

**Projet informatique 1ère année
PRO 3600**

PAC-MAN multijoueur (Projet 23)

Jessim BEN REJEB

Abinash THARMASEELAN

Ayman ORKHIS

Si-Belkacem KETIR

Othmane HIMMI

Enseignante responsable : Chourouk BELHEOUANE

24/05/2024



Table des matières

1	Introduction	3
2	Cahier des charges	4
	2.1 Présentation générale	4
	2.2 Modes activables depuis le menu de démarrage du jeu	5
	A Mode Pouvoir	5
	B Mode Bomberman	6
	C Mode Portail	7
	2.3 Architecture du projet	9
	2.4 Diagramme de classes du projet	10
3	Développement	12
	3.1 Conception détaillée	12
	3.2 Tests et résultats	22
	3.3 Problèmes rencontrés et résolutions	26
4	Manuel utilisateur	30
5	Bilan et conclusion	37
6	Annexes	38
	6.1 Plan de charges et suivi des activités	38
	6.2 Diagramme de Gantt	39
	6.3 Problèmes au niveau de la gestion de projet	40
	6.4 Code source	41

Chapitre 1

Introduction

Ce document présente notre projet de conception du jeu mythique Pac-Man, en offrant de nouvelles fonctionnalités telles que la possibilité de jouer à 2 ainsi que d'autres modes de jeu.

Dans le cadre du module PRO3600, ce rapport final comporte le cahier des charges, le détail du développement du jeu durant le projet, le manuel utilisateur permettant de lancer le jeu à partir du zip fourni et contenant une explication détaillée du jeu, et se conclut par un bilan du projet. Ce rapport final présente en annexes le plan de charges et le suivi des activités du projet, le diagramme de Gantt du projet, ainsi que le code source du jeu.

Le langage de programmation utilisé est Java, l'interface graphique du jeu est réalisée à l'aide de la bibliothèque JavaFX et le menu est réalisée à l'aide de la bibliothèque Java Swing.

Chapitre 2

Cahier des charges

2.1 Présentation générale

Le projet consiste en la réalisation d'une version du jeu *Pac-Man* qui a la particularité de comporter 3 modes de jeu. De plus, le joueur a le choix de jouer seul (format solo) ou contre un ami (format 2 joueurs) sur le même ordinateur.

Le but du jeu est simple : on contrôle un personnage rond appelé Pac-Man à travers un labyrinthe. L'objectif principal est de manger le maximum de pac-gommes dispersées dans le labyrinthe (elles réapparaissent au cours du jeu) tout en évitant d'être touché par les quatre fantômes qui errent dans le labyrinthe et dont les vitesses augmentent au cours du temps. Le jeu se termine lorsque Pac-Man est touché par un fantôme. On octroie trois vies à chaque pac-man pour que la partie dure. L'objectif est donc d'obtenir le score le plus élevé possible en naviguant habilement à travers le labyrinthe. L'image en *figure 1* est un prototype qui illustre la situation décrite ci-dessus (le labyrinthe du jeu a été récupéré sur Shutterstock.com et imaginé par XrCyc). En format 2 joueurs, si un des pac-man a consommé toute ses vies, il a l'opportunité de continuer la partie mais cette fois-ci en tant que fantôme pour chasser le Pac-Man rival restant.

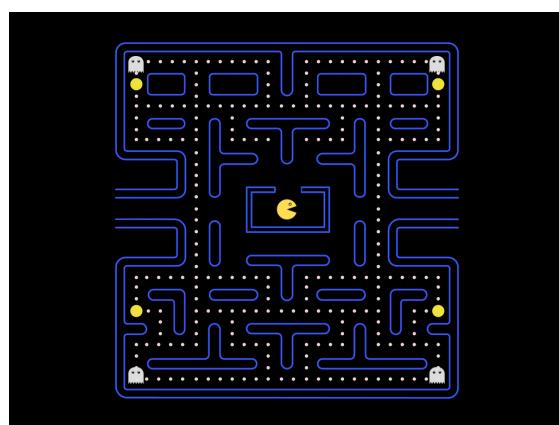


Figure 1 - Image du début d'une partie classique en format solo

Afin de proposer un jeu aussi dynamique que faire se peut, on laisse la possibilité au joueur d'activer plusieurs modes dans le menu de démarrage du jeu, et qui apportent tous des mécaniques de jeu différentes. Après avoir choisi entre le format solo et le format 2 joueurs, le joueur est redirigé vers une interface dans laquelle il doit cocher les options pour activer les modes (on peut sans problème cumuler plusieurs modes), ce après quoi la partie débute.

Le joueur peut aussi accéder au tableau des meilleurs scores lors du choix du nombre de joueurs. À la fin d'une partie, une fenêtre s'ouvre et demande le pseudonyme du joueur pour sauvegarder son score si ce dernier a battu l'un des records enregistrés dans la base de données.

La partie suivante aborde tous les modes de jeu pouvant être activés cumulativement.

2.2 Modes activables depuis l'interface de démarrage du jeu

A - Mode Pouvoir

Le premier mode que le joueur peut activer est le mode pouvoir. Ce mode fait apparaître des objets aléatoires (appelés “pouvoirs”) à des endroits différents de la map, qui ont la faculté d'avantager ou de mettre à mal le joueur ou son adversaire. De plus, il ne peut pas y avoir plusieurs bonus simultanés sur la map.

La *figure 2* montre l'item que doit ramasser le joueur pour acquérir un pouvoir :



Figure 2 - Item présent sur la map qui confère un pouvoir aléatoire

Les différents pouvoirs qui seront présents dans ce mode de jeu sont :

- *Changement de vitesse* : si un joueur collecte cet objet, il réduit ou accélère la vitesse de son adversaire ou la sienne pendant un certain temps ;
- *Bonus invulnérabilité* : si le joueur collecte cet objet, le joueur est invulnérable aux fantômes et aux bombes si le mode bomberman est activé. Ce bonus est activé pendant un certain temps ;
- *Malédiction* : ce malus attire les fantômes sur lui ou sur l'adversaire pendant un certain temps .

B - Mode Bomberman

Le deuxième mode activable est le mode “bomberman” (uniquement disponible en format 2 joueurs). Ce mode ajoute des bombes qui explosent en forme de “+” (voir *figure 4*) après un délai et qui font perdre une vie en cas de contact avec un des 2 Pac-Man (les fantômes sont immunisés). Les 2 joueurs commencent la partie avec 3 bombes (indicateur visuel du nombre de bombes affiché sur l’écran) et peuvent s’en servir à l'aide d'une commande pour tenter de tuer l'autre joueur. Des icônes de bombe apparaissent aléatoirement sur la carte et les joueurs peuvent s'en emparer en marchant dessus, ce qui augmente leur compteur de bombe de 1 (jusqu'à 5 maximum). La *figure 3* illustre les symboles des bombes à ramasser et ceux des bombes lorsqu'elles sont posées sur la map et prêtes à exploser :

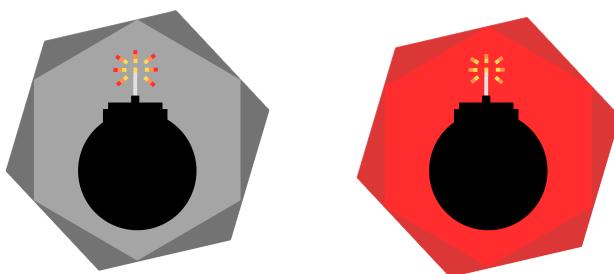


Figure 3 - Symbole d'une bombe à ramasser (en gris) et d'une bombe posée prête à exploser (en rouge)

Les déflagrations des bombes forment un “+” comme le montre la *figure 4*, ne traversent pas les murs/portails et peuvent activer prématurément des bombes en attente d’exploser. Par ailleurs, si un joueur est touché par sa propre bombe, son score diminue en plus de lui faire perdre une vie, tandis que s’il touche l’autre joueur avec une de ses bombes, son score augmente.

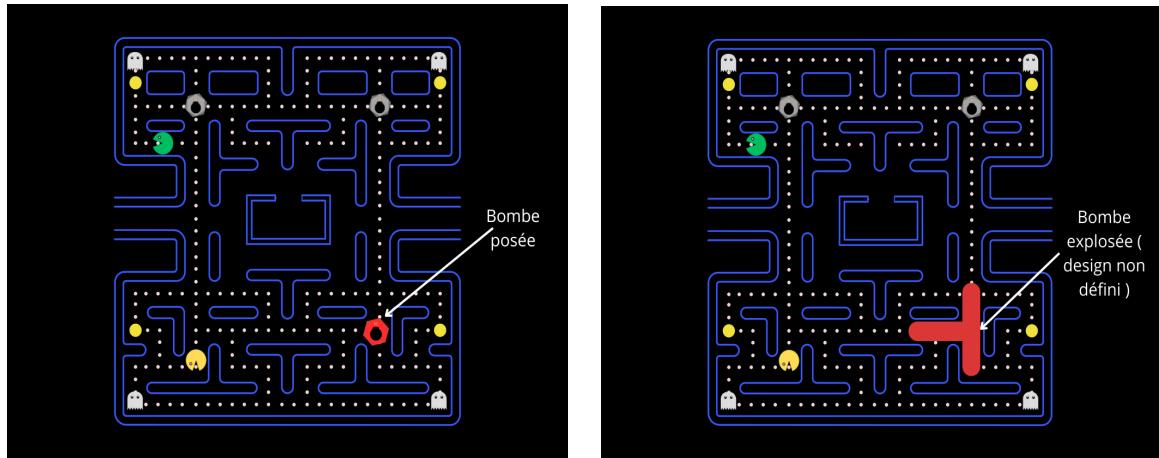


Figure 4 - Déflagration d'une bombe préalablement posée par un joueur

Un Pac-Man mort devenu fantôme peut ramasser les icônes de bombe et se servir de ces bombes. Une bombe occupe une case de la carte lorsqu’elle est posée en supprimant ce qui s’y trouvait précédemment (points de score, bonus, icône de bombe) et agit comme un mur pour les Pac-Man (les fantômes peuvent traverser les bombes).

C - Mode Portail

Enfin, le troisième et dernier mode de jeu proposé au joueur est le mode “portail”. Ce mode permet de générer un nombre fixe de portails sur la map, et qui permettent de téléporter le joueur qui l’emprunte à une position différente de la map. Cette mécanique de jeu offre une dynamique intéressante puisqu’elle peut mener à des situations où le joueur se retrouve d’un seul coup face à un fantôme qu’il doit absolument éviter, c’est l’aléatoire qui régit ce mode !

Nous choisirons un des trois portails de la **figure 5** pour leur design dans le jeu :

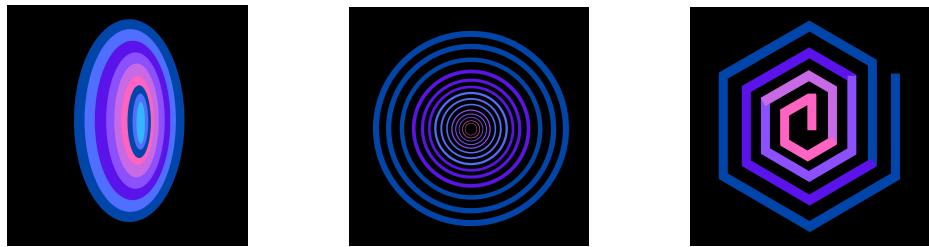


Figure 5 - Prototypes des aspects utilisés pour les portails

Lorsque le mode est activé, la partie débute avec un nombre fixe de portail sur la map (disons quatre portails). Si un joueur traverse un portail, il est téléporté de manière aléatoire vers l'un des trois autres portails. En revanche, un fantôme ne peut pas emprunter un portail tandis qu'en sortant d'un portail, le joueur ne peut pas l'emprunter à nouveau immédiatement.

La **figure 6** illustre le déplacement du joueur lorsqu'il traverse un portail :

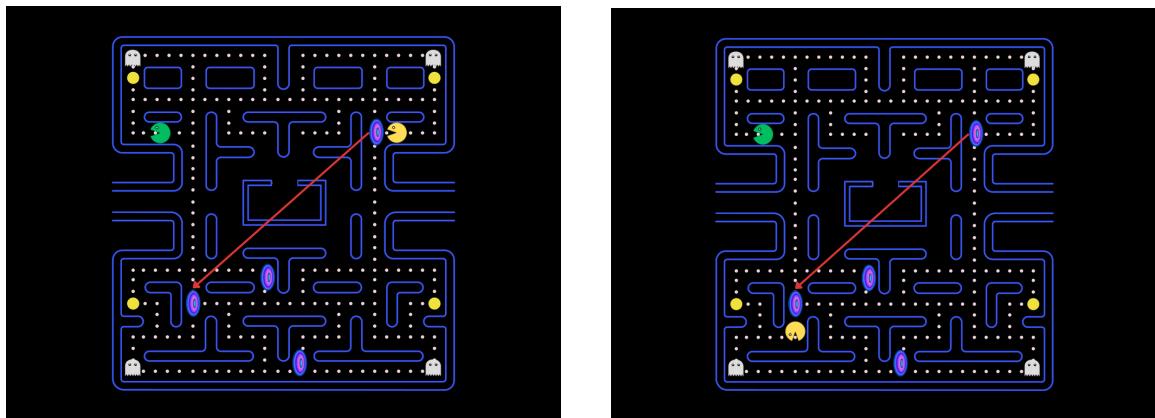


Figure 6 - Téléportation d'un Pac-Man après l'usage d'un portail

Pour ajouter encore plus de dynamisme à la partie, les portails se déplacent sur la map. Il faut donc avoir en tête que les portails apparaissent graduellement jusqu'à atteindre le nombre de 4 portails au début de la partie. Ce après quoi, les portails disparaissent graduellement pour réapparaître chacun à une autre position aléatoire selon la règle FIFO (le premier portail généré est celui qui va disparaître en premier, comme expliqué en **figure 7**). Chaque portail est donc présent pour une

certaine durée après son apparition avant de disparaître. La **figure 7** permet d'imager le concept décrit ci-dessus :

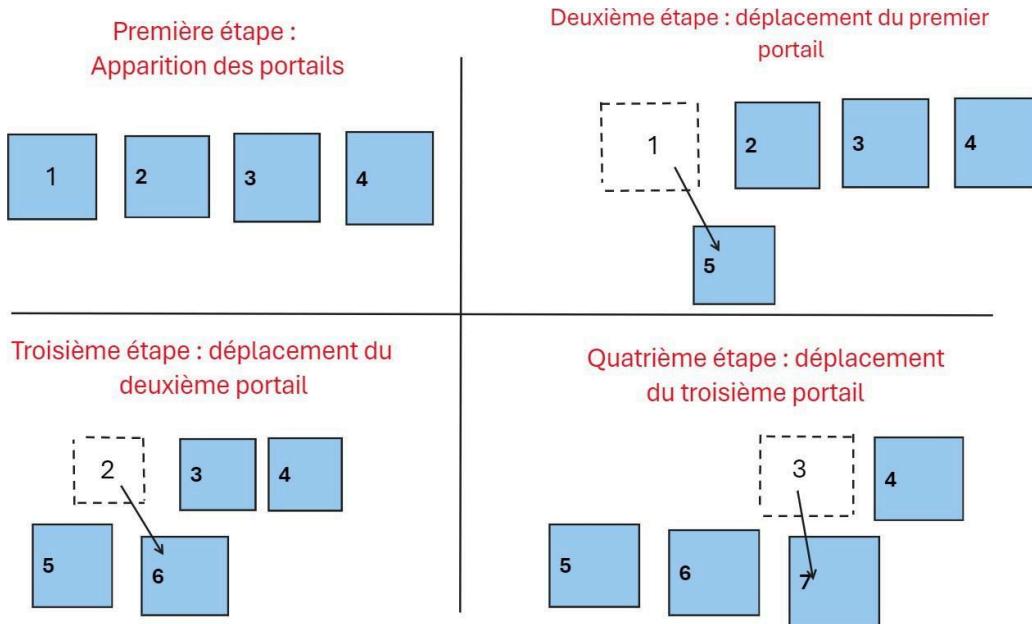


Figure 7 - Description de l'ordre de disparition des portails de la map

2.3 Architecture du projet

Nous allons développer le jeu Pac-Man multijoueur avec le langage de programmation JAVA sous Éclipse et IntelliJ. Nous le décomposons en 4 modules :

- le module *jeu* permettant l'exécution du jeu et la modification de certaines de ses constantes (vitesse de base des fantômes, de Pac-Man, temps avant qu'une bombe explose etc...),
- le module *modèle* assurant la gestion des données manipulées par le programme (position des joueurs, gestion des portails etc ...),
- le module *vue* assurant l'interface graphique pour l'utilisateur en fonction des données du programme (affichage du menu, de la map en temps réel etc...),
- le module *contrôleur* permettant à l'utilisateur d'interagir avec le programme et de modifier ses données en entrant des commandes.

Nous utilisons donc une architecture Modèle/Vue/Contrôleur (MVC) pour notre projet, les liens entre les modules sont représentés ci-dessous (cf. *figure 8*) :

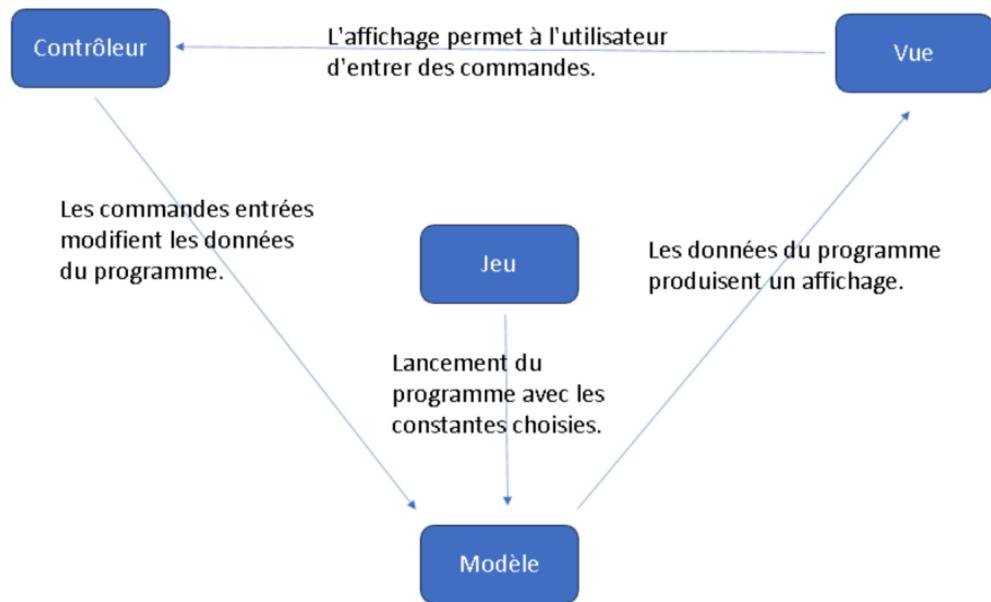
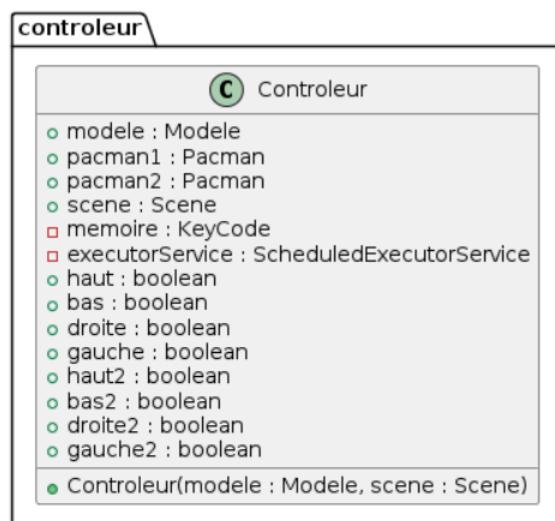


Figure 8 - Architecture MVC du projet

2.4 Diagramme de classes du projet

Pour le développement, nous avons suivi le diagramme de classes suivant (cf. *figure 9*) :



