16, 16, 16, 16, 16, 16, 20,
7         25, 24, 24, 24, 28, 0, 17, 16, 16, 16, 16, 16, 16, 16, 20,
8         0, 0, 0, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 20,
9         19, 18, 18, 18, 18, 18, 16, 16, 16, 16, 24, 24, 24, 24, 20,
10        17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0, 0, 0, 21,
11        17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0, 0, 0, 21,
12        17, 16, 16, 16, 24, 16, 16, 16, 16, 20, 0, 0, 0, 21,
13        17, 16, 16, 20, 0, 17, 16, 16, 16, 16, 18, 18, 18, 18, 20,
14        17, 24, 24, 28, 0, 25, 24, 24, 16, 16, 16, 16, 16, 16, 20,
15        21, 0, 0, 0, 0, 0, 0, 0, 0, 17, 16, 16, 16, 16, 16, 20,
16        17, 18, 18, 22, 0, 19, 18, 18, 16, 16, 16, 16, 16, 16, 20,
17        17, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 20,
18        17, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 20,
19        25, 24, 24, 24, 26, 24, 24, 24, 24, 24, 24, 24, 24, 24, 28
20    };
21 }
```

Ce code définit un tableau `levelData1` qui représente la disposition des éléments du niveau dans le jeu Pacman. Chaque nombre dans le tableau correspond à un élément spécifique du niveau. Voici une explication de chaque nombre :

- 0: Indique une case vide.
- 16: Indique un mur horizontal.
- 17: Indique un mur vertical.
- 18: Indique un mur en bas à droite.
- 19: Indique un mur en bas à gauche.
- 20: Indique un mur en haut à droite.
- 21: Indique un mur en haut à gauche.
- 22: Indique un coin en bas à droite.
- 24: Indique un coin en bas à gauche.
- 25: Indique un coin en haut à droite.
- 26: Indique un coin en haut à gauche.
- 28: Indique un point de pouvoir.

Ces nombres sont utilisés pour créer la carte de jeu Pacman. Chaque nombre correspond à un type de tuile sur laquelle le joueur et les fantômes peuvent se déplacer ou interagir.

Dans le tableau `levelData1`, chaque ligne représente une rangée de la carte, et chaque nombre dans une rangée représente une tuile dans cette rangée.

Le jeu peut utiliser ces données pour afficher graphiquement la carte et gérer les collisions entre les différents éléments du jeu. Par exemple, les murs empêchent le joueur de traverser, tandis que les points de pouvoir peuvent être collectés pour gagner des points.

## La classe `levelData2`

```
21
22 public final short levelData2[] = {
23     19, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,
24     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
25     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
26     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
27     17, 24, 24, 24, 24, 16, 16, 16, 16, 16, 24, 24, 24, 24, 20,
28     21, 0, 0, 0, 0, 17, 16, 16, 16, 20, 0, 0, 0, 0, 21,
29     21, 0, 0, 0, 0, 17, 16, 16, 16, 20, 0, 0, 0, 0, 21,
30     21, 0, 0, 0, 0, 17, 16, 16, 16, 20, 0, 0, 0, 0, 21,
31     21, 0, 0, 0, 0, 17, 16, 16, 16, 20, 0, 0, 0, 0, 21,
32     17, 18, 18, 18, 18, 16, 16, 16, 16, 16, 18, 18, 18, 18, 20,
33     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
34     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
35     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
36     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
37     25, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 28
38 };
39
```

Ce code définit un autre tableau appelé `levelData2`, qui représente une autre disposition des éléments du niveau dans le jeu Pacman. Les éléments du niveau sont représentés par des nombres entiers dans ce tableau, de manière similaire à `levelData1`.

Comme `levelData1`, chaque nombre dans `levelData2` représente un type spécifique de tuile sur laquelle le joueur et les fantômes peuvent se déplacer ou interagir.

`levelData2` semble représenter une autre disposition de niveau dans le jeu Pacman par rapport à `levelData1`. Les données sont organisées en lignes et colonnes pour former la carte de jeu. Chaque ligne représente une rangée de la carte, et chaque nombre dans une rangée représente une tuile dans cette rangée.

