Killian GRISLIN
Léa COURMONTAGNE
Sara TERRIER

BUT 3 Dev 1.1

Documentation Technique

I. Introduction

- **Présentation du projet :** Le projet consiste à développer une version moderne et fonctionnelle du jeu vidéo classique Pac-Man, tout en intégrant les normes actuelles de codage et les bonnes pratiques de développement logiciel. Pac-Man est un jeu d'arcade emblématique où le joueur contrôle un personnage principal, Pac-Man, qui doit collecter tous les points dans un labyrinthe en évitant des ennemis appelés fantômes. Le développement de ce projet s'inscrit dans une démarche de qualité logicielle, en mettant en œuvre des concepts tels que la modularité du code, les tests automatisés, et la gestion de projet collaboratif via GitHub. Une attention particulière sera portée à la lisibilité et à la documentation du code, afin de garantir une facilité de maintenance et d'évolution.

- **Objectif du projet :** L'objectif principal de ce projet est de recréer une version de Pac-Man à la fois ludique et respectueuse des standards modernes de développement. Le projet vise à fournir une expérience de jeu fidèle à l'original, tout en intégrant des fonctionnalités et des outils adaptés aux exigences actuelles. Plus précisément, les objectifs incluent :
  - Développer une version fonctionnelle de Pac-Man avec les éléments de base tels que le labyrinthe, les déplacements de Pac-Man, les fantômes ennemis, et le système de score.
  - Respecter les normes de codage modernes, en veillant à la structuration, la lisibilité et la documentation du code.
  - Assurer la qualité du logiciel en utilisant des tests unitaires, des tests d'intégration, et des revues de code systématiques via GitHub.
  - Mettre en œuvre une interface utilisateur intuitive, incluant des écrans de menu, de jeu, et de fin de partie.
  - Faciliter le travail collaboratif grâce à une gestion rigoureuse des versions, des branches, et des pull-requests sur GitHub.

II. Architecture générale

- **Langage et environnement de développement :** Dans le cadre de ce projet, nous avons opté pour le langage de programmation Python, accompagné de la bibliothèque Pygame pour la gestion de l'affichage et des éléments graphiques.

- **Structure du code :** Le projet contient, un dossier dist qui contient l'executable, un dossier Sprite qui contient les images du jeu (Pacman, les fantômes…), un dossier tests contenant 4 fichiers python pour les tests unitaires / d'intégrations et 10 fichiers python :

- Matrix.py

Le fichier Matrix.py contient la matrice nommée m_matrice, qui représente la structure du labyrinthe du jeu. Chaque élément de cette matrice correspond à une case spécifique, indiquant si celle-ci est un mur, un chemin, un point collectable, ou un autre élément. Cette matrice est la base sur laquelle s'appuient les autres modules pour déterminer la configuration du labyrinthe. Elle est essentielle pour gérer les positions des éléments et leurs déplacements. En outre, elle sert à dessiner graphiquement le labyrinthe dans le jeu et à placer les éléments comme Pac-Man, les fantômes et les points.

- Labyrinth.py

Le fichier Labyrinth.py définit la classe Labyrinth, responsable de la gestion visuelle et fonctionnelle du labyrinthe. Cette classe utilise la matrice définie dans Matrix.py pour construire et afficher le labyrinthe en jeu. Elle s'appuie sur la bibliothèque Pygame pour dessiner les murs et autres éléments visuels en fonction des cases définies. De plus, le module inclut des méthodes pour vérifier les interactions avec les cases du labyrinthe, telles que la gestion des collisions entre Pac-Man ou les fantômes et les murs. Ce fichier permet également de vérifier les murs autour de chaque bloc pour ajuster la navigation des entités.