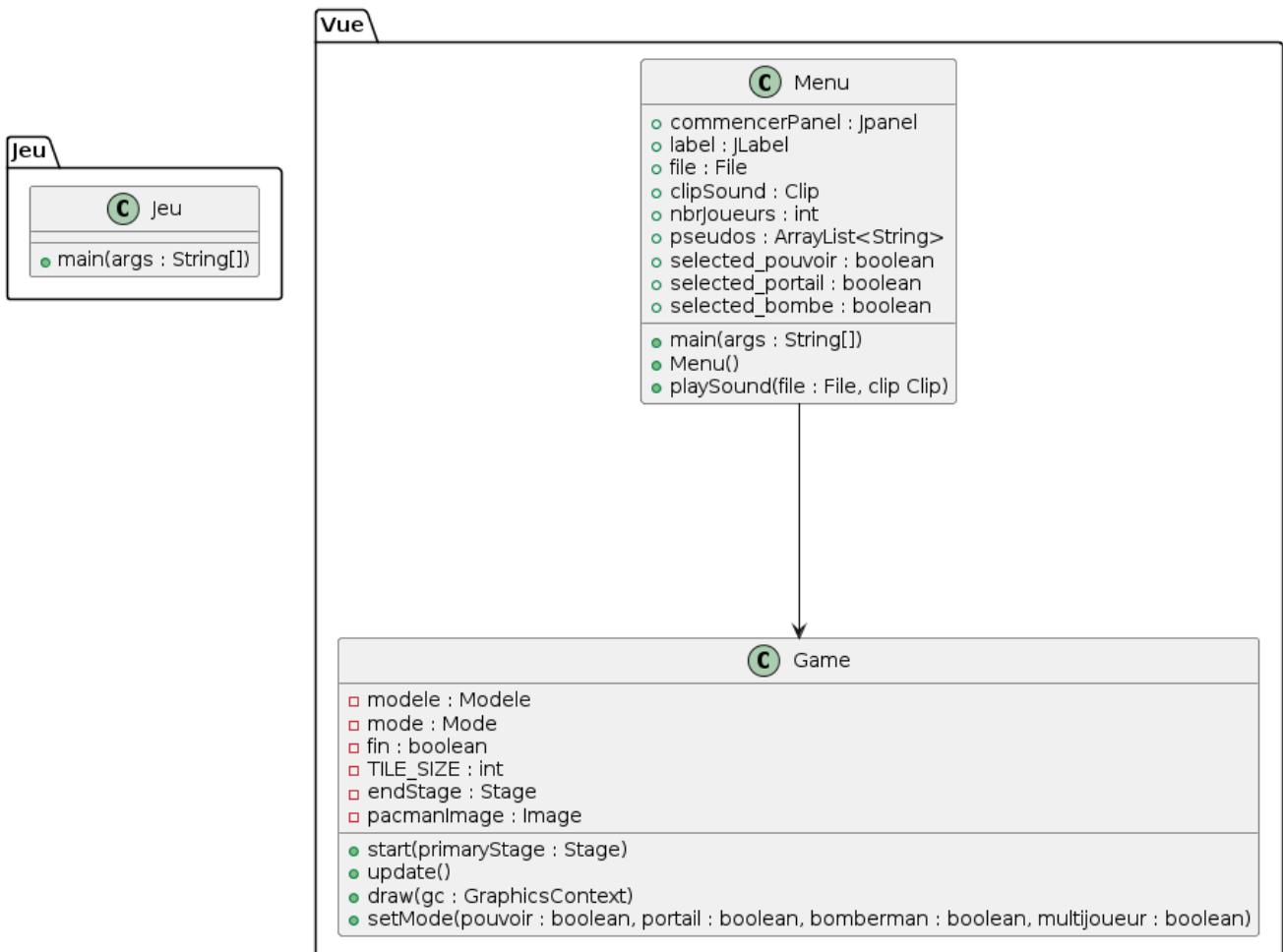
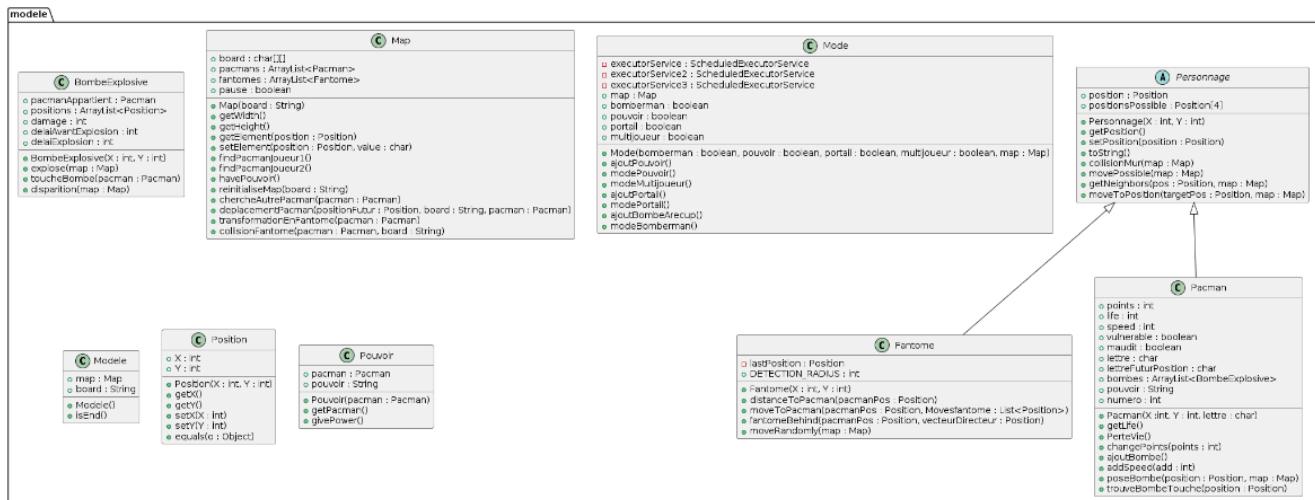


Figure 9 - Diagramme des classes du projet

Chapitre 3

Développement

3.1 Conception détaillée

3.1.1 Modèle

A - Cration et gestion de la map

Nous avons besoin de créer une map pour jouer à Pacman. Pour la modéliser et la créer, nous utilisons un double tableau de caractères. Afin de construire ce tableau, nous partons d'une chaîne de caractères avec des retours à la ligne comme suit :

Dans ce tableau : « P » et « Q » permet d'initialiser la position initiale des pacmans et les « F » indiquent les positions initiales des fantômes.

En **figure 10**, vous trouverez la signification des différents caractères que nous avons utilisés dans notre modélisation :

Élément Représenté	Caractères
Joueur 1	P
Joueur 2	Q
Position initial des fantômes	F
Mur	%
PacGomme	.
Pouvoir à récupérer	#
Bombe Posé	\$
Bombe à récupérer	@
Bombe qui a explosé	€
Portail	/

Figure 10 - Représentation des différents éléments par un caractère

Lorsqu'un Pac-Man se déplace, nous déplaçons sa lettre dans le tableau et nous appliquons les modifications nécessaires (augmentation des points, téléportation, ect....).

Pour rendre le code plus simple et moins chargé, les lettres « F » ne bougent pas. En effet, nous créons des objets fantômes qui ont une position et nous enregistrons les positions de ces fantômes. La position des fantômes sera utilisé pour dessiner les fantômes. Vous pouvez vous demander pourquoi nous ne déplaçons pas les lettres « F ». C'est parce qu'un fantôme ne ramasse rien et donc il aurait fallut utiliser un caractère pour dire que le fantôme et un item sont superposée (afin de dessiner à la fois le fantôme et l'item) et garder en mémoire l'item qui a été caché par le fantôme pour le remettre en place lorsqu'il continue son déplacement.

B - Gestion du déplacement du Pacman sur la map

Chaque pacman cherche ses mouvements possibles à l'aide de la méthode movePossible de la classe Personnage. Ensuite, pour déplacer le personnage après avoir récupéré le souhait de l'utilisateur (le fonctionnement du contrôleur est expliqué dans B.Contrôleur), nous actualisons la map en déplaçant la lettre du pacman et nous appliquons les conséquences liées à ce déplacement. Ensuite, nous terminons pas actualiser la position du Pacman.

C - Développement de l'IA

L'IA du fantôme dans le jeu Pacman utilise diverses méthodes pour suivre et attraper Pacman. Premièrement, le fantôme calcule la distance de Manhattan entre lui-même et Pacman pour déterminer sa proximité (méthode `distanceToPacman`). Lorsqu'il se déplace, il choisit le mouvement qui minimise cette distance (`moveToPacman`). Pour éviter les mouvements en boucle, l'IA utilise le champ `lastPosition` de la classe fantôme, empêchant le fantôme de revenir immédiatement à sa dernière position.

Une méthode clé de l'IA est la vérification de la position de Pacman par rapport au fantôme à l'aide du produit scalaire. Cette méthode, `fantomeBehind`, projette le vecteur directionnel du fantôme sur le vecteur allant du fantôme à Pacman. Si le produit scalaire de ces vecteurs est négatif, cela indique que Pacman est derrière le fantôme. Cela permet au fantôme d'inclure des mouvements inverses dans ses options pour empêcher Pacman de se faufiler derrière lui tout en respectant le rayon de détection expliqué dans le paragraphe suivant.

L'IA utilise également un rayon de détection (`DETECTION_RADIUS`) pour définir une distance de recherche pour le fantôme. En mode Solo, le fantôme commence à suivre le Pacman si seulement si ce dernier est dans sa zone de détection. En mode multijoueur, le fantôme cible le Pacman le plus proche ou le Pacman maudit, ajustant le rayon de détection en conséquence. Lorsqu'aucun Pacman n'est à proximité immédiate dans les deux modes, les fantômes effectuent des mouvements aléatoires parmi les options disponibles pour espérer mettre les pacmans dans leur zone de détection.

D - Modes de jeux (Portails, Bomberman et Pouvoir)

Pour créer les modes de jeux, nous avons commencé à créer des classes « Portails », « BombeRamassable », « BombeExplosive » et « Pouvoir ». Certaines de ces classes étaient peu développées et n'apportaient pas grand-chose. Nous avons donc décidé de supprimer les classes « BombeExplosive » et « Portails » car elles étaient déjà représentées sur la carte par des caractères et il n'était pas utile de créer des classes à part. Par exemple, pour le mode portail, lorsque dans la map, le pacman rencontre un '/' on le téléporte aléatoirement et nous n'avons pas besoin de créer un élément portail et de le garder en mémoire.

En revanche, nous avons eu besoin de créer des classes « Pouvoir » et « BombeExplosive » car ce sont des objets plus complexes qui doivent être accessibles dans le temps par les Pac-Man. Par exemple, les bombes explosé ne doivent être affichées sur la map que quelques secondes, et nous devons enregistrer à qui

appartient la bombe pour réaliser les changements nécessaires lorsqu'un Pacman est touché.

Nous avons créé une classe mode afin de gérer la gestion de l'apparition des bombes, des portails et des pouvoirs sur la map. Nous avons également eu besoin pour ce mode d'utiliser les bibliothèques «`java.util.concurrent.ScheduledExecutorService`», «`java.util.concurrent.Executors`», et «`java.util.concurrent.ScheduledFuture`», pour répéter une tâche qui s'effectue à une certaine fréquence.

E - Multijoueur

Initialement, nous avons une carte où les deux joueurs sont présents. Nous utilisons un tableau de taille 2 dans la carte pour stocker les deux Pacmans.

Si le mode multijoueur n'est pas activé, nous retirons la lettre « Q » de la carte et nous éliminons le deuxième Pacman du tableau. Ainsi, dans les autres classes, il suffit de vérifier la taille du tableau pour déterminer si nous sommes en mode solo ou multijoueur, et adapter nos différents codes en conséquence. Par exemple, dans le contrôleur, nous vérifions si nous sommes en mode multijoueur afin d'activer les commandes du joueur 2 : sans cette vérification, des erreurs surviendraient et le jeu ne serait pas jouable.

3.1.2 Contrôleur

Pour déplacer le pacman sur la map avec les touches, nous utilisons des bibliothèques de JavaFX : «`javafx.scene.Scene`» et «`javafx.scene.input.KeyCode`».

Nous avons un attribut mémoire de type `KeyCode` qui récupère la dernière touche appuyée. Selon la touche appuyée, nous définissons des booléens haut, bas, droite et gauche pour choisir la direction du Pac-Man. Ces variables permettent de déterminer la direction que Pacman doit prendre et de maintenir cette direction tant que la valeur des booléens n'a pas changé (la valeur des booléens est modifiée lorsque le joueur appuie sur une autre direction).

Pour continuer le mouvement de Pacman à une certaine fréquence (et donc à une certaine vitesse), nous utilisons un attribut de type `ScheduledExecutorService` avec les bibliothèques «`java.util.concurrent.Executors`», «`java.util.concurrent.ScheduledFuture`» et «`java.util.concurrent.ScheduledExecutorService`» .

3.1.3 Vue

A - Menu

Comme dans n'importe quelle jeu, le joueur arrive en premier lieu sur l'interface d'un menu au moment du démarrage.

L'objectif de cette partie du projet est alors de créer un menu simple d'utilisation doté d'une musique de fond et qui utilise des interfaces distinctes pour chaque étape de la configuration du jeu. Le menu est donc structuré en plusieurs interfaces successives qui emploient des images libres de droit ou générées par une intelligence artificielle.

Pour la conception de ce menu, il était initialement convenu de le réaliser au moyen de la bibliothèque JavaFX mais n'ayant découvert qu'un nombre limité de ressources sur internet utilisant JavaFX pour la conception de menus, nous avons opté pour la bibliothèque JavaSwing. Le code mis en place est donc inspiré de vidéos visualisées en auto-formation et de codes open-source disponibles sur Git.

PS : La description du code qui va suivre n'est pas exhaustive. Elle illustre surtout les méthodes principales utiles à la classe Menu.

L'idée générale du code est de construire une fenêtre dans laquelle on va superposer plusieurs interfaces ayant chacune une fonction distincte. Pour passer d'une interface à l'autre, il suffira juste d'utiliser une commande qui permet de rendre visible l'interface que l'on cherche à montrer, tandis qu'il faudra cacher toutes les autres interfaces.

Nous utiliserons 5 interfaces pour le menu :

- Interface 1 : Démarrage
- Interface 2 : Règles
- Interface 3 : Nombre de Joueurs
- Interface 4 : Sélection des pseudos
- Interface 5 : Sélection des modes activables

Par exemple, pour passer de interface1Label à interface2Label, on utilise la méthode d'instance setVisible(true) pour rendre visible interface2Label et setVisible(false) pour cacher interface1Label lorsque l'on clique sur le bouton « Jouer ». La **figure 11** ci-dessous permet d'illustrer la transition entre ces deux interfaces.

```

@Override
public void mouseClicked(MouseEvent e) {
    try {
        playSound(file3 , clipCommencerSound);
    } catch (LineUnavailableException | IOException | UnsupportedAudioFileException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
    // On passe à l'interface suivante.
    repaint();
    interface1Label.setVisible(false);
    interfaceregleLabel.setVisible(false);
    interface2Label.setVisible(true);
    //Ajout des boutons exit et music sur l'interface suivante.
    interface2Label.add(exitLabel);
    interface2Label.add(musicLabel);
}

```

Figure 11- Transition entre interface1Label et interface2Label

A.1 - Interface de démarrage

Il s'agit de la première interface visible par le joueur à l'ouverture du jeu. Celle-ci dispose de quatre boutons : “Jouer”, “Règle de jeu”, “Exit” et enfin l'icône “Sourdine” en haut à gauche. Si le Joueur clique sur “Jouer”, il passe directement à l'interface suivante qui lui propose de choisir le nombre de Joueur. S'il clique sur “Règle de Jeu” une interface avec les règles s'ouvre. La *figure 12* ci-dessous décrit le résultat final de la première interface.



Figure 12 - Interface de démarrage du jeu.

Pour créer tous ces boutons, il faut d'une part fixer leurs dimensions et leur localisation dans l'interface, définir le chemin du fichier image pour l'apparence du boutons et enfin expliciter ses event-handlers pour spécifier le comportement du

bouton selon l'action de l'utilisateur. En effet, le menu est notamment codé pour ajouter un bruitage et mettre en surbrillance les boutons lorsque l'on place la souris sur ces derniers.

Par exemple, le bouton “Jouer” devient vert et fait un bruitage lorsque l'on place le curseur dessus, revient à sa couleur d'origine lorsque l'on enlève le curseur et ouvre l'interface de sélection du nombre de joueurs lorsque l'on clique dessus. Pour implémenter tous ces événements en particulier, on utilise les méthodes *mouseEntered* et *mouseExited* pour définir le comportement du boutons « Jouer » comme le montre les *figures 13 et 14* ci-dessous.

```
//Label du bouton Jouer + ses event handlers
jouerLabel = new JLabel("");
jouerLabel.setBounds(500, 300, 283, 80);
jouerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/jouer.png")));
interface1Label.add(jouerLabel);
```

Figure 13 - Définition du bouton « Jouer » avec sa localisation, ses dimensions et son ajout à l'interface de démarrage.

```
@Override
//Lorsque l'on place seulement le curseur sur le Label "Jouer", on change la couleur et on met un bruitage
public void mouseEntered(MouseEvent e) {
    jouerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/jouer_hover.png")));
    try {
        playSound(file2 ,clipHoverSound);
    } catch (LineUnavailableException | IOException | UnsupportedAudioFileException e1) {
        // TODO Auto-generated catch block
        e1.printStackTrace();
    }
}
@Override
//Lorsque l'on retire le curseur du Label "Jouer", on revient à la couleur de base
public void mouseExited(MouseEvent e) {
    jouerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/jouer.png")));
}
```

Figure 14 - Comportement du bouton « Jouer » lorsque l'on place ou on enlève le curseur dessus.

A.2 - Interface des règles du jeu

Lorsque le joueur clique sur “Règle de Jeu”, il est redirigé sur l'interface que l'on peut voir sur la *figure 15* ci-dessous. On peut alors voir une image qui décrit les trois modes activables en multijoueurs et un bouton retour qui redirige le joueur sur l'interface de démarrage lorsque l'on clique dessus. Une nouvelle fois, des bruitages et effets sont ajoutés lorsque l'on clique sur ce bouton retour.

Nous aurions voulu coder de nouvelles interfaces avec une description explicative du mode sur lequel on clique mais le manque de temps ne nous a pas permis d'aller aussi loin. En l'état actuel de cette interface, le nom idéal du bouton "Règle de jeu" dans l'interface de démarrage aurait plutôt dû être "À propos".



Figure 15 - Interface des règles du Jeu

A.3- Interface de sélection du nombre de joueurs

Cette interface permet à l'utilisateur d'indiquer s'il joue tout seul ou avec un ami. Pour cela, il faut déjà ajouter les boutons de base "Exit" et "Sound" préalablement créés dans l'interface1Label tandis qu'il faut ensuite mettre en place le titre "Nombre de Joueur" et ensuite ajouter les deux boutons "1Player" et "2Players" (cf. **figure 16**). La méthode de création de ces nouveaux boutons est analogue à la description qui en est faite dans l'interface de démarrage.

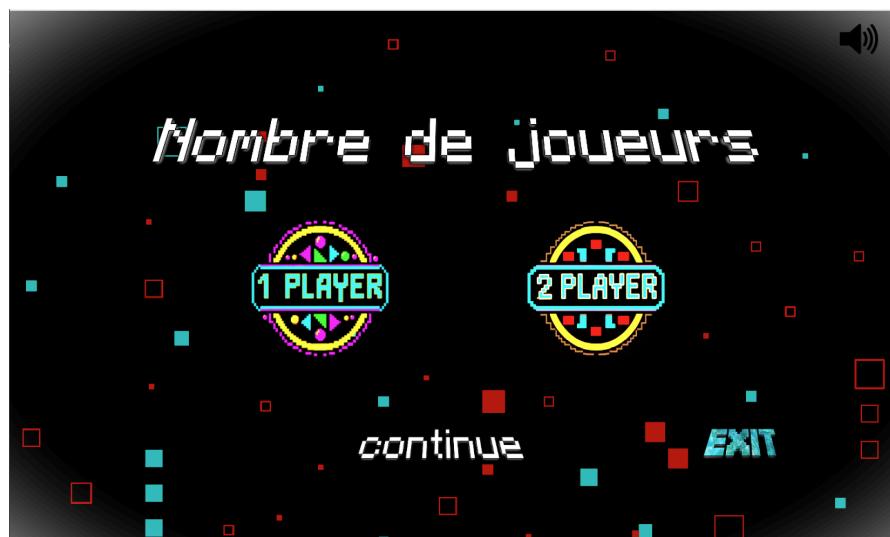


Figure 16 - Sélection du nombre de joueurs.

La nouveauté dans cette interface est l'ajout d'un gif sur l'icône sélectionnée pour améliorer l'aspect visuel de l'interface. Par ailleurs, il faut prendre des précautions au niveau du code pour que seul un des deux modes soit sélectionné.

Enfin, il y a une dernière subtilité. En effet, le bouton continue ne doit donner lieu à aucun résultat si aucun mode n'est sélectionné. Pour parer ce problème, on crée deux variables booléennes selected1 et selected2 qui valent respectivement true et false si on choisit le mode un joueur, ou false et true dans le cas contraire et on indique via “||” que cliquer sur le bouton “Continuer” ne donne lieu à aucun résultat si selected1 et selected2 ont pour valeur false et false comme le montre la *figure 17* :

```
if(selected1 || selected2) {  
    /*On passe à l'interface suivante si le nombre de joueurs a été choisi et  
     * que l'on clique sur continuer. On oublie pas de cacher les interfaces précédentes  
     * avant de passer à l'interface suivante  
     * et on ajoute les boutons "Exit" et "Music" à cette troisième interface  
    */  
    System.out.println("nbr de joueurs choisi :" + nbrJoueurs);  
    repaint();  
    interface1Label.setVisible(false);  
    interface2Label.setVisible(false);  
    interface3Label.setVisible(true);  
    interface3Label.add(exitLabel);  
    interface3Label.add(musicLabel);
```

Figure 17 - Passage à l'interface suivante uniquement si un des deux modes est sélectionné.