## La classe `levelData3`

```
39
40 public final short levelData3[] = {
41     19, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,
42     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
43     17, 16, 16, 16, 24, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
44     17, 24, 24, 28, 0, 25, 24, 24, 16, 16, 16, 16, 16, 16, 20,
45     21, 0, 0, 0, 0, 0, 0, 0, 17, 16, 16, 16, 16, 16, 20,
46     17, 18, 18, 22, 0, 19, 18, 18, 16, 16, 16, 16, 16, 16, 20,
47     17, 16, 16, 16, 18, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
48     17, 24, 24, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
49     21, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 24, 24, 24, 20,
50     17, 30, 0, 17, 16, 16, 16, 16, 16, 20, 0, 0, 0, 0, 21,
51     21, 0, 19, 16, 16, 16, 16, 16, 16, 28, 0, 27, 18, 18, 20,
52     21, 0, 25, 16, 16, 16, 16, 16, 20, 0, 0, 0, 17, 16, 20,
53     17, 22, 0, 17, 16, 16, 16, 16, 16, 18, 18, 18, 16, 16, 20,
54     17, 20, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,
55     25, 24, 26, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 28
56 };
```

Ce code définit un tableau appelé `levelData3`, qui représente une disposition spécifique des éléments du niveau dans le jeu Pacman. Chaque nombre dans ce tableau correspond à un élément spécifique du niveau.

Comme pour `levelData1` et `levelData2`, chaque nombre dans `levelData3` représente un type spécifique de tuile sur laquelle le joueur et les fantômes peuvent se déplacer ou interagir.

Le niveau représenté par `levelData3` semble être plus complexe que les précédents, avec des murs, des coins, et des points de pouvoir disposés de manière à créer un niveau de jeu plus intéressant et plus difficile. Les données sont organisées en lignes et colonnes pour former la carte de jeu, avec chaque nombre représentant une tuile dans cette carte.

## La classe `levelData4`

```
57-    },  
58    public final short levelData4[] = {  
59        19, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,  
60        17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,  
61        17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,  
62        17, 24, 16, 16, 24, 16, 24, 16, 16, 24, 16, 16, 24, 16, 20,  
63        21, 0, 17, 21, 0, 21, 0, 17, 20, 0, 17, 20, 0, 17, 20,  
64        21, 0, 17, 21, 0, 21, 0, 17, 20, 0, 17, 20, 0, 17, 20,  
65        21, 0, 17, 21, 0, 21, 0, 17, 20, 0, 17, 20, 0, 17, 20,  
66        21, 0, 17, 21, 0, 21, 0, 17, 20, 0, 17, 20, 0, 17, 20,  
67        21, 0, 17, 21, 0, 21, 0, 17, 20, 0, 17, 20, 0, 17, 20,  
68        17, 18, 16, 16, 18, 16, 18, 16, 16, 18, 16, 16, 18, 16, 20,  
69        17, 16, 16, 16, 24, 24, 24, 24, 24, 24, 24, 24, 24, 16, 20,  
70        17, 16, 16, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 20,  
71        17, 16, 16, 16, 18, 18, 18, 18, 18, 18, 18, 18, 18, 16, 20,  
72        25, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 24, 28  
73    };
```

Ce code définit un tableau appelé `levelData4`, qui représente une disposition spécifique des éléments du niveau dans le jeu Pacman. Chaque nombre dans ce tableau correspond à un élément spécifique du niveau.

Comme dans les autres exemples, chaque nombre dans `levelData4` représente un type spécifique de tuile sur laquelle le joueur et les fantômes peuvent se déplacer ou interagir.

Dans `levelData4`, la disposition des éléments du niveau est conçue de manière à former une structure de labyrinthe complexe et labyrinthique, avec des murs et des coins qui créent des passages étroits et des zones fermées. Cette disposition rendra le jeu plus difficile pour le joueur, car il devra naviguer à travers le labyrinthe pour éviter les fantômes et collecter les points de pouvoir.