- Graph.py

Le fichier Graph.py propose une structure de graphe correspondant au labyrinthe. Ce graphe est une représentation abstraite qui stocke les connexions possibles entre les différentes cases de la matrice. Il indique les positions accessibles et les directions dans lesquelles Pac-Man et les fantômes peuvent se déplacer. Ce module est essentiel pour calculer les chemins possibles et gérer les déplacements stratégiques des fantômes en fonction de leurs comportements. La navigation dans le labyrinthe est facilitée par ce graphe, rendant possible des algorithmes d'intelligence artificielle pour les fantômes.

- Party.py

Le fichier Partie.py définit la classe Partie, qui représente une session complète du jeu. Elle orchestre les interactions entre Pac-Man, les fantômes, le labyrinthe et les mécaniques du jeu. Cette classe initialise le plateau de jeu avec une matrice représentant le labyrinthe (v_matrice) et un graphe pour les déplacements (v_graphe). Elle configure également les entités principales : Pac-Man et les quatre fantômes (Blinky, Pinky, Inky, Clyde) avec leurs positions de départ et leurs sprites. Les méthodes principales sont :

- RedessinerPlateau : Met à jour et affiche l'état visuel du jeu.
- Collision : Gère les collisions entre Pac-Man et les fantômes.
- CollisionPacgum : Traite les interactions de Pac-Man avec les pacgums et active le mode spécial.
- DebutSuperMode et FinSuperMode : Activent et désactivent le mode "super" où Pac-Man peut manger les fantômes.
- DeplacementTiming : Gère le timing des déplacements des fantômes.
- AfficherPopup et AfficherScore : Affichent des messages de fin de partie et le score.

Ce fichier est essentiel pour coordonner l'ensemble des composants du jeu, assurer la logique des événements (déplacements, collisions, conditions de victoire/défaite) et maintenir une expérience de jeu fluide.

- Ghosts.py

Le fichier Ghosts.py définit une classe Ghosts, représentant les fantômes du jeu, chacun avec un comportement spécifique basé sur son nom (Blinky, Pinky, Inky, Clyde). Héritant de la classe Entities, chaque fantôme possède une position initiale (v_spawn), un nom (v_name), et des états particuliers comme v_mort (mort) et v_effraye (effrayé).

Les fantômes ont différents comportements selon leur état :

- Fuite : Lorsqu'ils sont effrayés, ils fuient dans la direction la plus éloignée de Pac-Man grâce à la méthode Fuite.
- MouvementFantomes : Contrôle le déplacement des fantômes en fonction de leur nom et de l'état du jeu (poursuite, fuite ou retour à la base). Chaque fantôme a une stratégie spécifique : Blinky suit directement Pac-Man, Pinky anticipe sa position, Inky mélange stratégie et aléatoire, tandis que Clyde se déplace aléatoirement.
- Meurt : Change l'état du fantôme pour signaler sa mort et le renvoie à sa position initiale.

Ce fichier est essentiel pour gérer le comportement dynamique des fantômes, ajoutant une diversité stratégique au gameplay et influençant directement la difficulté du jeu.

- Pacman.py

Le fichier Pacman.py définit une classe Pacman, qui hérite de la classe Entities. Cette classe représente le personnage principal du jeu, Pacman, et gère ses caractéristiques et ses déplacements. Pacman possède une position initiale (p_x, p_y), un sprite visuel (p_sprite) et un intervalle spécifique entre ses déplacements (p_intervalle). Une propriété supplémentaire, v_super, permet d'indiquer si Pacman est en mode "super" (par exemple, après avoir mangé un power-up). La méthode TesterDeplacement vérifie si Pacman peut se déplacer dans une direction donnée en se basant sur le graphe du labyrinthe (m_graphe). Ce fichier est crucial pour contrôler le comportement spécifique de Pacman et ses interactions avec l'environnement du jeu.

- Game.py

Le fichier Game.py est le point d'entrée principal pour l'exécution du jeu Pac-Man. Il orchestre les différents composants du jeu définis dans d'autres fichiers tels que Partie.py, en gérant la logique principale, les événements utilisateur et la boucle de jeu.

Fonctionnalités principales du fichier :

Initialisation :

- pygame.init() : Initialise la bibliothèque Pygame pour gérer les graphiques, les événements et les sons.