A.4 - Interface des pseudos

Surtout utile en mode 2 joueurs, cette interface permet d'attribuer un nom à chaque joueur. Pour cela, on crée deux instances de la classe JTextField et on règle les dimensions des icônes de texte.

Malgré de nombreuses recherches, il est impossible transformer l'arrière plan blanc du texte, en noir. Cela aurait amélioré l'aspect visuel de l'interface. On code ensuite un nouveau bouton “Continuer” qui permet d'accéder à la dernière interface (cf. *figure 18*).



Figure 18 - Sélection des pseudos en mode 2 Joueurs.

A.5 - Interface des modes activables

Il s'agit de la dernière interface du menu et qui permet de filtrer les bons modes de jeu. Par ailleurs, si aucun mode n'est sélectionné, alors il s'agit du jeu Pac-Man classique qui est lancé. Une nouvelle fois, des bruitages et des effets sont ajoutés aux boutons pour améliorer l'aspect graphique.

Enfin, les modes de jeu sélectionnés sont mis dans la liste "mode" qui contient les booléens des modes choisis et on ferme le menu via la commande `Menu.dispose()` lorsque l'on appuie sur le bouton "Commencer le Jeu" (cf. [figure 19](#)). À la fermeture du menu, on lance le jeu avec la commande `Game.launch(Game.class)` et le jeu va ensuite prendre en compte la liste des modes pour activer ceux choisis.

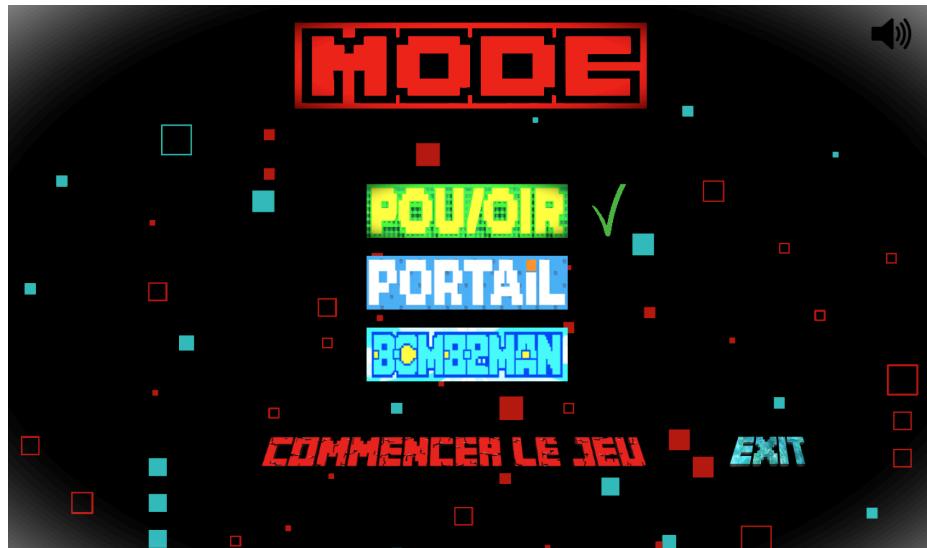


Figure 19 - Interface de sélection des différents modes de jeu

B - Interface graphique du jeu

Pour l'interface graphique du jeu, nous avons fait le choix de dessiner la grille de l'état du modèle sur un canvas et d'actualiser ce canvas avec un pas de temps choisi arbitrairement. Ainsi, nous dessinons directement sur le canvas certains éléments du jeu comme les pac-gommes ou les murs, mais nous chargeons des gifs créés par nos soins pour pacman et les fantômes. Les informations relatives à chaque joueur sont mises à jour en temps réel et sont affichées via des Label en même temps que le canvas.

À la fin de la partie, une nouvelle fenêtre s'ouvre et affiche le gagnant, ainsi que des Button permettant de retourner au menu, rejouer avec les mêmes paramètres, ou de quitter l'application.

3.2 Tests et Résultats

3.2.1 Classe Fantôme

Pour valider le comportement de l'IA des fantômes dans le jeu Pacman, nous avons mis en place des tests unitaire, intégration et validation en utilisant JUnit.

A - Test unitaire

Avant le test unitaire, la méthode setUp initialise la position et la vitesse d'un fantôme. La classe de test TestUnitaireFantome vérifie deux aspects cruciaux :

- 1) la précision du calcul de la distance entre le fantôme et Pacman,
- 2) la détermination de la position relative de Pacman par rapport au fantôme (Devant ou en arrière le fantôme).

Le test testDistanceToPacman s'assure que la méthode de calcul de la distance retourne la valeur correcte en plaçant Pacman à la position (X=4,Y=5) et en vérifiant que la distance calculée correspond à la distance attendue (dans notre cas 7 voir code source).

Le test testFantomeBehind vérifie que la méthode qui détermine si Pacman est derrière le fantôme fonctionne correctement, en utilisant des vecteurs directeurs pour simuler le mouvement et le signe du produit scalaire. Par conséquent, si le produit scalaire entre le vecteur directeur du fantôme et le vecteur entre le fantôme et le pacman est négatif, le Pacman est derrière le fantôme. S'il est positif, le Pacman est devant le fantôme.

Resultat Test unitaire (figure 20) :

```
Run TestUnitaireFantome (1) ×
Tests passed: 1 of 1 test - 12 ms
testDistanceToPacman() 12 ms /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java ...
Process finished with exit code 0
```

Figure 20 - Résultat Test unitaire

B - Test d'intégration

Avant un test d'intégration, la méthode setUp initialise une carte de jeu carrée avec un fantôme placé dans le côté bas droit du carré et un Pacman placé dans le côté haut gauche du carré.

testMoveRandomly, s'assure que le fantôme ne reste pas immobile lorsqu'il est censé se déplacer. Après avoir enregistré la position initiale du fantôme, il appelle la méthode moveRandomly du fantôme et vérifie que sa nouvelle position est différente de l'ancienne.

Résultat Test d'intégration (figure 21) :

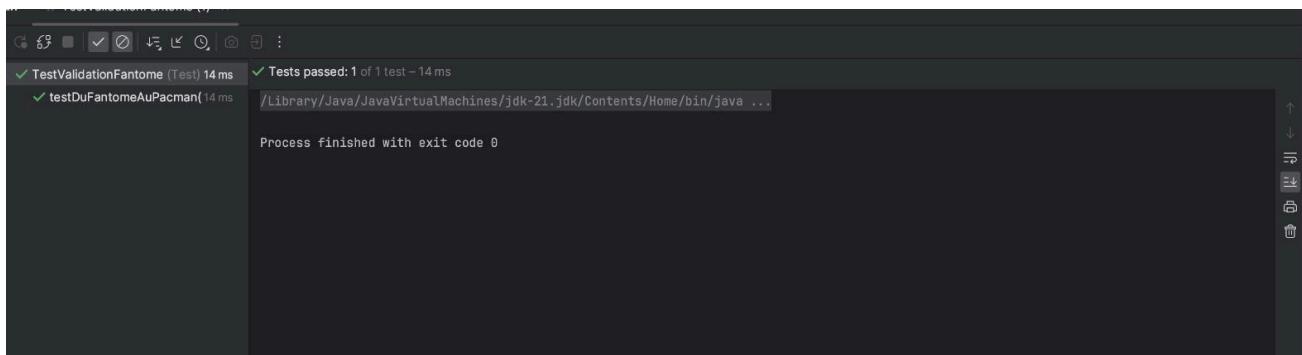
```
Run TestIntegrationFantome (1) ×
Tests passed: 1 of 1 test - 14 ms
testMoveRandomly() 14 ms /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java ...
Process finished with exit code 0
```

Figure 21 - Résultat Test d'intégration

C - Test de validation

Avant un test de validation, la méthode setUp initialise de la même façon que dans le test d'intégration mais en ajoutant un Pacman, testDuFantomeAuPacman, enregistre la position initiale de Pacman et mesure la distance entre le fantôme et Pacman. Après avoir simulé plusieurs mouvements du fantôme à l'aide de la méthode moveRandomly, il vérifie que la distance finale entre le fantôme et Pacman est inférieure ou égale à la distance initiale.

Résultat Test de Validation (figure 22) :



```
TestValidationFantome (Test) 14 ms
  ✓ Tests passed: 1 of 1 test – 14 ms
  ✓ testDuFantomeAuPacman(14 ms)

/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java ...
Process finished with exit code 0
```

Figure 22 - Résultat Test de validation

3.2.1 Classe Personnage

A - Test unitaire

Il s'agit ici de faire un test sur les setters et les getters de la classe Personnage. Pour cela, on crée une instance de Personnage à laquelle on affecte la position 2 en X et 1 en Y et on vérifie que getposition() envoie les bonnes coordonnées au moyen d'assertions (cf. *figure 23*).

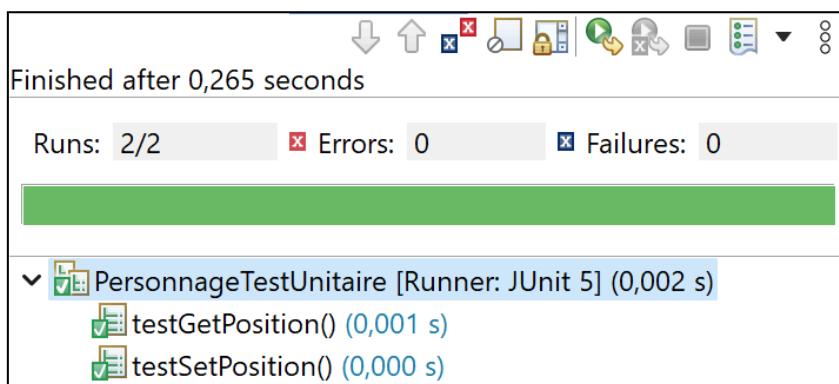


Figure 23 - Résultat du test unitaire

B - Test de validation

On vérifie que le lien entre la classe personnage et la classe map est correct. Pour cela, on se propose de tester la méthode ColisionMur() qui emploie elle-même des méthodes de la classe Map. Pour faire ce test, on fabrique un scénario dans lequel il y a un PacMan en haut à droite d'une map carré comme le montre la figure

ci-dessous. Le test de la méthode ColisionMur() consiste alors à vérifier que seules les directions à gauche et en bas du PacMan sont accessibles (cf. *figures 24 et 25*).

```
@BeforeEach
public void setUp() {
    personnage = new Personnage(2, 1, 1); // Correspond à la position de 'P' dans la carte
    String board = "%%%\\n% P%\\n% \\n%%%";
    map = new Map(board);
    /* %%%\\n
     * % P%\\n
     * % %\\n
     * %%%%
     */
}
```

Figure 24 - Création du scénario de test dans lequel il y a un joueur dans une petite map carré

```
@Test
public void testColisionMur() {
    // La carte est déjà configurée correctement par le constructeur

    // Appeler la méthode à tester
    Position[] result = personnage.colisionMur(map);

    // Vérifier que les positions possibles sont correctes
    assertNull(result[0]); // Haut bloqué par un mur
    assertNotNull(result[1]); // Bas accessible
    assertNotNull(result[2]); // Gauche accessible
    assertNull(result[3]); // Droite bloqué par un mur
}
```

Figure 25 - Code du test de la méthode ColisionMur()

De nouveau, le résultat du test de validation est positif comme le montre la *figure 26* ci-dessous.

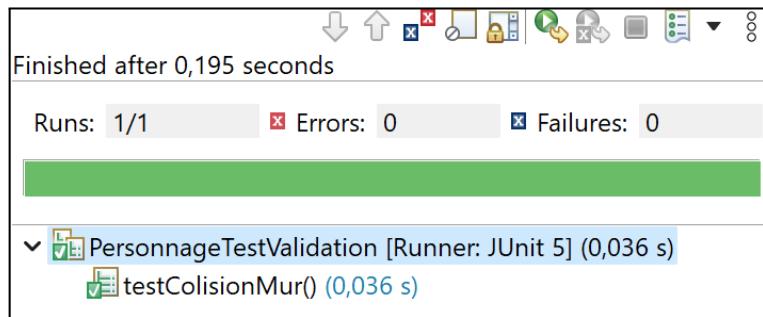


Figure 26 - Résultat du test de validation

C - Test d'intégration

Les méthodes de la classe personnage étant chacune indépendantes, il nous était impossible de faire un test d'intégration de la classe.

3.3 Problèmes rencontrés et résolutions

3.3.1 Modèle

A - Développement de l'IA

La première idée était d'utiliser l'IA du Pac-Man normal qui utilise plusieurs fantômes. Chacun de ces fantômes a un rôle. Blinky, le fantôme rouge, vous traque en vous suivant partout à travers la map. Pinky, le fantôme rose, cherchera toujours à se placer devant le pacman, etc...

Pour résoudre les problèmes où les fantômes restaient immobiles, des modifications ont été apportées à la classe `Game` pour assurer leur mouvement. En ajoutant la méthode `moveRandomly` à la classe Game. Les différentes versions de la méthode `moveRandomly` seront détaillées dans la suite de ce paragraphe.

Dans la première version, la méthode `moveRandomly` a été implémentée pour déplacer les fantômes de manière aléatoire sans suivre le Pac-Man. Cette méthode naïve avait des problèmes au début, notamment dans les impasses. Ce problème a été résolu en ajoutant un champ `lastMove`.

Dans la deuxième version de `moveRandomly`, une approche plus sophistiquée a été utilisée. La méthode `moveToPacman` a été développée pour permettre aux fantômes de suivre le Pac-Man en utilisant une stratégie basée sur la distance.

Cependant, le problème majeur de cette version était que le Pac-Man devenait invisible lorsqu'il se retrouvait derrière un fantôme.

Dans la dernière version, le problème de la version 2 a été résolu en ajoutant une méthode `fantomeBehind` qui permet de déterminer si le Pac-Man est devant ou derrière le fantôme. Cette version a également vu l'ajout d'un rayon de détection pour les fantômes, qui constitue la région de recherche des fantômes. Enfin, une dernière modification a été apportée avec l'ajout de la propriété "maudit" issue de la classe Pac-Man. Ce champ permet de favoriser un Pac-Man par rapport à un autre en ce qui concerne l'agressivité des fantômes (rayon de détection).

3.3.2. Vue

A - Dimensionnement des images pour le menu

Le plus grand problème lors de la création du menu était la complexité de la sélection des dimensions appropriées pour les images du menu. En effet, des images trop grandes ou trop petites et téléchargées directement depuis internet affectaient grandement la qualité visuelle et l'ergonomie du menu.

Pour remédier à cela, il a systématiquement fallu utiliser des logiciels qui redimensionnent les images afin de les adapter à la taille de la fenêtre, ou carrément apporter des modifications manuelles aux images via le logiciel "Paint".

Par ailleurs, certaines images sont issues d'une intelligence artificielle comme dit précédemment. Or, il est très difficile pour ces intelligences artificielles de générer une image avec de texte sans qu'il n'y ait d'incohérence. C'est pourquoi il a fallu trouver les meilleurs prompts, bien que certaines incohérences textuelles persistent toujours.

B - Relancement du menu à la fin de la partie

Le problème majeur non surmonté a été le bouton "Retour au menu". En effet, il permet de retourner au menu en fermant l'application JavaFX qui affichait le jeu, mais on ne peut pas relancer le jeu depuis ce menu : on ne peut utiliser la méthode `launch()` d'une application JavaFX qu'une seule fois durant l'exécution du programme.

Nous avons essayé de créer une nouvelle instance de la classe Game pour lancer la nouvelle instance, mais nous avons fait face au même problème.

C - Transition fluide pour Pac-Man et les fantômes

Bien que nous souhaitions jusqu'au bout implémenter des transitions fluides pour les déplacements de Pac-Man et des fantômes, nous n'avons pas été en mesure de le faire. Les réactualisations du canvas donnent l'impression que Pac-Man et les

fantômes se téléportent d'une case à l'autre.

Nous avons essayé de déplacer Pac-Man et les fantômes "plus lentement" sur le canvas en leur donnant des positions entre 2 cases durant la réactualisation du modèle, mais cela n'a pas abouti.

Nous avons ensuite tenté de superposer au canvas des sprites qui se déplacent fluidement de la position initiale à la position après réactualisation du canvas à l'aide de la classe PathTransition, mais nous avons constaté un décalage entre les sprites du canvas et ceux superposés. Nous avons finalement décidé d'abandonner l'idée d'implémenter des transitions fluides pour nous concentrer sur les autres fonctionnalités.

3.3.3. Contrôleur

Dans un premier temps, le contrôleur a été créé sans utiliser les booleans «Haut», «Bas», «Gauche» et «Droite». Le problème avec ce contrôleur est qu'il garde les anciennes directions données : par exemple si nous demandions à aller à droite puis à gauche, le pacman alternait entre ces deux directions. C'était un problème qui a été facilement corrigé .

Ensuite, dans la deuxième version du contrôleur qui a été faite, il n'y avait pas d'attribut pacmanTask qui enregistrait la tâche qui se répète. Avec ce contrôleur, lorsqu'on appuyait assez rapidement dans la même direction, le déplacement s'effectuait beaucoup plus rapidement (la vitesse doublait) : cela était dû au fait qu'un nouveau mouvement vers la même direction était lancé. Pour pallier ce problème, nous avons enregistré dans une instance « pacmanTask » de type ScheduledFuture<?>, la tâche qui répète le mouvement pour être sur qu'il n'y a qu'une seule tâche active et empêché d'annuler la tâche en cours.

Enfin, nous avons voulu mettre les commandes de touches en paramètres afin de créer deux objets contrôleurs (un pour chaque joueur) afin d'alléger le code. Mais ce contrôleur ne fonctionnait pas et empêchait de jouer. C'est pourquoi que les attributs que nous avons utilisés pour créer la commande du joueur 1 ont été tout simplement dupliqué, ainsi que des parties du code pour que tout fonctionne à partir d'un seul objet contrôleur.

3.3.4 Problème de l'exécutable du jeux

Nous avons réussi à créer un exécutable pour le premier prototype. Pour ce faire, nous avons utilisé un fichier batch qui spécifie l'emplacement du jeu et les bibliothèques utilisées par le jeu. Ce fichier batch a ensuite été transformé en exécutable à l'aide d'un logiciel capable de réaliser cette opération et de changer l'image de l'icône. Ainsi, nous avons obtenu un fichier exécutable comme illustré

dans la **figure 27** ci-dessous. Cependant, le changement de l'emplacement du jeu a entraîné un dysfonctionnement. Ce problème aurait pu être résolu, mais par manque de temps, nous avons renoncé à cette étape et nous nous sommes concentrés sur d'autres aspects plus fondamentaux.

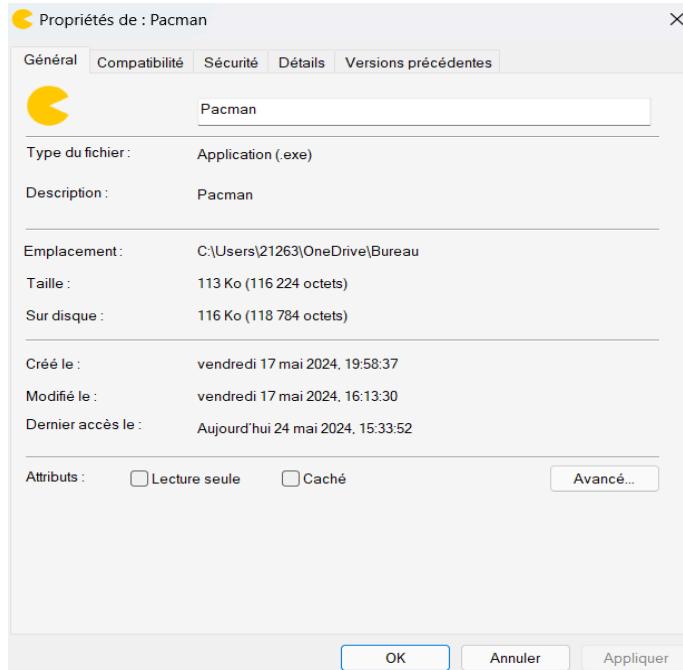


Figure 27 - Exécutable du jeu

Chapitre 4

Manuel Utilisateur

4.1 Guide d'installation

Afin de pouvoir jouer au jeu, il faut d'abord extraire le fichier zip, et l'importer dans Eclipse. Pour cela, aller sur File > Import project. Ensuite General > Projects from Folder or Archive, et cliquer sur Next. Cliquer sur Directory et chercher le dossier extrait, cliquer sur Ouvrir (cf. **figure 28**).