## La classe `levelData5`

```
74 public final short levelData5[] = {  
75     19, 18, 18, 18, 18, 18, 18, 18, 26, 26, 26, 26, 26, 26, 22,  
76     17, 16, 16, 16, 16, 16, 16, 16, 20, 0, 0, 0, 0, 0, 21,  
77     17, 16, 16, 24, 16, 16, 16, 16, 18, 18, 18, 18, 18, 18, 20,  
78     17, 16, 20, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,  
79     17, 24, 28, 0, 17, 16, 16, 16, 16, 16, 24, 24, 24, 24, 20,  
80     21, 0, 0, 0, 17, 16, 16, 16, 16, 20, 0, 0, 0, 0, 21,  
81     17, 18, 18, 18, 16, 16, 16, 16, 16, 20, 0, 0, 0, 0, 21,  
82     17, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0, 0, 0, 0, 21,  
83     17, 16, 16, 20, 16, 16, 16, 16, 16, 16, 18, 18, 18, 18, 20,  
84     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,  
85     17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,  
86     17, 18, 18, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,  
87     17, 16, 24, 24, 24, 24, 24, 16, 16, 16, 16, 16, 16, 16, 20,  
88     17, 20, 0, 0, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 20,  
89     25, 24, 26, 26, 26, 26, 26, 24, 24, 24, 24, 24, 24, 24, 28  
90 };  
91  
92 }
```

Ce code définit un tableau appelé `levelData5`, qui représente une disposition spécifique des éléments du niveau dans le jeu Pacman. Chaque nombre dans ce tableau correspond à un élément spécifique du niveau.

Chaque nombre dans `levelData5` représente un type spécifique de tuile sur laquelle le joueur et les fantômes peuvent se déplacer ou interagir.

Dans `levelData5`, la disposition des éléments du niveau est conçue pour former un labyrinthe avec des murs, des coins et des passages étroits. Il y a également des points de pouvoir et des cases vides pour permettre au joueur de se déplacer. La disposition spécifique des murs et des coins rendra le niveau de jeu plus complexe et stimulant pour le joueur.

## 9.0 Testing

Le processus de test dans le développement d'un jeu tel que Pacman est crucial pour garantir la qualité, la jouabilité et la fiabilité du jeu. Voici quelques étapes importantes dans le processus de test du jeu Pacman :

### 9.1 Test unitaire

Les composants individuels du jeu, tels que les mouvements de Pacman et des fantômes, la détection de collision, la collecte de points, doivent être testés individuellement pour s'assurer qu'ils fonctionnent correctement.

### 9.2 Test d'intégration

Une fois que les composants individuels sont testés, ils doivent être intégrés pour vérifier leur interaction correcte. Cela implique de tester la manière dont Pacman interagit avec les fantômes, la manière dont les points sont collectés, etc.

### 9.3 Test de régression

Après chaque modification du code source, il est important de réexécuter tous les tests pour s'assurer qu'aucun nouveau bogue n'a été introduit et que les fonctionnalités existantes n'ont pas été affectées.

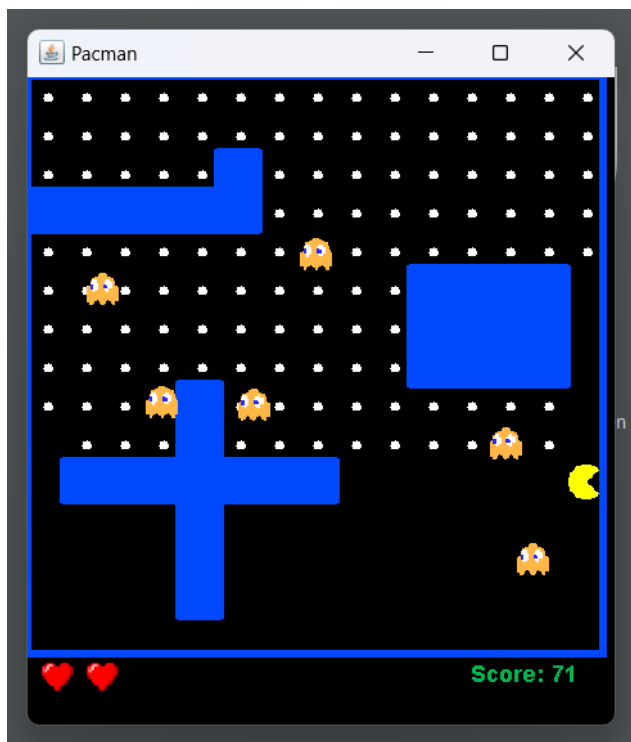
### 9.4 Test de performance

Le jeu doit être testé pour s'assurer qu'il fonctionne de manière fluide et réactive, même lorsqu'il y a de nombreux éléments à l'écran. Cela inclut le test de la vitesse de déplacement de Pacman et des fantômes, ainsi que la fluidité de l'animation.