- Partie() : Initialise une nouvelle partie avec le plateau, Pac-Man, les fantômes et les variables associées.
- Gestion des variables de mouvement et de temps : Ces variables permettent de synchroniser les déplacements de Pac-Man et des fantômes.

Gestion des événements utilisateur :

- La boucle for event in pygame.event.get() gère les entrées utilisateur :
    - Quitter le jeu (événement QUIT).
    - Mouvement de Pac-Man (flèches directionnelles).
    - Mettre en pause (touches P ou Échap).
    - Redémarrer ou quitter en cas de partie terminée.
    - Fin du mode spécial (événement personnalisé SUPER_MODE_END).

Logique de déplacement :

- Déplacement de Pac-Man :
    - Vérifie si un mouvement est validé via la méthode TesterDeplacement().
    - Met à jour la position en cas de mouvement valide et gère les interactions avec les pacgums (via CollisionPacgum()).
- Déplacement des fantômes :
    - Utilise la méthode DeplacementTiming() pour synchroniser les déplacements des fantômes en fonction de leur intervalle.

Mise à jour de l'affichage :

- La méthode RedessinerPlateau() de l'objet Partie est appelée pour mettre à jour l'écran à chaque itération de la boucle.

Gestion des collisions :

- Collision() : Vérifie si Pac-Man entre en contact avec un fantôme. Si Pac-Man est en mode spécial, il mange le fantôme, sinon la partie se termine.

Contrôle de la boucle :

- La boucle est limitée à 60 FPS pour garantir une fluidité optimale.

    - FollowInFrontOf.py

Le fichier FollowInFrontOf.py contient plusieurs fonctions qui aident à déterminer la direction que les fantômes doivent prendre pour suivre ou anticiper les mouvements de Pac-Man dans le labyrinthe. Ces fonctions reposent sur des algorithmes de recherche de chemin dans un graphe représentant le labyrinthe, notamment l'algorithme BFS (parcours en largeur) pour trouver le plus court chemin entre deux positions.

- BfsShortestPath : Trouve le plus court chemin entre deux positions dans le graphe du labyrinthe en utilisant une recherche en largeur. Elle retourne une liste de positions représentant ce chemin.

- FollowTarget : Permet à un fantôme de suivre Pac-Man en utilisant BfsShortestPath. Cette fonction trouve le chemin le plus court vers Pac-Man et retourne la direction à prendre pour que le fantôme le suive.
- FindTargetPosition : Trouve la position cible devant Pac-Man en fonction de sa direction. Cette fonction anticipe le déplacement de Pac-Man et détermine où il sera dans quelques cases en fonction de sa direction actuelle.
- GetFantomeDirection : Calcule la direction que le fantôme doit prendre pour se diriger vers une position située devant Pac-Man. Elle utilise la fonction FindTargetPosition pour déterminer la cible devant Pac-Man et la fonction BfsShortestPath pour calculer la direction optimale que le fantôme doit suivre pour atteindre cette cible.

Ce fichier est crucial pour gérer l'intelligence artificielle des fantômes, leur permettant de poursuivre Pac-Man ou de l'anticiper en fonction de la situation du jeu.

- Flee.py

Le fichier Flee.py contient des fonctions utilisées pour déterminer les mouvements des entités qui cherchent à fuir ou s'éloigner d'une cible, dans ce cas, Pac-Man. La fonction principale est FindFurthestFirection, qui calcule la direction la plus éloignée de Pac-Man pour un fantôme, en se basant sur le graphe du labyrinthe (m_graphe). Elle explore les voisins d'une position donnée, calcule la distance entre chaque voisin et Pac-Man, et retourne la direction menant à la position la plus éloignée. La fonction CalculateDistance est utilisée pour calculer la distance euclidienne entre deux positions, ce qui permet à FindFurthestFirection de déterminer avec précision la meilleure direction pour éloigner un fantôme de Pac-Man. Ce fichier est essentiel pour la logique de fuite des fantômes effrayés dans le jeu.