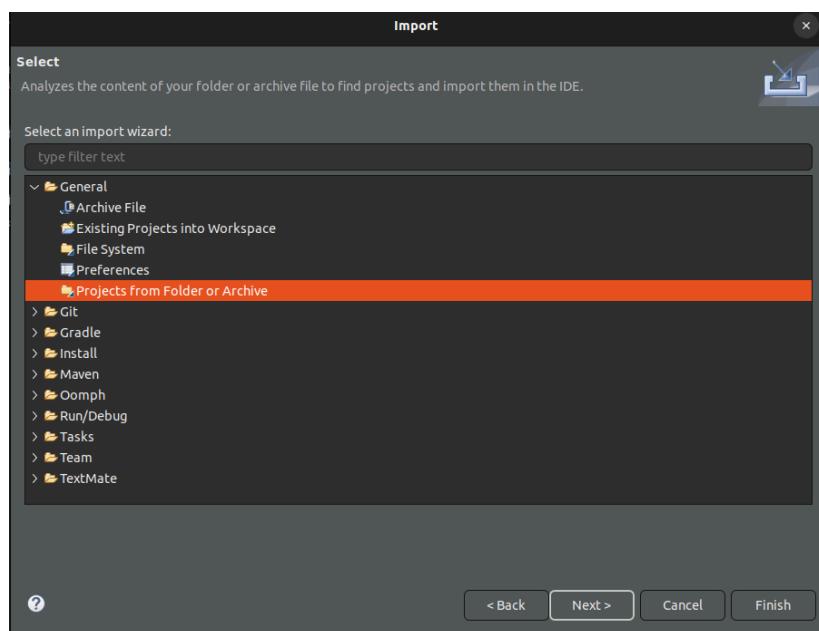


Figure 28 - Importation du projet

Le projet est dès à présent importé mais il n'est pas encore possible de compiler, il faut mettre en place la bibliothèque JavaFX. Pour cela, aller dans Help > Eclipse Marketplace, rechercher et télécharger “e(fx)clipse 3.8.0”.

Ensuite il faut aller sur ce lien <https://gluonhq.com/products/javafx/> et télécharger la version qui correspond à votre OS.

Aller dans l'onglet Window > preferences > Java > Build Path > User Libraries. Cliquer sur New et nommer le (javafx par exemple). Cliquer dessus et Add External JARs, sur le dossier lib du fichier téléchargé plus tôt, sélectionner tous les documents.jar et Ouvrir.

Maintenant que la bibliothèque est importée, il faut la configurer sur le projet. Faire un clique droit sur le projet, Build Path > Configure Build Path et vous devriez avoir l'interface ci-dessous (cf. *figure 29*) :

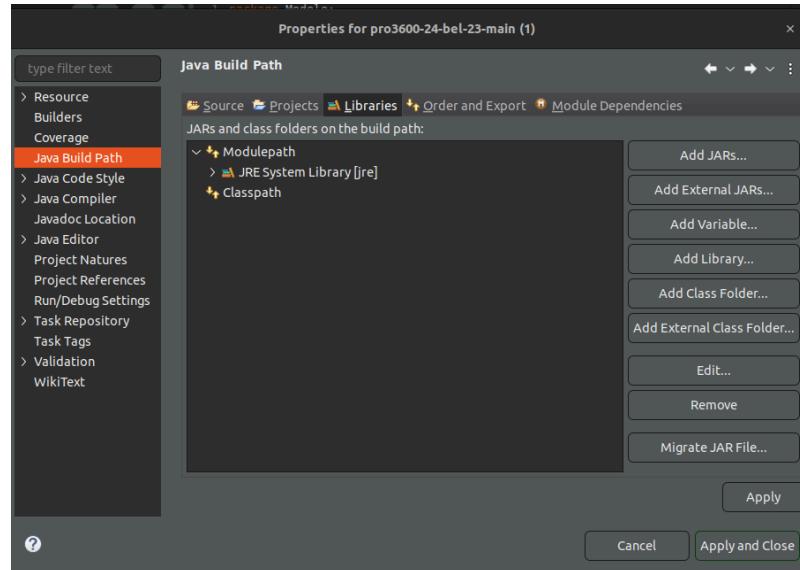


Figure 29 - Configuration du projet

Cliquer ensuite sur Classpath puis Add Library > User Library et sélectionner la bibliothèque javafx qui apparaît. La bibliothèque devrait maintenant apparaître dans Classpath.

Il devrait maintenant être possible de compiler le code (Run) et ainsi jouer à PAC-MAN multijoueur. Il faut lancer la classe “Jeu” pour obtenir l'écran montré sur la *figure 30* ci-dessous.

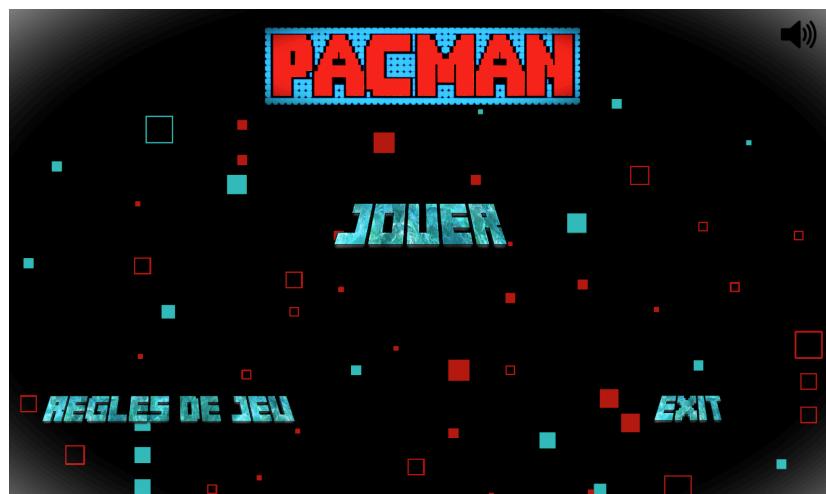


Figure 30 - Menu principal du jeu

4.2 Mode d'emploi

Sur la première interface du menu il y a 4 boutons : “JOUER”, “REGLES DE JEU”, “EXIT”, et une icône “son” (en haut à droite) pour enlever le son du jeu.

Le bouton “EXIT” permet tout simplement de quitter le jeu. Le bouton “REGLES DE JEU” permet d’expliquer les modes de jeu disponibles qui apparaissent accompagné d’une image chacun : “PORTAIL”, “POUVOIR”, “BOMBERMAN!” (cf. *figure 31*). Le bouton retour permet de revenir au menu principal.



Figure 31 - Règles du jeu

En cliquant sur “JOUER”, on peut alors choisir de jouer seul “1 PLAYER”, ou avec un ami “2 PLAYER” (cf. *figure 32*).

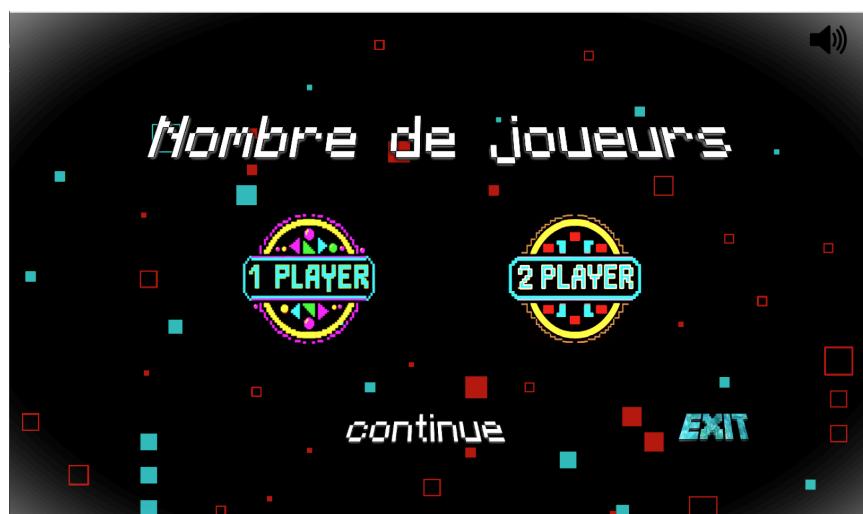


Figure 32 - Interface menu : nombre de joueurs

En choisissant le mode 2 joueurs, on peut ensuite entrer les pseudonymes des deux joueurs (cf. *figure 33*).



Figure 33 - pseudonymes 2 joueurs

Et en appuyant sur “continue”, on a le choix de sélectionner, ou non, chaque mode de jeu selon la préférence des joueurs. Les icônes deviennent vertes avec un “check” à côté pour indiquer que les modes ont bien été sélectionné (cf. *figure 34*).

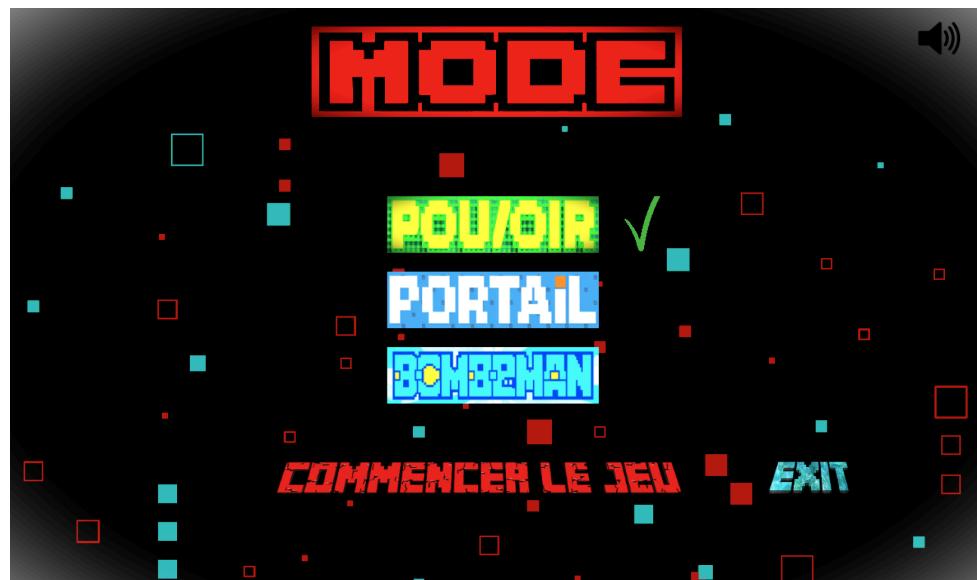


Figure 34 - modes de jeu

Le jeu commence directement en appuyant sur “COMMENCER LE JEU” et on obtient l’écran suivant (cf. *figure 35*) :

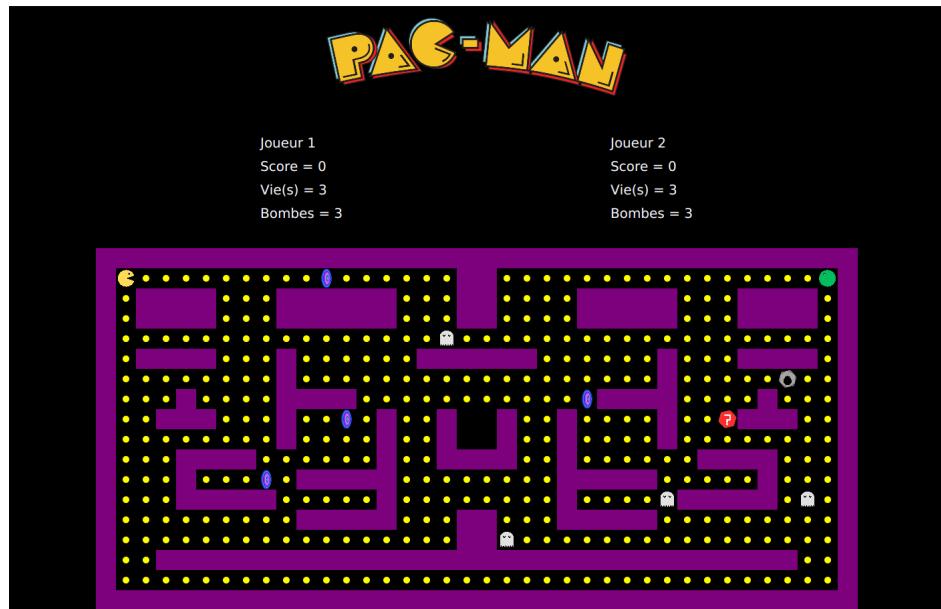


Figure 35 - Pacman 2 joueurs avec tous les modes

En mode 1 joueur, le PacMan se déplace en utilisant les flèches directionnelles (haut, bas, gauche, droite).

En mode 2 joueurs, le PacMan Jaune se déplace avec les flèches directionnelles, et pose une bombe avec ‘M’ (si le mode Bomberman est activé). Et le PacMan vert se déplace avec les touches ‘Z’, ‘Q’, ‘S’, ‘D’, et pose une bombe avec R.

En ramassant les “pac-gommes”, le score augmente, en touchant un fantôme ou l’explosion d’une bombe, le nombre de vies diminue, et en ramassant des bombes, son nombre augmente.

À la collision avec un portail, le pac-man est téléporté vers un autre portail aléatoirement dans la map.

Un pouvoir donne l’un des quatres attributs au pac-man qui l’a ramassé de manière aléatoire :

- Changement de vitesse : réduit ou accélère la vitesse de son adversaire ou la sienne pendant un certain temps.
- bonus invulnérabilité : invulnérable aux fantômes et aux bombes pendant un certain temps.
- malédiction : malus qui attire les fantômes sur soi ou sur l’adversaire pendant

- un certain temps.
- bonus fantôme : (mode 2 joueurs seulement) Transforme pendant un certain temps le pac-man en fantôme qui peut traverser les murs et tuer les pac-man adverse.

Une ligne de texte en haut vous permet de savoir quel pouvoir a été choisi (cf. *figure 36*).

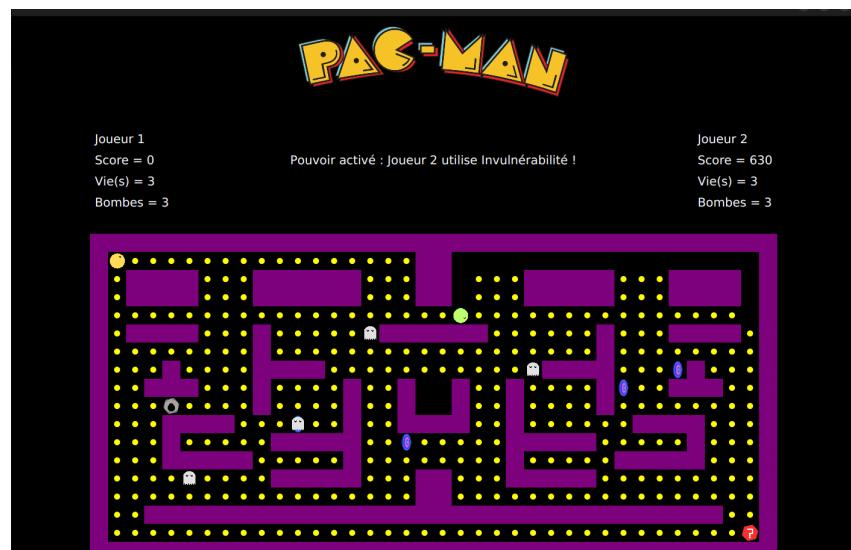


Figure 36 - exemple pouvoir invulnérabilité

Lorsqu'un joueur n'a plus de vies, il devient comme un fantôme : il ne peut plus augmenter son score en collectant des pac-gommes, mais peut tuer le pac-man adverse ce qui lui fait gagner des points. La partie se termine lorsque les 2 joueurs n'ont plus aucune vie (cf. *figure 37*).



Figure 37 - interface fin de partie

Chapitre 5

Bilan et Conclusion

Ce rapport décrit les principales étapes du développement de la version multijoueur d'un jeu Pacman utilisant une architecture MVC.

L'architecture MVC a permis de structurer l'application de manière claire, facilitant la gestion et l'évolution du code. Le modèle gère les données et la logique du jeu, notamment les positions des Pacmans, des fantômes, et des différents objets sur la carte. Il est basé sur un tableau à double entrée de caractères représentant la carte. La vue est divisée en deux parties : le menu codé à l'aide de Java Swing et l'interface du jeu codée avec JavaFX. Enfin, le contrôleur permet de gérer les entrées utilisateurs et d'appliquer les conséquences sur le jeu.

Il y a un léger écart par rapport au cahier des charges et aux objectifs fixés : la base de données et l'affichage des meilleurs scores n'ont pas été implantés. En effet, nous nous sommes pleinement concentré sur le jeu et ses fonctionnalités afin d'avoir une version du jeu sans bugs et problèmes. De plus, l'affichage des scores nous a paru être le point le moins important dans ce projet, car cette nouvelle version de Pacman a été réalisée pour être jouée principalement en multijoueur.

Bien que cette version du jeu soit pleinement fonctionnelle, de nombreuses améliorations sont possibles. Par exemple, l'introduction de nouvelles fonctionnalités comme des objets supplémentaires et des niveaux variés. Ce jeu peut également être amélioré par l'ajout de l'affichage des meilleurs scores, le seul point qui n'a pas été respecté du cahier des charges.

Chapitre 6

Annexes

6.1 Plan de charges et suivi des activités

Nous donnons ci-dessous en *figure 38 et 39* le plan de charges prévisionnel et le suivi des activités du projet:

PLAN DE CHARGES PRÉVISIONNEL							
Description de l'activité	Charge en H / Participant						
	Charge en %	Charge en H	Jessim	Abinash	Si-Belkacem	Othmane	Ayman
Total	100%	250	50	50	50	50	50
Gestion de projets							
Réunion de lancement (avec tutrice)	3%	5	1	1	1	1	1
Planning prévisionnel et Suivi d'activités	3%	5	1	1	1	1	1
Réunions de suivi (avec tutrice)	15%	30	6	6	6	6	6
Rédaction	8%	15	3	3	3	3	3
Spécification							
Définition des fonctionnalités	5%	10	2	2	2	2	2
Conception préliminaire							
Définition des modules	5%	10	2	2	2	2	2
Conception de l'architecture	1%	1	1	0	0	0	0
Conception détaillée							
Définition des classes	5%	10	2	2	2	2	2
Définition des méthodes	5%	10	2	2	2	2	2
Définition des tests unitaires	5%	10	2	2	2	2	2
Auto-formation	7%	14	4	2	4	2	2
Conception de l'architecture finale	3%	5	1	1	1	1	1
Définition de la base de données	5%	9	1	2	2	2	2
Codage							
Codage des classes et méthodes	30%	60	12	14	10	12	12
Codage des tests unitaires	6%	12	3	3	2	2	2
Intégration							
Intégration des modules	5%	9	0	0	3	3	3
Tests d'intégration	5%	10	2	2	2	2	2
Soutenance							
Préparation de la soutenance	10%	20	4	4	4	4	4
Soutenance	3%	5	1	1	1	1	1

Figure 38 - Plan de charges

SUIVI D'ACTIVITÉS (Charge Consommée)							
Description de l'activité	Charge en H / Participant						
	Charge en %	Charge en H	Jessim	Abinash	Si-Belkacem	Othmane	Ayman
Total	126%	315	65	72	65	41	72
Gestion de projets							
Réunion de lancement (avec tutrice)	2%	5	1	1	1	1	1
Planning prévisionnel et Suivi d'activités	2%	5	1	1	1	1	1
Réunions de suivi (avec tutrice)	8%	20	4	4	4	4	4
Rédaction	8%	19	4	4	4	3	4
Spécification							
Définition des fonctionnalités	6%	15	3	3	3	3	3
Conception préliminaire							
Définition des modules	4%	10	2	2	2	2	2
Conception de l'architecture	0%	1	1	0	0	0	0
Conception détaillée							
Définition des classes	2%	6	1	1	1	1	2
Définition des méthodes	4%	9	1	3	2	1	2
Définition des tests unitaires	1%	2	0	0	1	0	1
Auto-formation	30%	74	20	4	18	12	20
Conception de l'architecture finale	1%	2	2	0	0	0	0
Définition de la base de données	0%	0	0	0	0	0	0
Codage							
Codage des classes et méthodes	46%	114	20	30	23	13	28
Codage des tests unitaires	0%	1	0	0	0	0	1
Intégration							
Intégration des modules	10%	26	5	19	1	0	1
Tests d'intégration	2%	6	0	0	4	0	2
Soutenance							
Préparation de la soutenance	0%	0	0	0	0	0	0
Soutenance	0%	0	0	0	0	0	0

Figure 39 - Suivi des activités

6.2 Diagramme de Gantt

Nous avons divisé le travail de programmation en 3 équipes : 2 personnes ont réalisé le module “Vue” (auto-formation), 2 autres personnes le module “Modèle” et le dernier membre s'est occupé du module “Contrôleur”. Tous les membres ont pu travailler en parallèle sur une même classe en cas de difficulté. Une fois les modules précédents implémentés, l'équipe s'est concentrée sur l'intégration des modules et sur la réalisation du module “Jeu”. Une fois que tous les modules implémentés, nous nous sommes concentrer sur les tests unitaires et d'intégration, puis nous avons utilisé le reste du temps pour le perfectionnement et la correction des défauts du programme (cf. **figure 40** ci-dessous) :

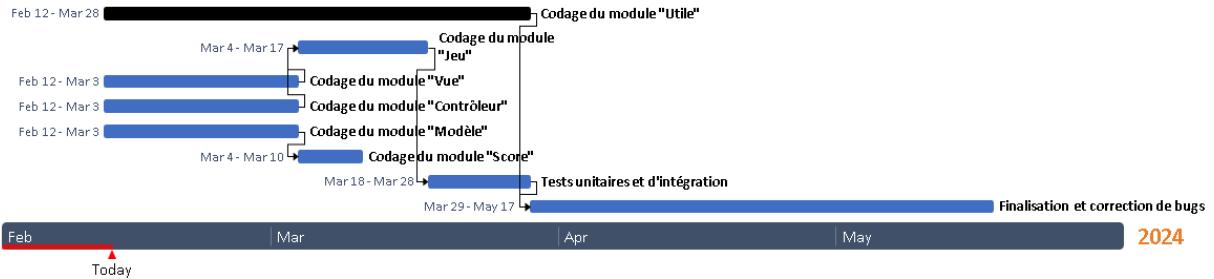


Figure 40 - Diagramme de Gantt

6.3 Problèmes au niveau de la gestion de projet

Le problème majeur qui a drastiquement ralenti l'avancée du projet est en relation avec le git. Certaines membres du groupe n'arrivaient pas à importer le projet dans leur IDE, tandis que d'autres n'arrivaient pas à push, pull, ou même à compiler. La solution qui a été trouvée est d'envoyer manuellement le projet en zip à ceux qui n'arrivaient pas à utiliser le Gitlab et de nous envoyer entre nous les classes java qui avaient été modifiées et d'utiliser google drive, en plus du Gitlab. Cela a pu être possible car nous travaillions tous sur des classes distinctes.