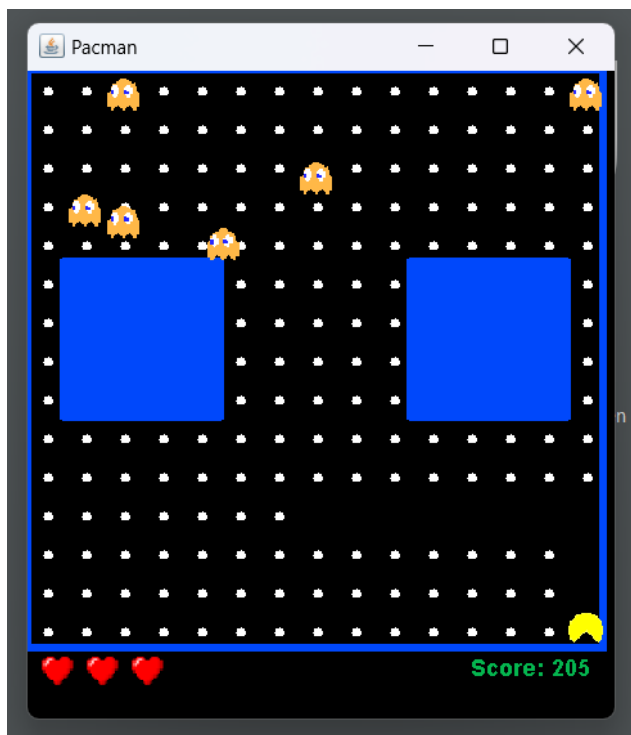
En mettant en œuvre ces différentes étapes de test de manière systématique et rigoureuse, les développeurs peuvent s'assurer que le jeu Pacman est exempt de bugs et offre une expérience de jeu optimale pour les joueurs.

## 9.5 Résultats des tests et tout bug ou problème découvert

- Capture d'écran du niveau 1



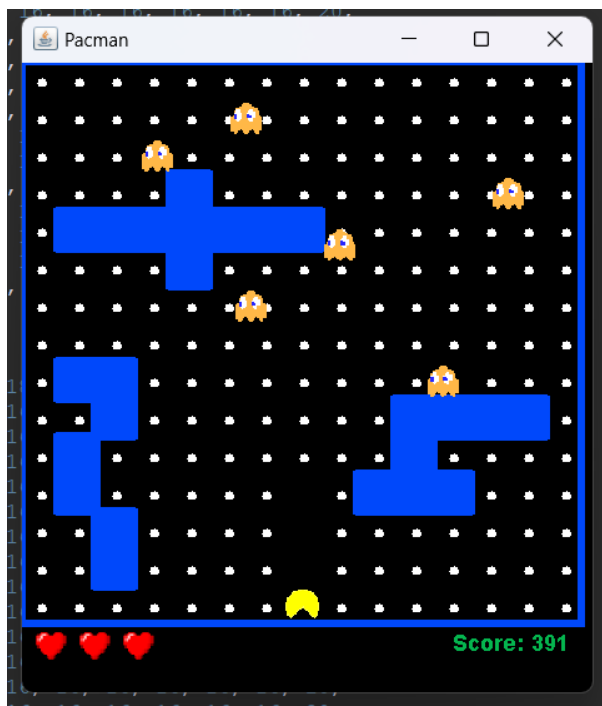
- Capture d'écran du niveau 2



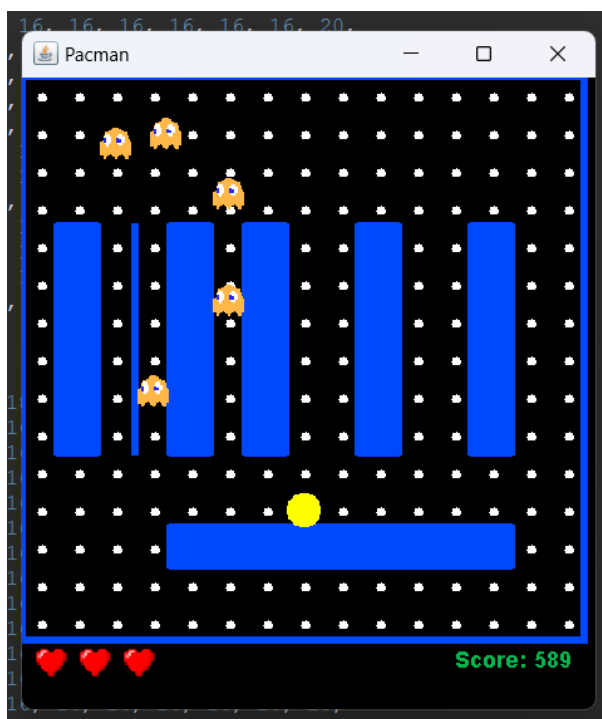


## 9.6 Capture d'écran du test

- Capture d'écran du niveau 3



- Capture d'écran du niveau 4



## 10.0 Conclusion

En conclusion, le jeu Pacman développé en Java avec Eclipse se distingue en offrant une expérience classique et captivante qui transcende les générations. Avec ses cinq niveaux de difficulté croissante, le jeu propose un défi stimulant, adapté à un large éventail de joueurs. L'utilisation de la plateforme Eclipse pour le développement a permis de créer un environnement de développement robuste et efficace.