- Entities.py

Le fichier Entities.py définit une classe nommée Entities, représentant les entités mobiles du jeu, telles que les personnages ou objets animés. Chaque entité possède une position (v_x, v_y), une vitesse de déplacement (v_vitesse) et une direction. Elle est associée à un sprite (image) pour son affichage graphique. La classe inclut des méthodes pour afficher l'entité sur l'écran (Affichage), vérifier si elle est alignée sur une case de la grille (SurCase), récupérer sa position sur la grille (GetPosition), gérer son déplacement (Mouvement) et détecter les collisions avec d'autres entités (Collision). Ce fichier est essentiel pour gérer les interactions et les déplacements dynamiques des éléments dans le jeu.

Pour le dossier tests, il contient 4 fichiers pour les tests unitaires (TestPacman.py et TestGhost.py) et les tests d'intégrations (TestParty.py et TestGameInterface.py).

- TestPacman.py

Le fichier TestPacman.py contient des tests unitaires pour vérifier différentes fonctionnalités du jeu Pac-Man en utilisant le framework unittest.

- setUp : Initialise la fenêtre de jeu Pygame et crée une instance de la partie avant chaque test.
- tearDown : Ferme la fenêtre de jeu après chaque test.
- testPacGum : Vérifie que la collision de Pac-Man avec une pacgum augmente le score de 10 points. Le test place une pacgum à une position spécifique, puis vérifie que le score est mis à jour.
- testSuperPacGum : Vérifie la collision avec une super pacgum. Ce test s'assure que Pac-Man entre en mode super et que tous les fantômes deviennent effrayés.
- testCollisionFantome : Vérifie que la collision entre Pac-Man et un fantôme termine la partie. Le test place Pac-Man et un fantôme à la même position et vérifie que le jeu se termine lorsque les deux se percutent.
- testMouvementPacman : Vérifie que Pac-Man se déplace correctement lorsqu'une direction est donnée. Le test simule un mouvement vers la droite et vérifie que les coordonnées de Pac-Man sont mises à jour correctement.

Ces tests permettent de valider les principales interactions du jeu, notamment la collecte de pacgums, les collisions avec les fantômes, et les déplacements de Pac-Man.

- TestGhost.py

Le fichier TestGhost.py contient des tests unitaires pour vérifier le comportement des fantômes dans le jeu Pac-Man. Le test utilise le framework unittest pour s'assurer que les fantômes se déplacent correctement.

- setUp : Initialise la fenêtre de jeu Pygame et crée une instance de la partie avant chaque test.
- tearDown : Ferme la fenêtre de jeu après chaque test.
- testMouvementFantome : Vérifie que le mouvement des fantômes fonctionne correctement. Dans ce test, un fantôme (ici, Blinky) est placé à une position spécifique. Ensuite, le test simule le mouvement du fantôme avec la méthode MouvementFantomes, en lui indiquant une direction (ici, 'DROITE'). Enfin, il vérifie que les coordonnées du fantôme ont changé, ce qui confirme qu'il a bien bougé.

Ce test s'assure que les fantômes réagissent correctement aux instructions de mouvement, garantissant ainsi que leur logique de déplacement dans le jeu fonctionne comme prévu.

- TestParty.py

Le fichier TestParty.py contient un test unitaire pour vérifier le bon fonctionnement d'une partie du jeu Pac-Man en utilisant le framework unittest.

- setUp : Initialise la fenêtre de jeu avec Pygame et crée une instance de la partie avant chaque test.
- tearDown : Ferme la fenêtre de jeu après chaque test.
- testGameOver : Vérifie que la partie se termine correctement. Le test simule la condition de fin de partie en définissant v_finished sur True, puis vérifie que la partie est effectivement terminée avec la méthode assertTrue.

Ce test permet de s'assurer que la condition de fin de jeu est bien gérée et que le jeu reconnaît correctement la fin d'une partie.