6.3 Code Source

6.3.1 Contrôleur

```
1 package Controleur;
2
3 import Modele.*;
4 import Modele.Pacman;
5
6 import javafx.scene.Scene;
7 import javafx.scene.input.KeyCode;
8
9 import java.util.concurrent.Executors;
10 import java.util.concurrent.ScheduledExecutorService;
11 import java.util.concurrent.ScheduledFuture;
12 import java.util.concurrent.TimeUnit;
13 public class Controleur {
14     public Modele modele; //le contrôleur applique des changements sur un modèle
15     public Pacman pacman1; // on récupère le joueur1
16     public Pacman pacman2; //on récupère le joueur2 s'il y en a un
17     public Scene scene; //on récupère la scène de l'interface graphique
18     KeyCode memoire; //on récupère la touche qui a été appuyé dans une variable
19     private ScheduledExecutorService executorService = Executors.
newSingleThreadScheduledExecutor(); // pour gérer une tâche qui s'exécute à une
certaine fréquence
20
21     // pour joueur 1 : on enregistre le mouvement en cours à l'aide de boolean
22     boolean haut = false;
23     boolean bas = false;
24     boolean droite = false;
25     boolean gauche = false;
26     private ScheduledFuture<?> pacmanTask; // il va nous servir pour éviter de créer
plusieurs tâche qui se répète et donc d'accumuler des directions : on a
seulement besoin d'une tâche qui fait déplacer le pacman
27
28     // pour joueur 2 : on enregistre le mouvement en cours à l'aide de boolean
29     boolean haut2 = false;
30     boolean bas2 = false;
31     boolean droite2 = false;
32     boolean gauche2 = false;
33     private ScheduledFuture<?> pacmanTask2; // il va nous servir pour éviter de créer
plusieurs tâche qui se répète et donc d'accumuler des directions : on a
seulement besoin d'une tâche qui fait déplacer le pacman
34
35     //constructeur :
36     public Controleur(Modele modele, Scene scene) {
37         this.modele = modele;
38         this.pacman1= modele.map.pacmans.get(0);
39         if (modele.map.pacmans.size()==2) {
40             this.pacman2 = modele.map.pacmans.get(1);
41         }
42         this.scene = scene;
43         setSceneEventHandler();
44     }
45
46     //méthode qui applique les modifications nécessaires lors de l'appui d'une
touche
```

```

47     private void setSceneEventHandler() {
48         scene.setOnKeyReleased(event -> {
49             // commande du joueur 1
50             //on vérifie que lorsque le joueur appuie sur une touche, le
51             //mouvement est bien possible afin d'éviter des bugs de déplacement
52             if((event.getCode() == KeyCode.UP && pacman1.movePossible(modele
53             .map)[0] != null) || (event.getCode() == KeyCode.DOWN && pacman1.movePossible(
54             modele.map)[1] != null) || (event.getCode()==KeyCode.LEFT && pacman1.
55             movePossible(modele.map)[2] != null) || (event.getCode() == KeyCode.RIGHT &&
56             pacman1.movePossible(modele.map)[3] != null) || event.getCode() == KeyCode.M ) {
57                 //on vérifie que le joueur a fait une demande de changement
58                 //de direction possible (évite tous les problèmes liés au spam d'une direction)
59                 if (memoire != event.getCode() || memoire == KeyCode.M) {
60                     //on enregistre dans la mémoire la dernière touche
61                     active
62                     memoire = event.getCode();
63                     //on regarde le déplacement que doit effectuer le pacman
64                     if (memoire == KeyCode.UP && pacman1.movePossible(modele
65             .map)[0] != null) { //HAUT
66                         haut = true;
67                         bas = false;
68                         droite = false;
69                         gauche = false;
70                     } else if (memoire == KeyCode.DOWN && pacman1.
71             movePossible(modele.map)[1] != null) { //BAS
72                         haut = false;
73                         bas = true;
74                         droite = false;
75                         gauche = false;
76                     } else if (memoire == KeyCode.LEFT && pacman1.
77             movePossible(modele.map)[2] != null) { //GAUCHE
78                         haut = false;
79                         bas = false;
80                         droite = false;
81                         gauche = true;
82                     } else if (memoire == KeyCode.RIGHT && pacman1.
83             movePossible(modele.map)[3] != null) { //DROITE
84                         haut = false;
85                         bas = false;
86                         droite = true;
87                         gauche = false;
88                     }
89                     if (pacmanTask != null) { //évite d'accumuler des tâches
90                         : seul une tâche non nulle doit être active
91                         pacmanTask.cancel(false);
92                     }
93
94                     //répétition du déplacement à une certaine fréquence (
95                     //vitesse du pacman)
96                     pacmanTask = executorService.scheduleAtFixedRate(() -> {
97                         //si le jeu n'a pas été mis en pause
98                         if (!modele.map.pause) {
99                             if (haut && !bas && !gauche && !droite) {
100                                 modele.map.deplacementPacman(pacman1.
101             movePossible(modele.map)[0], modele.board, pacman1);
102                             } else if (!haut && bas && !gauche && !droite) {
103                                 modele.map.deplacementPacman(pacman1.
104             movePossible(modele.map)[1], modele.board, pacman1);
105                             } else if (!haut && !bas && gauche && !droite) {
106                                 modele.map.deplacementPacman(pacman1.
107             movePossible(modele.map)[2], modele.board, pacman1);
108                             } else if (!haut && !bas && !gauche && droite) {
109                                 modele.map.deplacementPacman(pacman1.
110             movePossible(modele.map)[3], modele.board, pacman1);
111                             }
112                         }
113                     }
114                 }
115             }
116         }
117     }

```



```

131                     droite2 = false;
132                     gauche2 = false;
133             } else if (memoire == KeyCode.S && pacman2.movePossible(
134                 modele.map)[1] != null) { //BAS
135                     haut2 = false;
136                     bas2 = true;
137                     droite2 = false;
138                     gauche2 = false;
139             } else if (memoire == KeyCode.Q && pacman2.movePossible(
140                 modele.map)[2] != null) { //GAUCHE
141                     haut2 = false;
142                     bas2 = false;
143                     droite2 = false;
144                     gauche2 = true;
145             } else if (memoire == KeyCode.D && pacman2.movePossible(
146                 modele.map)[3] != null) { //DROITE
147                     haut2 = false;
148                     bas2 = false;
149                     droite2 = true;
150                     gauche2 = false;
151
152             }
153
154         if (pacmanTask2 != null) { //évite d'accumuler des
155             taches : seul une tache non nulle doit être active
156             pacmanTask2.cancel(false);
157         }
158
159         if (modele.map.pacmans.size() == 2) { // déplacement
160             possible si on est en mode multijoueur
161             //répétition du déplacement à une certaine fréquence
162             (vitesse du pacman)
163             pacmanTask2 = executorService.scheduleAtFixedRate(() -
164             {
165                 //si le jeu n'a pas été mis en pause
166                 if (!modele.map.pause) {
167                     if (haut2 && !bas2 && !gauche2 && !droite2)
168                     {
169                         modele.map.deplacementPacman(pacman2.
170                         movePossible(modele.map)[0], modele.board, pacman2);
171                     } else if (!haut2 && bas2 && !gauche2 && !
172                         droite2) {
173                         modele.map.deplacementPacman(pacman2.
174                         movePossible(modele.map)[1], modele.board, pacman2);
175                     } else if (!haut2 && !bas2 && gauche2 && !
176                         droite2) {
177                         modele.map.deplacementPacman(pacman2.
178                         movePossible(modele.map)[2], modele.board, pacman2);
179                     } else if (!haut2 && !bas2 && !gauche2 &&
180                         droite2) {
181                         modele.map.deplacementPacman(pacman2.
182                         movePossible(modele.map)[3], modele.board, pacman2);
183                     }
184                 } else { //si le jeu a été en pause = pacman
185                     mort, donc est initialement immobile
186                     memoire = null;
187                     haut2 = false;
188                     bas2 = false;
189                     gauche2 = false;
190                     droite2 = false;
191                     System.out.println("Le jeu est en pause. Arr

```

```

    ét des mouvements de Pac-Man.");

175                                         modele.map.deplacementPacman(null, modele.
board, pacman2);
176                                         }
177                                         },
178                                         0, pacman2.speed, TimeUnit.MILLISECONDS);
179                                         }

180                                         //pose la bombe "derrière" le pacman
181                                         // dans ce contrôleur si le pacman est immobile entre
182 deux mur il ne peut pas poser de bombe sur le côté : donc à modifier si besoin/
183 possible
184                                         if (memoire == KeyCode.R && pacman2.nbrBombes != 0) {
185                                         if (haut2 && !bas2 && !gauche2 && !droite2 &&
pacman2.positionsPossible[1] != null) { //pose la bombe dans le sens opposé de
la direction si possible
186                                         pacman2.poseBombe(pacman2.positionsPossible[1],
modele.map);
187                                         } else if (!haut2 && bas2 && !gauche2 && !droite2 &&
pacman2.positionsPossible[0] != null) {
188                                         pacman2.poseBombe(pacman2.positionsPossible[0],
modele.map);
189                                         } else if (!haut2 && !bas2 && gauche2 && !droite2 &&
pacman2.positionsPossible[3] != null) {
190                                         pacman2.poseBombe(pacman2.positionsPossible[3],
modele.map);
191                                         } else if (!haut2 && !bas2 && !gauche2 && droite2 &&
pacman2.positionsPossible[2] != null) {
192                                         pacman2.poseBombe(pacman2.positionsPossible[2],
modele.map);
193                                         }
194                                         }
195                                         }
196                                         }
197                                         });
198                                         }
199 }

```

6.3.2 Modèle

```
1 package Modele;
2
3 import java.util.Random;
4 public class Modele {
5     public Map map;
6
7     //Exemple de map
8
9     /*
10     % = mur
11     . = pacgomme
12     P = pacman
13     # = pouvoirs
14     F = fantome
15     @ = bombes posé
16     $ = bombe explosé
17     / = portails
```

```
1 package Modele;
2 import java.util.ArrayList;
3 import java.util.Random;
4 import javafx.animation.PauseTransition;
5 import javafx.util.Duration;
6
7 public class Map {
8     public char[][] board; //on crée un double tableau de caractères à partir d'un
9     string, qui sera notre map
10    public ArrayList<Pacman> pacmans; //on va enregistrer les pacmans dans une liste
11    . Si cette liste est égale à 2, alors on joue en multijoueur
12    public ArrayList<Fantome> fantomes; //on va enregistrer les fantomes dans une
13    liste.
14    public boolean pause = false; // va nous servir pour mettre en pause le jeu après
15    une réinitialisation
16
17    //librement inspiré du tp (adapté à notre problème)
18    public Map(String board){
```

```

15     fantomes = new ArrayList<Fantome>(); // crée nos fantomes
16     pacmans = new ArrayList<Pacman>(); // crée nos pacmans
17     board = board.strip(); // retire les retours à la ligne au début et à la fin
18     du string
19     String[] lines = board.split("\n"); // récupère les lignes de notre map
20     this.board = new char[lines[0].strip().length()][lines.length]; //
21 initialise la taille du tableau
22     for (int i=0; i < lines.length; i++){
23         String line=lines[i].strip();
24         //on enregistre les caractères dans le double tableau board et on récupère
25         //les fantomes et pacmans
26         for (int j=0; j < line.length(); j++){
27             if (line.charAt(j) == 'P') { // on récupère le pacman1
28                 Pacman pacman = new Pacman(j,i, 'P');
29                 pacmans.add(pacman);
30                 this.board[j][i] = 'P';
31             }
32             else if (line.charAt(j) == 'Q') { // on récupère le pacman2
33                 Pacman pacman = new Pacman(j,i, 'Q');
34                 pacmans.add(pacman);
35                 this.board[j][i] = 'Q';
36             }
37             else if (line.charAt(j)=='F') { // on récupère les fantomes
38                 Fantome fantome = new Fantome(j,i);
39                 this.fantomes.add(fantome);
40                 this.board[j][i] = 'F';
41             }
42         }
43     }
44
45     //récupère la longueur de la map
46     public int getWidth() {
47         return this.board.length;
48     }
49
50     //récupère la hauteur de la map
51     public int getHeight() {
52         return this.board[0].length;
53     }
54
55     //récupère l'élément se trouvant à une certaine position
56     public char getElement(Position position){
57         return this.board[position.getX()][position.getY()];
58     }
59
60     //on change un élément du tableau en prenant en paramètre la positon qu'on veut
61     //changer sur la map et la valeur
62     public void setElement(Position position, char value){
63         this.board[position.getX()][position.getY()] = value;
64     }
65
66     //return la position du pacman1
67     public Position findPacmanJoueur1(){
68         for (int k=0; k<board.length; k++){
69             for(int j=0; j<board[0].length; j++){
70                 if(board[k][j] == 'P'){
71                     Position position = new Position(k,j);

```

```

71             return position;
72     }
73 }
74
75 // probleme quand la position du pacman n'est pas trouver important pour IA
76 return new Position(0, 0); // Renvoie une position par défaut, la dernière
77 position valide connue
78 }

79 //return la position du pacman2
80 public Position findPacmanJoueur2(){
81     for (int k=0; k<board.length; k++){
82         for(int j=0; j<board[0].length; j++){
83             if(board[k][j] == 'Q'){
84                 Position position = new Position(k,j);
85                 return position;
86             }
87         }
88     }
89
90 // probleme quand la position du pacman n'est pas trouver important pour IA
91 return new Position(0, 0); // Renvoie une position par défaut, la dernière
92 position valide connue
93 }
94
95 //verifie l'existence ou non d'un pouvoir sur la map
96 public boolean havePouvoir(){
97     for (int k=0; k<board.length; k++){
98         for(int j=0; j<board[0].length; j++){
99             if(this.board[k][j]== '#'){
100                 return true;
101             }
102         }
103     }
104
105     return false;
106 }
107
108 //renitialise la map sans passer par une nouvelle instance;
109 public void reinitialiseMap(String board){
110     fantomes = new ArrayList<Fantome>(); // crée nos fantomes
111     board = board.strip(); // retire les retours à la ligne au début et à la fin
112     du string
113     String[] lines = board.split("\n"); // récupère les lignes de notre map
114     this.board = new char[lines[0].strip().length()][lines.length]; //
115     initialise la taille du tableau
116     for (int i=0; i < lines.length; i++){
117         String line=lines[i].strip();
118         //on enregistre les caractères dans le double tableau board et on récupère
119         // les fantomes et pacmans
120         for (int j=0; j < line.length(); j++){
121             if (line.charAt(j) == 'P') { // on récupère le pacman1
122                 this.pacmans.get(0).position = new Position(j,i);
123                 this.board[j][i] = 'P';
124             }
125             else if (line.charAt(j) == 'Q' && pacmans.size()==2) { //on récupère
126             le pacman2
127                 this.pacmans.get(1).position = new Position(j,i);
128                 this.board[j][i] = 'Q';
129             }
130             else if (line.charAt(j) == 'Q' && pacmans.size()!=2){ // on enlève
131             le pacman2 si on joue pas en multijoueur
132         }
133     }
134 }

```

```

124         this.board[j][i] = '.';
125     }
126     else if (line.charAt(j)=='F') {
127         Fantome fantome = new Fantome(j,i); // on récupère les fantomes
128         this.fantomes.add(fantome);
129         this.board[j][i] = 'F';
130     } else {
131         this.board[j][i] = line.charAt(j);
132     }
133 }
134
135 //on enlève les pouvoirs/malus des pacmans
136 for(int k=0; k<pacmans.size();k++){
137     pacmans.get(k).maudit=false;
138     pacmans.get(k).vulnerable=true;
139     pacmans.get(k).speed = 190;
140     pacmans.get(k).pouvoir = null;
141 }
142
143
144 //on cherche l'autre pacman (celui qui est différent du pacman en paramètre)
145 public Pacman chercheAutrePacman(Pacman pacman){
146     if(pacmans.size()==2) {
147         if (pacman.equals(this.pacmans.get(0))) {
148             return pacmans.get(1);
149         } else {
150             return pacmans.get(0);
151         }
152     }
153     return null;
154 }
155
156 //changement de l'état de la map lors du déplacement du pacman
157 // on prend en paramètre le pacman qui se déplace, la position futur du pacman
158 // et board va nous servir à réinitialiser la map
159 public void deplacementPacman(Position positionFutur, String board, Pacman
160 pacman) {
161     if (pacman.lettre != 'F') { // si le pacman n'a pas été transformé en
162         fantome
163         this.setElement(pacman.position, ' ');
164         if (this.getElement(positionFutur) == '.') { //récupération de la
165             pacgomme
166             pacman.changePoints(30); //augmentation des points
167             pacman.setPosition(positionFutur); //actualise sa position
168             this.setElement(pacman.position, pacman.lettre); //actualise la map
169
170         } else if (this.getElement(positionFutur) == '#') { //récupération d'un
171             pouvoir
172             Pouvoir pouvoir = new Pouvoir(pacman); // crée notre pouvoir
173             pouvoir.givePower(); // et on le donne au pacman
174             pacman.setPosition(positionFutur); // actualise la positon du pacman
175             this.setElement(pacman.position, pacman.lettre); // actualise la map
176
177         } else if (this.getElement(positionFutur) == '/') { //prendre un portail
178             //choix aléatoire de la futur position
179             Random random = new Random();
180             int X = random.nextInt(this.getWidth());
181             int Y = random.nextInt(this.getHeight());
182             Position positionTeleportation = new Position(X, Y);
183             //on vérifie qu'on va se téléporter dans un bon endroit de la map (