L'icône emblématique de Pacman, accompagnée des fantômes en tant que personnages secondaires, crée une dynamique de jeu intemporelle. Cette dynamique met à l'épreuve les joueurs, exigeant stratégie, réflexion rapide et réflexes aiguisés. Chaque niveau représente un défi unique, offrant une progression progressive qui maintient l'engagement du joueur.

Atteindre le niveau 5 du jeu représente une étape significative, mettant à l'épreuve les compétences acquises tout au long du parcours. En surmontant les obstacles de chaque niveau, les joueurs développent leur habileté à naviguer dans le labyrinthe, à éviter les fantômes et à collecter les points de pouvoir. Ce défi progressif ajoute une dimension stratégique au jeu, incitant les joueurs à perfectionner leurs compétences au fil du temps.

En fin de compte, Pacman sur Java Eclipse offre une expérience de jeu divertissante et enrichissante, évoquant la nostalgie des jeux d'arcade classiques. Que ce soit pour relever le défi de terminer les cinq niveaux ou simplement pour se divertir, Pacman reste un jeu intemporel qui continuera à captiver les joueurs pendant de nombreuses années à venir. L'utilisation de Java et Eclipse a permis de créer un jeu stable, fonctionnel et mémorable qui témoigne de l'expertise et de l'engagement de l'équipe de développement.

## 11.0 Références

1. “Java Class and Objects (with Example).” *Programiz.com*, 2020, [www.programiz.com/java-programming/class-objects](http://www.programiz.com/java-programming/class-objects).
2. “Java Tutorial.” *GeeksforGeeks*, 20 Feb. 2019, [www.geeksforgeeks.org/java-tutorial/](http://www.geeksforgeeks.org/java-tutorial/).
3. “Pacman Code.” *DaniWeb*, 3 Nov. 2012, [www.daniweb.com/programming/software-development/threads/439009/pacman-code](http://www.daniweb.com/programming/software-development/threads/439009/pacman-code). Accessed 22 Feb. 2024.
4. *Google.com*, 2024, [www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi6zNHGwryEAxXRxQIHHXJoBa4QFnoECB8QAAQ&url=https%3A%2F%2Fwww.tutorialspoint.com%2F eclipse%2Findex.htm&usg=AOvVawlpYHJ-aWp3EpS\\_xU3S74QH&opi=89978449](http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwi6zNHGwryEAxXRxQIHHXJoBa4QFnoECB8QAAQ&url=https%3A%2F%2Fwww.tutorialspoint.com%2F eclipse%2Findex.htm&usg=AOvVawlpYHJ-aWp3EpS_xU3S74QH&opi=89978449). Accessed 22 Feb. 2024.
5. *Google.com*, 2024, [www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiuoYuMw7yEAxVo-gIHHYx3B\\_cQFnoECAgQAQ&url=https%3A%2F%2Fopenjfx.io%2F&usg=AOvVaw2P5dkXL034-ktNP6qMX847&opi=89978449](http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiuoYuMw7yEAxVo-gIHHYx3B_cQFnoECAgQAQ&url=https%3A%2F%2Fopenjfx.io%2F&usg=AOvVaw2P5dkXL034-ktNP6qMX847&opi=89978449). Accessed 22 Feb. 2024.