- TestGameInterface.py

Le fichier TestGameInterface.py contient des tests unitaires pour vérifier l'interface du jeu et son interaction avec les entrées de l'utilisateur, telles que les événements de clavier.

- setUp : Initialise la fenêtre de jeu avec Pygame et crée une instance de la partie avant chaque test. Ce bloc est exécuté avant chaque test pour préparer l'environnement.
- tearDown : Ferme la fenêtre de jeu après chaque test. Ce bloc est exécuté après chaque test pour nettoyer et fermer l'environnement Pygame.
- testPauseJeu :
  - Ce test simule l'appui sur la touche P pour mettre le jeu en pause.
  - La méthode AfficherPopup est utilisée pour afficher un message de pause.
  - Le test vérifie ensuite si un événement de type KEYDOWN avec la touche P a été enregistré, indiquant que le jeu a bien été mis en pause.
- testReprendreJeu :
  - Ce test simule l'appui sur la touche R pour reprendre le jeu après une pause.
  - Après l'appui sur R, la méthode RedessinerPlateau est appelée pour redessiner le plateau de jeu.
  - Le test vérifie si un événement KEYDOWN avec la touche R est bien enregistré, indiquant que le jeu a été repris.
- testQuitterJeu :
  - Ce test simule l'appui sur la touche Échap (Escape) pour quitter le jeu.
  - Le test vérifie si un événement KEYDOWN avec la touche Échap a bien été enregistré, ce qui devrait entraîner la fermeture du jeu.
- testRecommencerJeu :
  - Ce test simule l'appui sur la touche R pour redémarrer la partie.
  - Après l'appui sur R, la méthode RedessinerPlateau est utilisée pour réinitialiser le plateau de jeu.
  - Le test vérifie si un événement KEYDOWN avec la touche R est bien enregistré, confirmant que le jeu a été redémarré.

III. Fonctionnalités principales du jeu

Le jeu propose des fonctionnalités essentielles permettant une expérience fidèle au jeu original Pac-Man. Le joueur contrôle Pac-Man à l'aide des touches directionnelles du clavier pour collecter des Pac-Dots répartis dans un labyrinthe composé de murs infranchissables. Les fantômes ennemis au comportement varié (chasse, fuite, aléatoire) poursuivent Pac-Man. Des Power Pellets permettent temporairement à Pac-Man d'inverser les rôles et de manger les fantômes. Un système de score affiche en temps réel les performances du joueur.

IV. Déploiement et configuration du jeu

➔ **Méthode 1 :**

Lancer l'executable Game.exe

➔ **Méthode 2 :**

Pré-requis techniques

1. Langage de programmation : Python 3.6 ou supérieur.
2. Dépendances : pygame, une bibliothèque utilisée pour la création de jeux en 2D.

Instructions d'installation et d'exécution

1. **Installer Python** : Téléchargez et installez Python depuis le site officiel python.org.
2. **Installer la dépendance PyGame** : Ouvrez un terminal ou une invite de commandes, puis exécutez :

```
pip install pygame
```

3. **Télécharger le code source depuis GitHub** : Dans un terminal, clonez le dépôt GitHub en utilisant la commande suivante :

```
git clone https://github.com/Ysope/COURMONTAGNE_TERRIER_GRISLIN_Projet.git .
```

4. **Lancer le jeu** : Accédez au répertoire contenant le jeu dans un terminal, puis exécutez la commande :

```
python Game.py
```

V. Déploiement et configuration des test unitaires / d'intégrations

Pour pouvoir activer les tests unitaires et d'intégrations :

1. **Installer la dépendance Unittest** : Ouvrez un terminal ou une invite de commandes, puis exécutez :

```
pip install unittest
```

2. **Aller dans le répertoire où se trouve le jeu** avec le terminal ou l'invite de commande
3. **Lancer les tests unitaires et d'intégrations** : dans le terminal ou l'invite de commande, exécutez :

```
python -m unittest discover -s tests
```