```

```

pas un mur, fantome ou autre portail )
179         while (this.getElement(positionTeleportation) == '/' || this.
getElement(positionTeleportation) == 'F' || this.getElement(
positionTeleportation) == '%') {
180             int X1 = random.nextInt(this.getWidth());
181             int Y1 = random.nextInt(this.getHeight());
182             positionTeleportation = new Position(X1, Y1);
183         }
184         //téléportation du pacman
185         pacman.setPosition(positionTeleportation);
186         this.setElement(pacman.position, pacman.lettre);

187     } else if (this.getElement(positionFutur) == '$') { //on arrive sur une
188     bombe qui a explosé
189         //verifie si la bombe lui appartient
190         if (pacman.trouveBombeTouche(positionFutur) != null) {
191             pacman.trouveBombeTouche(positionFutur).toucheBombe(pacman);
192             pause = true; //on le mettra en false dans la classe game
193             //fait une pause
194             PauseTransition pause = new PauseTransition(Duration.millis
(1500));
195             pause.setOnFinished(event -> {
196                 this.reinitialiseMap(board); //réinitialise la map car perdu
197             });
198             pause.play();
199         } else { //si la bombe appartient à l'adversaire
200             Pacman pacmanAdverse = this.chercheAutrePacman(pacman); //on ré
cupère l'autre pacman
201             pacmanAdverse.trouveBombeTouche(positionFutur).toucheBombe(
pacman); //on applique les conséquences
202             pause = true; //on le met en false dans la classe game car c'est
la classe game
203             //fait une pause
204             PauseTransition pause = new PauseTransition(Duration.millis
(1500));
205             pause.setOnFinished(event -> {
206                 this.reinitialiseMap(board); //réinitialise la map car perdu
207             });
208             pause.play();
209         }
210     } else if (this.getElement(positionFutur) == '€') { // position futur =
211     bombe à récupérer
212         this.setElement(pacman.position, ' ');
213         pacman.setPosition(positionFutur); //actualise la position du pacman
214         pacman.ajoutBombe(); //on augmente le compteur de bombe
215         this.setElement(positionFutur, pacman.lettre); //actualise la map

216     } else if (this.getElement(positionFutur)== '@' || positionFutur==null) {
217     //si on arrive sur une bombe qui a été posé mais pas explosé ( elle doit se
comporter comme un mur, donc on ne fait rien )

218     } else{ //c'est tout les autres cas
219         pacman.setPosition(positionFutur); //actualise la position du pacman
220         this.setElement(pacman.position, pacman.lettre); //actualise la map
221     }
222     } else { //si c'est un fantome
223         if (this.getElement(positionFutur) != '%') { //on verifie seulement qu'
on ne se dirige pas vers un mur
224             this.setElement(pacman.position, pacman.lettreFuturPosition);
pacman.lettreFuturPosition=this.getElement(positionFutur);
225         }

```

```

226         pacman.setPosition(positionFutur);
227     }
228
229     }
230 }
231
232 //lorsqu'on a perdu, on devient un fantome
233 public void transformationEnFantome(Pacman pacman){
234     if(pacman.getLife()==0) {
235         pacman.lettre = 'F'; //la lettre du pacman change et on devient un F
236         this.setElement(pacman.position, 'F');
237     }
238 }
239
240 // permet de gérer les collisions avec les fantomes et le pacman-fantome
241 public void collisionFantome(Pacman pacman, String board){
242     //s'il est vulnérable (peut être false si on a pris un pouvoir)
243     if(pacman.vulnerable) {
244         for (int k = 0; k < 4; k++) {
245             if (pacman.position.equals(fantomes.get(k).position) && pacman.
246         lettre != 'F') { //on verifie qu'on est pas un fantome et qu'on a la même
247             position q'un fantome
248                 pacman.PerteVie();
249                 pause = true; //on le met en false dans la classe game car c'est
250             la classe game qui permet de mettre en pause le jeu
251                 reinitialiseMap(board);
252             }
253         }
254     }
255     //pour le mode multijoueur
256     if (pacmans.size()==2) {
257         //cas où l'autre pacman est un fantome
258         if ((chercheAutrePacman(pacman).lettre == 'F' && chercheAutrePacman(
259             pacman).position.equals(pacman.position)) || (chercheAutrePacman(pacman).lettre
260             == 'F' && pacman.position.equals(chercheAutrePacman(pacman))) ) {
261             pacman.PerteVie();
262             chercheAutrePacman(pacman).changePoints(300); // on augmente les
263             points du pacman-fantome
264             pause = true; //on le met en false dans la classe game car c'est la
265             classe game qui permet de mettre en pause le jeu
266             reinitialiseMap(board);
267         }
268     }
269 }
270 }
271
272 }
```

```

1 package Modele;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5 import java.util.concurrent.Executors;
6 import java.util.concurrent.ScheduledExecutorService;
7 import java.util.concurrent.TimeUnit;
8
9 public class Mode {
10     private ScheduledExecutorService executorService = Executors.
11     newSingleThreadScheduledExecutor(); // pour gérer une tache qui s'exécute à une
12     certaine fréquence
13     private ScheduledExecutorService executorService2 = Executors.
14     newSingleThreadScheduledExecutor(); // pour gérer une tache qui s'exécute à une
```

```

certaine fréquence
12     private ScheduledExecutorService executorService3 = Executors.
13     newSingleThreadScheduledExecutor(); // pour gérer une tache qui s'exécute à une
14     certaine fréquence
15     Map map;
16     boolean bomberman ; //boolean pour savoir si le mode bomberman a été activé
17     boolean pouvoir; //boolean pour savoir si le mode pouvoir a été activé
18     boolean portail; //boolean pour savoir si le mode portail a été activé
19     boolean multijoueur; //boolean pour savoir si le multijoueur a été activé
20
21     public Mode(boolean bomberman, boolean pouvoir, boolean portail, boolean
22     multijoueur, Map map){
23         this.map = map;
24         this.bomberman= bomberman;
25         this.pouvoir=pouvoir;
26         this.portail=portail;
27         this.multijoueur = multijoueur;
28     }
29
30     //ajout d'un "pouvoir" dans la map + retourne sa position
31     public Position ajoutPouvoir(){
32         Random random = new Random();
33         int X = random.nextInt(map.getWidth());
34         int Y = random.nextInt(map.getHeight());
35         Position positionPouvoir = new Position(X, Y); // on a crée une position al
36         éatoire
37         while (map.getElement(positionPouvoir) != ' ' && map.getElement(
38             positionPouvoir) != '.') { //on verifie que c'est une bien une positon où on
39             peut mettre le pouvoir ( ne pas remplacer un mur ou un pacman par ex )
40             int X1 = random.nextInt(map.getWidth());
41             int Y1 = random.nextInt(map.getHeight());
42             positionPouvoir = new Position(X1, Y1);
43         }
44         map.setElement(positionPouvoir, '#'); //on modifie la map
45         return positionPouvoir;
46     }
47
48     //gestion du fonctionnement du mode pouvoir ( appartition des pouvoir sur la
49     map )
50     public void modePouvoir(){
51         executorService3.scheduleAtFixedRate(() -> {
52             // si le mode pouvoir a été activé et que la map ne contient pas de
53             pouvoir, on en pose un
54             if(pouvoir && !map.havePouvoir()) {
55                 this.ajoutPouvoir();
56             }
57         }, 5, 7, TimeUnit.SECONDS); //répétiton de cette tache
58     }
59
60     //mode multijoueur : on enlève le pacman en trop car initialement on place , le
61     joueur2 dans notre board
62     public void modeMultijoueur(){
63         if(!multijoueur){
64             map.setElement(map.findPacmanJoueur2(), '.');
65             map.pacmans.remove(1);
66         }
67     }
68
69     //ajout d'un portail dans la map + retourne sa position

```

```

62     public Position ajoutPortail(){
63         Random random = new Random();
64         int X = random.nextInt(map.getWidth());
65         int Y = random.nextInt(map.getHeight());
66         Position positionPortail = new Position(X, Y); // on a crée une position aléatoire
67         while (map.getElement(positionPortail) != ' ' && map.getElement(
68             positionPortail) != '.') { //on vérifie que c'est une bien une positon où on
69             peut mettre le pouvoir ( ne pas remplacer un mur ou un pacman par ex )
70             int X1 = random.nextInt(map.getWidth());
71             int Y1 = random.nextInt(map.getHeight());
72             positionPortail = new Position(X1, Y1);
73         }
74         map.setElement(positionPortail, '/'); //on actualise la map
75         return positionPortail;
76     }
77
78     //gestion des portails ( ajout et disparition des portails dans la map )
79     public void modePortail() {
80         if (portail) {
81             ArrayList<Position> positionsPortails = new ArrayList<>(); // on va
82             garder en mémoire les positions des portails
83             //le compteur va nous servir à savoir quel portail enlever ( en faisant
84             modulo 4 )
85             final int[] compteur = {0}; // on doit utiliser un tableau pour pouvoir
86             l'utiliser dans un lambda expression (suggestion/correction de intelliJ)
87
88             // Planification de la tâche pour ajouter/supprimer un portail toutes
89             les 3 secondes
90             executorService.scheduleAtFixedRate(() -> {
91                 int indice = compteur[0] % 4;
92                 //gestion des 4 premiers portails
93                 if(positionsPortails.size() != 4){
94                     positionsPortails.add(this.ajoutPortail());
95                 }
96                 else { //dés que notre tableau à 4 position de portails, on enlève
97                     et ajoute un portail à chaque fois sur la map
98                     map.setElement(positionsPortails.get(indice), '.');
99                     positionsPortails.set(indice, this.ajoutPortail());
100                }
101                compteur[0] += 1;
102
103            }, 3, 3, TimeUnit.SECONDS);
104        }
105    }
106
107    //ajoute une bombe récupérable dans la map
108    public Position ajoutBombeArecup(){
109        Random random = new Random();
110        int X = random.nextInt(map.getWidth());
111        int Y = random.nextInt(map.getHeight());
112        Position positionBombe = new Position(X, Y); // position aléatoire de la
113        bombe
114        while (map.getElement(positionBombe) != ' ' && map.getElement(positionBombe)
115        != '.') { //on vérifie que c'est une postion possible
116            int X1 = random.nextInt(map.getWidth());
117            int Y1 = random.nextInt(map.getHeight());
118            positionBombe = new Position(X1, Y1);
119        }
120        map.setElement(positionBombe, '€'); // on actualise la map

```

```

112         return positionBombe;
113     }
114
115     // "gestion" du mode bomberman ( apparition des bombes ) (pour l'explosion et la
116     // disparition des bombes, c'est gérer dans la classe pacman car c'est le pacman
117     // qui fait exploser la bombe)
118     public void modeBomberman() {
119         if (bomberman) {
120             // on ajoute les bombes initiale au pacmans
121             for(int k=0; k<map.pacmans.size(); k++){
122                 map.pacmans.get(k).nbrBombes= 3;
123             }
124             // Planification de la tâche pour ajouter une bombe à récupérer toutes
125             // les 10 secondes
126             executorService2.scheduleAtFixedRate(() -> {
127                 Position positionBombe = this.ajoutBombeArecup();
128                 map.setElement(positionBombe, '€');
129             }, 5, 10, TimeUnit.SECONDS);
130         }
131     }

```

```

1 package Modele;
2 public class Position {
3     public int X;
4     public int Y;
5     public Position(int X,int Y){
6         this.X=X;
7         this.Y=Y;
8     }
9     public int getX(){
10         return X;
11     }
12     public int getY(){
13         return Y;
14     }
15     public void setX(int X){
16         this.X=X;
17     }
18     public void setY(int Y){
19         this.Y=Y;
20     }
21
22     //méthode pour vérifier que deux positions sont égales
23     @Override
24     public boolean equals(Object o) {
25         if (this == o){
26             return true;
27         }
28         if (o == null || getClass() != o.getClass()){
29             return false;
30         }
31         Position position = (Position) o;
32         return X == position.X && Y == position.Y;
33     }
34 }


```

```

1 package Modele;
2

```

```

3 public class Personnage { // soit fantome ou pacman
4     public Position position; // position de notre personnage
5     public Position[] positionsPossible; // déplacement possible Haut,Bas,Gauche,
6     Droite
7
8     public Personnage(int X, int Y) {
9         position = new Position(X, Y);
10        positionsPossible = new Position[4];
11    }
12
13    public Position getPosition() {
14        return position;
15    }
16    public void setPosition(Position position) {
17        this.position = position;
18    }
19    public String toString() {
20        return "(" + position.getX() + ";" + position.getY() + ")";
21    }
22
23    // collision avec le mur + move possible
24    public Position[] colisionMur(Map map) {
25        Position Haut = new Position(this.position.getX(), this.position.getY() - 1);
26        // création de la position haut
27        Position Bas = new Position(this.position.getX(), this.position.getY() + 1);
28        // création de la position bas
29        Position Droite = new Position(this.position.getX() + 1, this.position.getY());
30        // création de la position droite
31        Position Gauche = new Position(this.position.getX() - 1, this.position.getY());
32        // création de la position gauche
33        Position[] Direction = {Haut, Bas, Gauche, Droite};
34        for (int k = 0; k < Direction.length; k++) {
35            if (map.getElement(Direction[k]) == '%') { // si la direction k est un
36                mur, on dit que le mouvement n'est pas possible
37                this.positionsPossible[k] = null;
38            } else {
39                this.positionsPossible[k] = Direction[k];
40            }
41        }
42        return this.positionsPossible;
43    }
44}

```

```

1 package Modele;
2
3 import java.util.ArrayList;
4 import java.util.Timer;
5 import java.util.TimerTask;
6
7 public class Pacman extends Personnage {
8     public int points; //compteur de points
9     public int life; // nombre de vie
10    public int speed = 200; //vitesse
11

```

```

12     public boolean vulnerable = true; //s'il peut etre attaqué par des fantomes ( pour le mode pouvoir )
13     // pour le mode pouvoir :
14     public boolean maudit = false; //s'il est préféré par les fantomes + le trouvent plus rapidement (pour le mode pouvoir)
15     public int nbrBombes = 0;
16     public char lettre; //permet de savoir si c'est le joueur 1,2 ou un fantome
17     public char lettreFuturPosition; //permet de récupérer la lettre qu'il remplace lorsque qu'il se déplace (quand c'est un fantome)
18     public ArrayList<BombeExplosive> bombes = new ArrayList<>(); //garde en mémoire les bombes qu'il a posé et qui ont explosé
19     public String pouvoir; //on récupère le nom du pouvoir (s'il en a pris un )
20     public int numero;
21
22     public Pacman(int X, int Y, char lettre) {
23         super(X, Y);
24         this.life=3;
25         this.lettre = lettre;
26         if(lettre == 'P') {
27             this.numero = 0; //joueur1
28         } else {
29             this.numero = 1; //joueur2
30         }
31     }
32     public int getLife(){
33         return life;
34     }
35     public void PerteVie(){
36         this.life -= 1;
37     }
38     public void changePoints(int points){
39         this.points += points;
40     }
41     public void ajoutBombe(){
42         nbrBombes++;
43     }
44     public void addSpeed(int add ){
45         speed+=add;
46     } //modification de la vitesse en ajoutant un nombre ( qui peut être négatif également )
47
48     //fait poser une bombe sur la map par le pacman et gère son explosion et sa disparition
49     public void poseBombe(Position position, Map map){
50         if (nbrBombes != 0) { // on vérifie qu'il a bien le droit de poser une bombe
51             nbrBombes--; // on décrémente le compteur
52
53             map.setElement(position, '@'); //on actualise la map ( on pose une bombe mais pas explosé encore )
54             BombeExplosive bombeRamasse = new BombeExplosive(position.getX(), position.getY()); //on crée notre bombe
55
56             // Faire exploser la bombe après un certain temps
57             Timer timer = new Timer();
58             timer.schedule(new TimerTask() {
59                 @Override
60                 public void run() {
61                     bombeRamasse.pacmanAppartient = Pacman.this; // affecte à la propriété pacmanAppartient de l'objet bombeRamasse une référence à l'instance actuelle de la classe Pacman

```

```

62         bombes.add(bombeRamasse); // Ajouter la bombe à la liste de
bombes
63         bombeRamasse.explose(map); // on fait explosé la bombe
64         timer.cancel();
65     }
66 }, bombeRamasse.delaiAvantExplosion); // Délai avant explosion

67
68 //disparition de la bombe après un certain temps
69 Timer timer2 = new Timer();
70 timer2.schedule(new TimerTask() {
71     @Override
72     public void run() {
73         bombeRamasse.dispariton(map); //on fait disparaître la bombe
74         bombes.remove(bombeRamasse); // on enlève la bombe de notre
liste
75         timer2.cancel();
76     }
77 }, bombeRamasse.delaiAvantExplosion + bombeRamasse.delaiExplosion); // D
élai disparition
78 }
79 }

80
81 //permet de savoir si la bombe qu'on a touché est la notre ou pas ( ie si c'est
pas la notre, c'est que c'est à l'autre pacman )
82 // et permet de savoir quels bombes à été touché
83 public BombeExplosive trouveBombeTouche(Position position){
84     for(int k=0; k<bombes.size(); k++){
85         for(int j=0; j<bombes.get(k).positions.size(); j++){
86             if(position.equals(bombes.get(k).positions.get(j))){
87                 return bombes.get(k);
88             }
89         }
90     }
91     return null;
92 }
93 }

```

```

1 package Modele;
2 import java.util.ArrayList;
3 import java.util.List;
4 import java.util.Random;
5
6 public class Fantome extends Personnage {
7     private Position lastPosition = null; // Utiliser pour eviter les probleme d'
impasse
8     public int DETECTION_RADIUS = 5; //Rayon de detection
9     public Fantome(int X, int Y) {
10         super(X, Y);
11     }
12     public int distanceToPacman(Position pacmanPos) {// Calcule la distance entre le
Pacman et le fantome
13         return Math.abs(this.position.getX() - pacmanPos.getX()) + Math.abs(this.
position.getY() - pacmanPos.getY());
14     }
15     private Position moveToPacman(Position pacmanPos, List<Position> Movesfontome) {
16         // Oriente le fantome vers le Pacman en se basant sur la distance
17         Position closestMove = null;
18         int minDistance = Integer.MAX_VALUE;
19
20         for (Position move : Movesfontome) {// Choisi le move qui permet de

```

```

minimiser la distance
    int distance = Math.abs(move.getX() - pacmanPos.getX()) + Math.abs(move.
getY() - pacmanPos.getY());
    if (distance < minDistance) {// Choisi la distance minimal
        closestMove = move;
        minDistance = distance;
    }
}

if (closestMove != null) {// Si il y a une distance minimal
    return closestMove;
} else {
    return this.position;
}
}

public boolean fantomeBehind(Position pacmanPos, Position vecteurDrirecteur) {
    // Vérifiez si Pacman est derrière le fantôme en projetant le vecteur
    // directeur sur le vecteur du fantôme à Pacman
    Position FantomeToPacman = new Position(pacmanPos.getX() - this.position.
getX(), pacmanPos.getY() - this.position.getY());
    int ProduitScalaire = vecteurDrirecteur.getX() * FantomeToPacman.getX() +
vecteurDrirecteur.getY() * FantomeToPacman.getY(); // produit scalaire(Projection
)
    return ProduitScalaire < 0; //Si le produit scalaire est négative alors il
    est en arrière
}
// Méthode random La plus avancé
public void moveRandomly(Map map) {
    Position[] possibleMoves = movePossible(map);
    List<Position> Movesfontome = new ArrayList<>();
    Position pacmanPos = null;

    //1er cas : les deux pacman ne sont pas maudit
    //on choisit le pacman le plus proche des deux si on joue en multijoueur
    if (map.pacmans.size() == 2) {
        //1er cas : les deux pacmans ne sont pas maudit
        //on choisit le pacman le plus proche des deux
        if (!map.pacmans.get(0).maudit && !map.pacmans.get(1).maudit) {
            if (distanceToPacman(map.findPacmanJoueur1()) > distanceToPacman(map.
findPacmanJoueur2())) {
                pacmanPos = map.findPacmanJoueur2();
                DETECTION_RADIUS = 5; //Rayon de détection
            } else {
                pacmanPos = map.findPacmanJoueur1();
                DETECTION_RADIUS = 5; //Rayon de détection
            }
        }
        } else if (map.pacmans.get(0).maudit && !map.pacmans.get(1).maudit){//2è
me cas : le joueur 1 est maudit : donc il est préféré par les fantomes et on
augmenter les "cercles" des fantomes
            pacmanPos = map.findPacmanJoueur1();
            DETECTION_RADIUS = 10; //Rayon de détection
        }
        } else if (!map.pacmans.get(0).maudit && map.pacmans.get(1).maudit){//3è
me cas : le joueur 2 est maudit : donc il est préféré par les fantomes et on
augmenter les "cercles" des fantomes
            pacmanPos = map.findPacmanJoueur2();
            DETECTION_RADIUS = 10; //Rayon de détection
        }
}

```

```

69
70     }
71 } else { //cas où on joue en solo
72     pacmanPos = map.findPacmanJoueur1();
73     //s'il est maudit (pour le mode pouvoir)
74     if (map.pacmans.get(0).maudit){
75         DETECTION_RADIUS = 10; //Rayon de detection
76     } else {
77         DETECTION_RADIUS = 5; //Rayon de detection
78     }
79 }
80
81 Position vecteurdurecteur = null;
82 if (possibleMoves.length == 0) {
83     return; // Aucun mouvement possible, sortie anticipée
84 }
85
86 // Récupère les mouvements possibles et identifie le mouvement inverse
87 for (Position move : possibleMoves) {
88     if (move != null && !move.equals(lastPosition)) {
89         Movesfontome.add(move);
90         // Calcule la direction de chaque mouvement possible par rapport à
91         // la position actuelle
92         if (vecteurdurecteur == null) { // Initialise vecteurdurecteur avec
93             le premier mouvement valide
94             vecteurdurecteur = new Position(move.getX() - this.position.getX(),
95             move.getY() - this.position.getY());
96         }
97     }
98 }
99
100 if (Movesfontome.isEmpty()) {
101     if (lastPosition != null) {
102         this.setPosition(lastPosition); // Si il y a pas autre position
103         retourne à ta position d'avant
104         return;
105     } else {
106         return; // If no last position available, no movement can be made (stuck at a corner or enclosed space)
107     }
108 }
109
110 Position PositionChoisi = this.position; // Par défaut reste en place
111
112 // Si Pacman est détecté à proximité
113 if (distanceToPacman(pacmanPos) <= DETECTION_RADIUS) {
114     // Si Pacman est derrière, inclure la direction inverse dans les choix
115     possibles
116     if (lastPosition != null && fantomeBehind(pacmanPos, vecteurdurecteur))
117     {
118         Movesfontome.add(lastPosition);
119     }
120
121     // Choisir la position la plus proche de Pacman
122     PositionChoisi = moveToPacman(pacmanPos, Movesfontome);
123 } else if (!Movesfontome.isEmpty()) {
124     // Choisir un mouvement aléatoire parmi les options valides
125     Random rand = new Random();
126     PositionChoisi = Movesfontome.get(rand.nextInt(Movesfontome.size()));
127 }

```

```

122     lastPosition = this.position; // Mettre à jour la dernière position
123     this.setPosition(PositionChoisi); // Déplacer le fantôme
124
125 }
126
127 }

1 package Modele;
2 import java.util.Random;
3 import java.util.concurrent.Executors;
4 import java.util.concurrent.ScheduledExecutorService;
5 import javafx.animation.PauseTransition;
6 import javafx.util.Duration;
7 public class Pouvoir {
8     Pacman pacman; // pour associer le pouvoir à un pacman
9     String pouvoir; // nom du pouvoir
10
11     public Pouvoir(Pacman pacman) {
12         //liste des pouvoirs possibles
13         String[] pouvoirs = new String[]{"changementVitesse", "invulnérable", "malédition"};
14
15         //choix aléatoire du pouvoir
16         Random random = new Random();
17         int i = random.nextInt(pouvoirs.length);
18         this.pouvoir=pouvoirs[i];
19         this.pacman = pacman;
20     }
21
22     public Pacman getPacman() {
23         return pacman;
24     }
25
26     //on applique le pouvoir au pacman
27     public void givePower(){
28         if (pacman.pouvoir != null) { //si on a déjà un pouvoir, prendre un deuxième
29             pouvoir efface ce qu'on a déjà
30             pacman.vulnerable = true;
31             pacman.speed = 190;
32             pacman.maudit = false;
33             pacman.pouvoir = null;
34         } else { //si on a pas de pouvoir
35             if (pouvoir.equals("invulnérable")){ //le cas où a eu le pouvoir invulnérable
36                 pacman.vulnerable=false;
37                 pacman.pouvoir = "Invulnérabilité !";
38                 //pour réactiver la vulnérabilité après 10 secondes
39                 PauseTransition attente1 = new PauseTransition(Duration.seconds(10))
40 ;
41                 attente1.setOnFinished(event -> {
42                     pacman.vulnerable = true;
43                     pacman.pouvoir = null;
44                 });
45                 attente1.play();
46             } else if (pouvoir.equals("changementVitesse")) { // le cas où a eu le
47             pouvoir changement Vitesse
48                 Random random = new Random();
49                 int randomNumber = random.nextInt(2); // génère 0 ou 1
50                 int result = (randomNumber == 0) ? -1 : 1; // convertit 0 en -1 et 1
51                 en 1 ( si on a 1, on augmente la vitesse et si on a -1 on diminue la vitesse )
52             }
53         }
54     }
55 }

```

```

48         pacman.addSpeed(result*100);
49         pacman.pouvoir = "Changement de vitesse !";
50         // pour remettre la vitesse de base après 10 secondes
51         PauseTransition attente2 = new PauseTransition(Duration.seconds(10))
52 ;
53         attente2.setOnFinished(event -> {
54             pacman.speed = 190;
55             pacman.pouvoir = null;
56         });
57         attente2.play();
58     }
59     else if (pouvoir.equals("malédiction")){ //le cas où on a eu le malus
60     malédiction
61         pacman.maudit = true;
62         pacman.pouvoir = "Malédiction (sur lui-même !)";
63         // pour enlever la malédiction après 10 secondes
64         PauseTransition attente3 = new PauseTransition(Duration.seconds(10))
65 ;
66         attente3.setOnFinished(event -> {
67             pacman.maudit = false;
68             pacman.pouvoir = null;
69         });
70         attente3.play();
71     }
72 }
73 }
```

```

1 package Modele;
2
3 import java.util.ArrayList;
4 import java.util.Random;
5
6 public class BombeExplosive {
7     public Pacman pacmanAppartient; //on garde en mémoire à qui appartient la bombe
8     pour pouvoir faire perdre des vies et gagner/perdre des points
9     public ArrayList<Position> positions; // une position s'il n'a pas encore explosé
10    et 5 positions max s'il a explosé
11    public int damage = 100; // points gagné si on touche son adversaire et points
12    perdu si on s'auto touche
13    public int delaiAvantExplosion = 3000; //millisecondes delai avant explosion
14    public int delaiExplosion = 5000; //millisecondes delai explosion
15
16    public BombeExplosive(int X, int Y ) {
17        Position position = new Position(X,Y);
18        positions = new ArrayList<Position>();
19        positions.add(position);
20    }
21
22    public void explose(Map map){ // fait explosé la bombe en lui "rajoutant" des
23    positions
24
25        //ajout des positions selon la règle de la croix s'il n'y a pas de mur ou
26        pacman ou fantome ou portail ou pouvoir
27        Position positionHaut = new Position(positions.get(0).getX(), positions.get
28        (0).getY()+1);
29        Position positionBas = new Position(positions.get(0).getX(), positions.get
30        (0).getY()-1);
31        Position positionGauche = new Position(positions.get(0).getX()-1, positions.
32        get(0).getY());
```

```

25     Position positionDroite = new Position(positions.get(0).getX()+1, positions.
get(0).getY());
26
27     if(map.getElement(positionHaut) == ' ' || map.getElement(positionHaut)=='.'){
//ajout des positions selon la règle de la croix s'il n'y a pas de mur ou
pacman ou fantome ou portail ou pouvoir
        positions.add(positionHaut);
    }
28     if(map.getElement(positionBas) == ' ' || map.getElement(positionBas)=='.'){
//ajout des positions selon la règle de la croix s'il n'y a pas de mur ou pacman
ou fantome ou portail ou pouvoir
        positions.add(positionBas);
    }
29     if(map.getElement(positionDroite) == ' ' || map.getElement(positionDroite)=='.'){
//ajout des positions selon la règle de la croix s'il n'y a pas de mur ou
pacman ou fantome ou portail ou pouvoir
        positions.add(positionDroite);
    }
30     if(map.getElement(positionGauche) == ' ' || map.getElement(positionGauche)=='.'){
//ajout des positions selon la règle de la croix s'il n'y a pas de mur ou
pacman ou fantome ou portail ou pouvoir
        positions.add(positionGauche);
    }
31     for(int k=0;k<4;k++){
        map.setElement(positions.get(k), '$'); //on actualise la map
    }
32
33 }
34
35
36
37
38
39
40
41
42
43
44
45 // vérifie si il y a une collision avec un pacman
46 public void toucheBombe(Pacman pacman){
47     if(pacman==this.pacmanAppartient){ //si le pacman touché est le propriétaire
de la bombe
        pacman.changePoints(-damage); //on perd des points
        pacman.PerteVie(); //on perd une vie
    } else { // si le pacman touché n'est pas le propriétaire de la bombe
        pacmanAppartient.changePoints(damage); //on augmente les points du
propriétaire
        pacman.PerteVie();
    }
48 }
49
50
51
52
53
54
55
56 //fait disparaître la bombe de la map
57 public void disparition(Map map){
58     for(int k=0; k<positions.size(); k++){
        map.setElement(positions.get(k), '.');
    }
59 }
60
61 }
62
63 }

```

6.3.3 Vue

```

1 package Vue;
2
3
4 import Controleur.Controleur;
5 import Modele.*;

```

```

6 import javafx.animation.AnimationTimer;
7 import javafx.application.Application;
8 import javafx.application.Platform;
9 import javafx.geometry.Insets;
10 import javafx.geometry.Pos;
11 import javafx.scene.Scene;
12 import javafx.scene.canvas.Canvas;
13 import javafx.scene.canvas.GraphicsContext;
14 import javafx.scene.control.Button;
15 import javafx.scene.control.Label;
16 import javafx.scene.image.Image;
17 import javafx.scene.image.ImageView;
18 import javafx.scene.input.KeyCode;
19 import javafx.scene.layout.HBox;
20 import javafx.scene.layout.VBox;
21 import javafx.scene.paint.Color;
22 import javafx.scene.text.Font;
23 import javafx.stage.Stage;

24
25
26
27 public class Game extends Application {
28
29
30     private static Modele modele = new Modele(); // creation du modele du jeu
31     private static Mode mode = new Mode(Menu.selected_bombe, Menu.selected_pouvoir,
32         Menu.selected_portail, Menu.multijoueur, modele.map); // recuperation des modes
33         choisis dans le menu
34     private boolean fin = false; // permet de savoir si le jeu est fini
35
36     private static final int TILE_SIZE = 30; // taille d'une cellule de la grille
37     private final int WIDTH = modele.map.getWidth(); // nombre de cellules dans la
38         grille (largeur)
39     private final int HEIGHT = modele.map.getHeight(); // nombre de cellules dans la
40         grille (hauteur)
41     private long lastUpdateTime = System.currentTimeMillis(); // heure de la
42         derniere mise a jour de l'interface graphique
43     private final long UPDATE_INTERVAL = 190; // delai avant reactualisation fantome
44         (permet de regler la vitesse des fantomes)
45
46     private Stage endStage = new Stage(); // stage affiche en fin de partie
47
48     private boolean showDetectionRadius = false; // pour dessiner les cercles des
49         fantomes (voir fonctionnement ia fantome)
50
51     // chargement des ressources necessaires au jeu
52     public Image pacman1Image = new Image(getClass().getResourceAsStream("Ressources/
53         /pacman1droite.gif"));
54     public Image pacman2Image = new Image(getClass().getResourceAsStream("Ressources/
55         /pacman2gauche.gif"));
56     public ImageView pacman1View = new ImageView(pacman1Image);
57     private Image fantomeImage = new Image(getClass().getResourceAsStream("Ressources/
58         /fantome.png"));
59     private Image portailImage = new Image(getClass().getResourceAsStream("Ressources/
60         /portail.png"));
61     private Image bombePoseeImage = new Image(getClass().getResourceAsStream("Ressources/
62         /bombeposee.png"));
63     private Image bombeRamasserImage = new Image(getClass().getResourceAsStream("Ressources/
64         /bomberamasser.png"));
65     private Image pouvoiriImage = new Image(getClass().getResourceAsStream("Ressources/
66         /pouvoiri.png"));

```

```

Ressources/pouvoir.png"));

53
54     @Override
55     public void start(Stage primaryStage) { // methode appelee lors du lancement de
l'application

56         VBox root = new VBox(40); // vertical box qui contient les elements
graphiques du jeu
57         VBox score1 = new VBox(10); // contient le score du joueur 1
58         VBox score2 = new VBox(10); // contient le score du joueur 2
59         VBox annonce =new VBox(10); // permet d'afficher le pouvoir utilise si le
mode pouvoir est active
60         HBox info = new HBox(200); // horizontal box qui contient les scores et l'
annonce

61         root.setPadding(new Insets(25));
62         root.setAlignment(Pos.CENTER);
63         root.setStyle("-fx-background-color: BLACK");

64         Canvas canvas = new Canvas(WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE); // //
canvas pour dessiner le jeu

65
66         Image titre = new Image(getClass().getResourceAsStream("Ressources/
PacmanLogo.png")); // chargement du titre
67         ImageView titreView = new ImageView(titre);
68         titreView.setFitWidth(500);
69         titreView.setFitHeight(150);

70
71         Label scorej1 = new Label(); // affichage du score du joueur 1
72         scorej1.setTextFill(Color.GHOSTWHITE);
73         scorej1.setFont(new Font("Serif", 20));
74         Label lifej1 = new Label(); // affichage du nombre de vie du
joueur 1
75         lifej1.setTextFill(Color.GHOSTWHITE);
76         lifej1.setFont(new Font("Serif", 20));
77         Label bombej1 = new Label(); // affichage du nombre de bombe du
joueur 1
78         bombej1.setTextFill(Color.GHOSTWHITE);
79         bombej1.setFont(new Font("Serif", 20));
80         Label pseudoj1 = new Label(); // affichage du pseudo du joueur 1
81         pseudoj1.setTextFill(Color.GHOSTWHITE);
82         pseudoj1.setFont(new Font("Serif", 20));

83
84         score1.getChildren().addAll(pseudoj1, scorej1, lifej1); // ajout des labels
dans score1
85         if(Menu.selected_bombe) {
86             score1.getChildren().add(bombej1);
87         }

88         Label scorej2 = new Label(); // affichage du score du joueur 2
89         scorej2.setTextFill(Color.GHOSTWHITE);
90         scorej2.setFont(new Font("Serif", 20));
91         Label lifej2 = new Label(); // affichage du nombre de vie du
joueur 2
92         lifej2.setTextFill(Color.GHOSTWHITE);
93         lifej2.setFont(new Font("Serif", 20));
94         Label bombej2 = new Label(); // affichage du nombre de bombe du
joueur 2
95         bombej2.setTextFill(Color.GHOSTWHITE);
96         bombej2.setFont(new Font("Serif", 20));

```

```

101     Label pseudoj2 = new Label();           // affichage du pseudo du joueur 2
102     pseudoj2.setTextFill(Color.GHOSTWHITE);
103     pseudoj2.setFont(new Font("Serif", 20));
104
105     Label annonce1 = new Label();           // affichage des pouvoirs du joueur
1
106     annonce1.setTextFill(Color.GHOSTWHITE);
107     annonce1.setFont(new Font("Serif", 20));
108     annonce1.setVisible(true);
109
110     Label annonce2 = new Label();           // affichage des pouvoirs du joueur
2
111     annonce2.setTextFill(Color.GHOSTWHITE);
112     annonce2.setFont(new Font("Serif", 20));
113     annonce2.setVisible(true);
114
115     annonce.getChildren().addAll(annonce1, annonce2); // ajout des labels dans
116     annonce
117
118     info.setAlignment(Pos.CENTER);
119     info.getChildren().addAll(score1, annonce);      // ajout des box dans info
120
121     if(Menu.multijoueur == true) {
122         score2.getChildren().addAll(pseudoj2, scorej2, lifej2);
123         if (Menu.selected_bombe) {
124             score2.getChildren().add(bombej2);
125         }
126         info.getChildren().add(score2);                // ajout de score2 si le mode
127         multijoueur est active
128     }
129
130     root.getChildren().addAll(titreView, info, canvas); //ajout des box et du
131     canvas sur la toile
132
133     Scene scene = new Scene(root, 1400, 900);        //création d'une scène
134     montrant la toile
135
136     // pour afficher les cercles
137     scene.setOnKeyPressed(event -> {
138         if (event.getCode() == KeyCode.SPACE) {          // cliquer sur espace pour
139             afficher les cercles
140             showDetectionRadius = !showDetectionRadius;
141         }
142         // chargement des bons sprites en fonction des déplacements et du modèle
143         if (event.getCode() == KeyCode.UP) {
144             if(modele.map.pacmans.get(0).life == 0) {
145                 pacman1Image = new Image(getClass().getResourceAsStream("Ressources/morthaut.gif"));
146             } else {
147                 if(modele.map.pacmans.get(0).pouvoir != null) {
148                     pacman1Image = new Image(getClass().getResourceAsStream("Ressources/spacman1haut.gif"));
149                 } else {
150                     pacman1Image = new Image(getClass().getResourceAsStream("Ressources/pacman1haut.gif"));
151                 }
152             }
153         } if (event.getCode() == KeyCode.DOWN) {
154             if(modele.map.pacmans.get(0).life == 0) {
155                 pacman1Image = new Image(getClass().getResourceAsStream("Ressources/morthaut.gif"));
156             }
157         }
158     }
159 
```

```

        Ressources/mortbas.gif"));
    } else {
        if(modele.map.pacmans.get(0).pouvoir != null) {
            pacman1Image = new Image(getClass().getResourceAsStream("Ressources/spacman1bas.gif"));
        } else {
            pacman1Image = new Image(getClass().getResourceAsStream("Ressources/pacman1bas.gif"));
        }
    }
} if (event.getCode() == KeyCode.LEFT) {
    if(modele.map.pacmans.get(0).life == 0) {
        pacman1Image = new Image(getClass().getResourceAsStream("Ressources/mortgauche.gif"));
    } else {
        if(modele.map.pacmans.get(0).pouvoir != null) {
            pacman1Image = new Image(getClass().getResourceAsStream("Ressources/spacman1gauche.gif"));
        } else {
            pacman1Image = new Image(getClass().getResourceAsStream("Ressources/pacman1gauche.gif"));
        }
    }
} if (event.getCode() == KeyCode.RIGHT) {
    if(modele.map.pacmans.get(0).life == 0) {
        pacman1Image = new Image(getClass().getResourceAsStream("Ressources/mortdroite.gif"));
    } else {
        if(modele.map.pacmans.get(0).pouvoir != null) {
            pacman1Image = new Image(getClass().getResourceAsStream("Ressources/spacman1droite.gif"));
        } else {
            pacman1Image = new Image(getClass().getResourceAsStream("Ressources/pacman1droite.gif"));
        }
    }
} if (event.getCode() == KeyCode.Z) {
    if(Menu.multijoueur == true) {
        if(modele.map.pacmans.get(1).life == 0) {
            pacman2Image = new Image(getClass().getResourceAsStream("Ressources/morthaut.gif"));
        } else {
            if(modele.map.pacmans.get(1).pouvoir != null) {
                pacman2Image = new Image(getClass().getResourceAsStream("Ressources/spacman2haut.gif"));
            } else {
                pacman2Image = new Image(getClass().getResourceAsStream("Ressources/pacman2haut.gif"));
            }
        }
    }
} if (event.getCode() == KeyCode.S) {
    if(Menu.multijoueur == true) {
        if(modele.map.pacmans.get(1).life == 0) {
            pacman2Image = new Image(getClass().getResourceAsStream("Ressources/mortbas.gif"));
        } else {
            if(modele.map.pacmans.get(1).pouvoir != null) {
                pacman2Image = new Image(getClass().getResourceAsStream("Ressources/spacman2bas.gif"));
            }
        }
    }
}

```

```

197             } else {
198                 pacman2Image = new Image(getClass().getResourceAsStream("Ressources/pacman2bas.gif"));
199             }
200         }
201     }
202     if (event.getCode() == KeyCode.Q) {
203         if(Menu.multijoueur == true) {
204             if(modele.map.pacmans.get(1).life == 0) {
205                 pacman2Image = new Image(getClass().getResourceAsStream("Ressources/mortgauche.gif"));
206             } else {
207                 if(modele.map.pacmans.get(1).pouvoir != null) {
208                     pacman2Image = new Image(getClass().getResourceAsStream("Ressources/spacman2gauche.gif"));
209                 } else {
210                     pacman2Image = new Image(getClass().getResourceAsStream("Ressources/pacman2gauche.gif"));
211                 }
212             }
213         }
214     if (event.getCode() == KeyCode.D) {
215         if(Menu.multijoueur == true) {
216             if(modele.map.pacmans.get(1).life == 0) {
217                 pacman2Image = new Image(getClass().getResourceAsStream("Ressources/mortdroite.gif"));
218             } else {
219                 if(modele.map.pacmans.get(1).pouvoir != null) {
220                     pacman2Image = new Image(getClass().getResourceAsStream("Ressources/spacman2droite.gif"));
221                 } else {
222                     pacman2Image = new Image(getClass().getResourceAsStream("Ressources/pacman2droite.gif"));
223                 }
224             }
225         }
226     });
227
228 // lancement des modes en fonction des choix effectues dans le menu
229 mode.modePortail();
230 mode.modeBomberman();
231 mode.modePouvoir();
232 mode.modeMultijoueur();
233
234 Controleur controleur = new Controleur(modele, scene); // creation d'un
235 controleur pour les joueurs
236 GraphicsContext gc = canvas.getGraphicsContext2D(); // variable contenante
237 le contexte graphique du canvas
238
239 new AnimationTimer() { // permet l'actualisation du jeu
240     private long lastUpdate = 0;
241
242     @Override
243     public void handle(long now) { // gère les mises à jour de l'affichage
244         avec l'actualisation du jeu
245
246         if (now - lastUpdate >= 1_000_000) { // actualisation toutes les
247             10ms

```

```

246         update();           // methode update a completer par les
autres membres en cas de besoin
247         draw(gc);          // permet de dessiner le canvas
248         long currentTime = System.currentTimeMillis();
249
250         // mise a jour des labels
251         updateScore(scorej1,0);
252         updateLife(lifej1,0);
253         updateBombe(bombej1,0);
254         updatePseudo(pseudoj1,0);
255         updateAnnonce(annonce1, 0);
256
257         if (Menu.multijoueur == true) {
258             updateScore(scorej2,1);
259             updateLife(lifej2,1);
260             updateBombe(bombej2,1);
261             updatePseudo(pseudoj2,1);
262             updateAnnonce(annonce2,1);
263         }
264
265         // gere la transformation en fantome en cas de mort
266         for(int k=0; k<modele.map.pacmans.size();k++){
267             modele.map.transformationEnFantome(modele.map.pacmans.get(k));
268         }
269
270         // mise à jour de chaque fantôme
271         if (currentTime - lastUpdateTime > UPDATE_INTERVAL) {
272             for(int k=0; k<4; k++) {
273                 modele.map.fantomes.get(k).moveRandomly(modele.map);
274             }
275             lastUpdateTime = currentTime;
276         }
277
278         // creation des pauses entre les reinitialisations
279         if (modele.map.pause) {
280             try {
281                 // Met le thread en pause pour 1 seconde
282                 Thread.sleep(1000);
283             } catch (InterruptedException e) {
284                 throw new RuntimeException(e);
285             }
286             modele.map.pause = false; // fin de la pause
287         }
288
289         // gestion de la fin du jeu
290         if(modele.isEnd() && fin == false){
291             fin = true;
292
293             //chargement des gif finaux
294             pacman1Image = new Image(getClass().getResourceAsStream("Ressources/pacman1droite.gif"));
295             ImageView pacman1View = new ImageView(pacman1Image);
296             pacman1View.setFitWidth(100);
297             pacman1View.setFitHeight(100);
298
299             pacman2Image = new Image(getClass().getResourceAsStream("Ressources/pacman2droite.gif"));
300             ImageView pacman2View = new ImageView(pacman2Image);
301             pacman2View.setFitWidth(100);

```

```

302         pacman2View.setFitHeight(100);
303
304         primaryStage.close(); // ferme la fenetre de jeu
305
306         VBox endScreenRoot = new VBox(50); // vertical box qui
307         contient les elements graphiques de l'ecran de fin
308         VBox res = new VBox(30); // vertical box qui affiche
309         les informations du gagnant
310         HBox info = new HBox(50); // horizontal box qui affiche
311         les infos des 2 joueurs
312         VBox score1 = new VBox(10); // vertical box qui
313         contient les infos du joueur 1
314         VBox score2 = new VBox(10); // vertical box qui
315         contient les infos du joueur 2
316         HBox allButton = new HBox(30); // horizontal box qui
317         contient les boutons
318
319         // changement graphique
320         endScreenRoot.setPadding(new Insets(20));
321         endScreenRoot.setAlignment(Pos.CENTER);
322         endScreenRoot.setStyle("-fx-background-color: BLACK");
323
324         info.setAlignment(Pos.CENTER);
325
326         allButton.setAlignment(Pos.CENTER);
327
328         res.setAlignment(Pos.CENTER);
329
330         Label scorej1 = new Label("Score : " + modele.map.pacmans.
331     get(0).points); // label recuperant le score final du joueur 1
332         scorej1.setTextFill(Color.GHOSTWHITE);
333         scorej1.setFont(new Font("Serif", 20));
334         Label pseudoj1 = new Label(Menu.pseudos.get(0));
335
336         // label recuperant le pseudo du joueur 1
337         pseudoj1.setTextFill(Color.GHOSTWHITE);
338         pseudoj1.setFont(new Font("Serif", 20));
339
340         score1.getChildren().addAll(pseudoj1, scorej1);
341
342         info.getChildren().add(score1);
343
344         Label resultat = new Label("Bien joué " + Menu.pseudos.get
345     (0) + " !");
346         resultat.setTextFill(Color.GHOSTWHITE);
347         resultat.setFont(new Font("Serif", 20));
348
349         // creation des labels relatifs aux informations du joueur 2
350         si le mode multijoueur est active
351         if (Menu.multijoueur == true) {
352             Label scorej2 = new Label("Score : " + modele.map.pacmans.
353     get(1).points);
354
355             scorej2.setTextFill(Color.GHOSTWHITE);
356             scorej2.setFont(new Font("Serif", 20));
357             Label pseudoj2 = new Label(Menu.pseudos.get(1));
358             pseudoj2.setTextFill(Color.GHOSTWHITE);
359             pseudoj2.setFont(new Font("Serif", 20));
360             score2.getChildren().addAll(pseudoj2, scorej2);
361             if (modele.map.pacmans.get(1).points > modele.map.pacmans.
362     get(0).points) {
363                 resultat.setText("Bien joué " + Menu.pseudos.get(1) +
364

```

```

        !");
    res.getChildren().addAll(pacman2View, resultat);
} else if(modele.map.pacmans.get(1).points == modele.map.
pacmans.get(0).points) {
    resultat.setText("Egalité !");
} else {
    res.getChildren().addAll(pacman1View, resultat);
}
info.getChildren().add(score2);
}

Button replayButton = new Button("Rejouer"); // bouton
permettant de rejouer
replayButton.setFont(new Font("Serif", 20));
replayButton.setOnAction(event -> { // relance le jeu avec
les memes parametres
modele = new Modele();
mode = new Mode(Menu.selected_bombe, Menu.selected_pouvoir
, Menu.selected_portail, Menu.multijoueur, modele.map);
fin = false;
Controleur controleur = new Controleur(modele, scene);
mode.modePortail();
mode.modePouvoir();
mode.modeBomberman();
mode.modeMultijoueur();
primaryStage.setScene(scene);
primaryStage.setTitle("Pac-Man");
primaryStage.setResizable(true);
primaryStage.show();
endStage.hide();
});

Button menuButton = new Button("Retour au menu"); // bouton
permettant de retourner dans le menu, NE PERMET PAS DE RELANCER DEPUIS LE MENU
menuButton.setFont(new Font("Serif", 20));
menuButton.setOnAction(event -> {
Platform.exit();
Menu.clipBgSound.stop();
Menu.main(null);
});

Button quitButton = new Button("Quitter"); // bouton
permettant d'arreter le programme
quitButton.setFont(new Font("Serif", 20));
quitButton.setOnAction(event -> { // quitte l'application
endStage.close();
System.exit(0);
});

allButton.getChildren().addAll(replayButton, menuButton,
quitButton); // ajout des boutons dans la hbox

endScreenRoot.getChildren().addAll(titreView, res, info,
allButton); // ajout de tous les elements necessaires a l'ecran de fin

// creation de la scene et affichage du stage
Scene end = new Scene(endScreenRoot, 1500, 1000);
endStage.setScene(end);
endStage.setTitle("Game Over !");
endStage.setResizable(true);

```

```

401             endStage.show();
402         }
403
404         lastUpdate = now;
405     }
406
407     }
408 }.start();

409 //ouverture d'une fenetre affichant la scene
410 primaryStage.setScene(scene);
411 primaryStage.setTitle("Pac-Man");
412 primaryStage.setResizable(true);
413 primaryStage.show();
414 }
415 }

416 private void update() {
417     // Mise à jour de l'interface, a completer si besoin
418     for (int k=0; k<modele.map.pacmans.size(); k++) {
419         modele.map.collisionFantome(modele.map.pacmans.get(k), modele.board); // permet de corriger des bugs de collision
420     }
421 }
422 }

423 // methode dessinant le canvas
424 private void draw(GraphicsContext gc) {
425
426     // Efface le canvas
427     gc.clearRect(0, 0, WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE);
428
429     // Fond d'écran du canvas
430     gc.setFill(Color.BLACK);
431     gc.fillRect(0, 0, WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE);
432
433     //dessiner les cercles
434     if (showDetectionRadius) {
435         gc.setStroke(Color.RED);
436         gc.setLineWidth(2);
437         for (Fantome fantome : modele.map.fantomes) {
438             int radiusPixels = fantome.DETECTION_RADIUS * TILE_SIZE;
439             int centerX = fantome.position.getX() * TILE_SIZE + TILE_SIZE / 2;
440             int centerY = fantome.position.getY() * TILE_SIZE + TILE_SIZE / 2;
441             gc.strokeOval(centerX - radiusPixels / 2, centerY - radiusPixels / 2, radiusPixels, radiusPixels);
442         }
443     }
444
445     // Dessine un mur
446     for (int i = 0; i < WIDTH; i++) {
447         for (int j = 0; j < HEIGHT; j++) {
448             if (modele.map.board[i][j] == '%') {
449                 gc.setFill(Color.PURPLE);
450                 gc.fillRect(i * TILE_SIZE, j * TILE_SIZE, TILE_SIZE, TILE_SIZE);
451             }
452         }
453     }
454
455     // Dessine un portail
456     for (int i = 0; i < WIDTH; i++) {
457         for (int j = 0; j < HEIGHT; j++) {

```

```

459         if (modele.map.board[i][j] == '/') {
460             gc.drawImage(portailImage, i * TILE_SIZE, j * TILE_SIZE,
461             TILE_SIZE, TILE_SIZE);
462         }
463     }
464
465     // Dessine un pouvoir
466     for (int i = 0; i < WIDTH; i++) {
467         for (int j = 0; j < HEIGHT; j++) {
468             if (modele.map.board[i][j] == '#') {
469                 gc.drawImage(pouvoirImage, i * TILE_SIZE, j * TILE_SIZE,
470                 TILE_SIZE, TILE_SIZE);
471             }
472         }
473     }
474
475     // Dessine une pacgomme
476     for (int i = 0; i < WIDTH; i++) {
477         for (int j = 0; j < HEIGHT; j++) {
478             if (modele.map.board[i][j] == '.') {
479                 gc.setFill(Color.YELLOW);
480                 gc.fillOval(i * TILE_SIZE + TILE_SIZE/3 , j * TILE_SIZE +
481                 TILE_SIZE/3, TILE_SIZE/3, TILE_SIZE/3);
482             }
483         }
484     }
485
486     // Dessine fantome
487     for (int i = 0; i < 4; i++) {
488         gc.drawImage(fantomeImage, modele.map.fantomes.get(i).position.getX() *
489         TILE_SIZE, modele.map.fantomes.get(i).position.getY() * TILE_SIZE, TILE_SIZE,
490         TILE_SIZE);
491     };
492
493
494     // Dessine PacMan1
495     gc.drawImage(pacman1Image, modele.map.pacmans.get(0).position.X * TILE_SIZE ,
496     modele.map.pacmans.get(0).position.Y * TILE_SIZE, TILE_SIZE, TILE_SIZE);
497
498     // Dessine PacMan2
499     if(Menu.multijoueur == true) {
500         gc.drawImage(pacman2Image, modele.map.pacmans.get(1).position.X *
501         TILE_SIZE, modele.map.pacmans.get(1).position.Y * TILE_SIZE, TILE_SIZE,
502         TILE_SIZE);
503     }
504
505     // Dessine une bombe posée
506     for (int i = 0; i < WIDTH; i++) {
507         for (int j = 0; j < HEIGHT; j++) {
508             if (modele.map.board[i][j] == '@') {
509                 gc.drawImage(bombePoseeImage, i * TILE_SIZE, j * TILE_SIZE,
510                 TILE_SIZE, TILE_SIZE);
511             }
512         }
513     }
514
515     // Dessine une bombe explosée
516     for (int i = 0; i < WIDTH; i++) {

```

```

510     for (int j = 0; j < HEIGHT; j++) {
511         if (modele.map.board[i][j] == '$') {
512             gc.setFill(Color.RED);
513             gc.fillRect(i * TILE_SIZE, j * TILE_SIZE, TILE_SIZE, TILE_SIZE);
514         }
515     }
516 }
517
518 // Dessine une bombe à récupérer
519 for (int i = 0; i < WIDTH; i++) {
520     for (int j = 0; j < HEIGHT; j++) {
521         if (modele.map.board[i][j] == '€') {
522             gc.drawImage(bombeRamasserImage, i * TILE_SIZE, j * TILE_SIZE,
523             TILE_SIZE, TILE_SIZE);
524         }
525     }
526 }
527
528 private void updateScore(Label label, int i) { //mise a jour étiquette du
529 score
530     label.setText("Score = " + modele.map.pacmans.get(i).points);
531 }
532
533 private void updateLife(Label label, int i) { //mise a jour etiquette nombre
534 de vies
535     label.setText("Vie(s) = " + modele.map.pacmans.get(i).life);
536 }
537
538 private void updateBombe(Label label, int i) { //mise a jour etiquette nombre
539 de bombes
540     label.setText("Bombes = " + modele.map.pacmans.get(i).nbrBombes);
541 }
542
543 private void updatePseudo(Label label, int i) { //mise a jour etiquette du
544 pseudo
545     label.setText(Menu.pseudos.get(i));
546 }
547
548 public static void updateAnnonce(Label label, int i) { // mise a jour etiquette
549 annoncant les pouvoirs en cours
550     if(modele.map.pacmans.get(i).pouvoir == null) {
551         label.setVisible(false);
552     } else {
553         label.setText("Pouvoir activé : " + Menu.pseudos.get(i) + " utilise " +
554 modele.map.pacmans.get(i).pouvoir);
555         label.setVisible(true);
556     }
557 }
558
559 public void setMode(boolean pouvoir, boolean portail, boolean bomberman, boolean
560 multijoueur){ // setter de mode
561     mode = new Mode(bomberman,pouvoir,portail,multijoueur, modele.map);
562 }
563
564 }
565
566 package Vue;
567
568 import java.awt.EventQueue;

```



```

59             e.printStackTrace();
60         }
61     });
62 }
63
64 /**
65 * Create the frame.
66 * @throws IOException
67 * @throws LineUnavailableException
68 * @throws UnsupportedAudioFileException
69 */
70 public Menu() throws IOException, UnsupportedAudioFileException,
LineUnavailableException {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(0, 0, 1300, 800);
71
72
73
74
75
76
77 /*
78 * Panel du premiere interface du menu .
79 */
80 commencerPanel = new JPanel();
81 commencerPanel.setBackground(new Color(204, 0, 204));
82 commencerPanel.setBorder(new LineBorder(Color.BLACK, 2));
83 setContentPane(commencerPanel);
84 commencerPanel.setLayout(null);
85
86 //Lancer la music au lancement du jeu
87 file1 = new File("./Pacman/src/Vue/Ressources/start_sound.wav");
88 AudioInputStream audioStream = AudioSystem.getAudioInputStream(file1);
89 clipBgSound = AudioSystem.getClip();
90 clipBgSound.open(audioStream);
91 clipBgSound.start();
92
93 //Charger le son de selection
94 file2 = new File("./Pacman/src/Vue/Ressources/hoversound.wav");
95
96 //Charger le son de lancement de jeu
97 file3 = new File("./Pacman/src/Vue/Ressources/commencersound.wav");
98
99
100 /* On va maintenant créer les différentes interfaces du menu :
101 *      interface1 = Ecran de d'accueil avec le titre du jeu
102 *      interfaceregle = Ecran qui décrit les différents modes
103 *      interface2 = Ecran qui permet la selection du mode solo ou multijoueur
104 *      interface3 = Ecran qui permet l'inscription des pseudos
105 *      interface4 = Selection des différents modes activables
106 */
107
108 //Label du la premiere interface
109 interface1Label = new JLabel("");
110 interface1Label.setBounds(0, 0, 1300, 800);
111 interface1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/
TOTO.png")));
112 commencerPanel.add(interface1Label);
113
114 //Label des règle du jeu
115 interfaceregleLabel = new JLabel("");
116 interfaceregleLabel.setBounds(4, -18, 1300, 800);

```

```

117     interfaceregleLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
118 /regleffff.png")));
119     commencerPanel.add(interfaceregleLabel);
120
121     //Label de la 2eme interface
122     interface2Label = new JLabel("");
123     interface2Label.setBounds(0, 0, 1300, 800);
124     interface2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/
125 TOTO.png")));
126     commencerPanel.add(interface2Label);
127
128     //Label de la 3eme interface
129     interface3Label = new JLabel("");
130     interface3Label.setBounds(0, 0, 1300, 800);
131     interface3Label.setIcon(new ImageIcon(getClass().getResource("Ressources/
132 TOTO.png")));
133     commencerPanel.add(interface3Label);
134
135     //Label de la 4eme interface
136     interface4Label = new JLabel("");
137     interface4Label.setBounds(0, 0, 1300, 800);
138     interface4Label.setIcon(new ImageIcon(getClass().getResource("Ressources/
139 TOTO.png")));
140     commencerPanel.add(interface4Label);
141
142     //Label du titre "PACMAN"
143
144     titreLabel = new JLabel("");
145     titreLabel.setBounds(400, 30, 490, 120);
146     titreLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/test0.
147 png")));
148     interface1Label.add(titreLabel);
149
150
151     //Label du bouton Jouer + ses event handlers
152     jouerLabel = new JLabel("");
153     jouerLabel.setBounds(500, 300, 283, 80);
154     jouerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/jouer.
155 png")));
156     interface1Label.add(jouerLabel);
157
158     jouerLabel.addMouseListener(new MouseAdapter() {
159         @Override
160         // Lorsque l'on clique sur le label "Jouer"
161         public void mouseClicked(MouseEvent e) {
162             try {
163                 playSound(file3 , clipCommencerSound);
164             } catch (LineUnavailableException | IOException |
165             UnsupportedAudioFileException e1) {
166                 // TODO Auto-generated catch block
167                 e1.printStackTrace();
168             }
169             // On passe à l'interface suivante.
170             repaint();
171             interface1Label.setVisible(false);
172             interfaceregleLabel.setVisible(false);
173             interface2Label.setVisible(true);
174             //Ajout des boutons exit et music sur l'interface suivante.
175             interface2Label.add(exitLabel);
176             interface2Label.add(musicLabel);

```

```

170     }
171     @Override
172     //Lorsque l'on place seulement le curseur sur le Label "Jouer", on
173     change la couleur et on met un bruitage
174     public void mouseEntered(MouseEvent e) {
175         joueurLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
jouer_hover.png")));
176         try {
177             playSound(file2 ,clipHoverSound);
178         } catch (LineUnavailableException | IOException |
179 UnsupportedAudioFileException e1) {
180             // TODO Auto-generated catch block
181             e1.printStackTrace();
182         }
183     }
184     @Override
185     //Lorsque l'on retire le curseur du Label "Jouer", on revient à la
186     couleur de base
187     public void mouseExited(MouseEvent e) {
188         joueurLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
jouer.png")));
189     }
190     });
191
192     //Label du bouton "Regles de jeu" + ses event handlers
193     reglesLabel = new JLabel("");
194     reglesLabel.setBounds(50, 600, 400, 49);
195     reglesLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/regles.
png")));
196     interface1Label.add(reglesLabel);
197
198     reglesLabel.addMouseListener(new MouseAdapter() {
199         @Override
200         public void mouseClicked(MouseEvent e) {
201             try {
202                 playSound(file3 , clipCommencerSound);
203             } catch (LineUnavailableException | IOException |
204 UnsupportedAudioFileException e1) {
205                 e1.printStackTrace();
206             }
207             // Passer à l'interface des règles lorsque l'on clique sur le label
208             "Règle"
209             repaint();
210             interface1Label.setVisible(false);
211             interfaceregleLabel.setVisible(true);
212         }
213
214         //Lorsque l'on place seulement le curseur sur le Label "Règle", on
215         change la couleur et on met un bruitage
216         public void mouseEntered(MouseEvent e) {
217             reglesLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
regles_hover.png")));
218             try {
219                 playSound(file2 ,clipHoverSound);
220             } catch (LineUnavailableException | IOException |
221 UnsupportedAudioFileException e1) {
222                 e1.printStackTrace();
223             }
224         }
225     }

```

```

219
220     @Override
221     //Lorsque l'on retire le curseur du Label "Règle", on revient à la
222     couleur de base
223     public void mouseExited(MouseEvent e) {
224         reglesLabel.setIcon(new ImageIcon(getClass().getResource("Ressources
225         /regles.png")));
226     }
227 );
228
229     //Label du bouton "Retour" + ses event handlers
230
231
232     retourLabel = new JLabel("");
233     retourLabel.setBounds(900, 550, 278, 204);
234     retourLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/RetourB
235     .png")));
236     interfaceregleLabel.add(retourLabel);
237
238     retourLabel.addMouseListener(new MouseAdapter() {
239         @Override
240         public void mouseClicked(MouseEvent e) {
241             try {
242                 playSound(file3 , clipCommencerSound);
243             } catch (LineUnavailableException | IOException |
244             UnsupportedAudioFileException e1) {
245                 e1.printStackTrace();
246             }
247             interfaceregleLabel.setVisible(false);
248             interface1Label.setVisible(true);
249
250         }
251         @Override
252         public void mouseEntered(MouseEvent e) {
253             retourLabel.setIcon(new ImageIcon(getClass().getResource("Ressources
254             /RetourR.png")));
255             try {
256                 playSound(file2 ,clipHoverSound);
257             } catch (LineUnavailableException | IOException |
258             UnsupportedAudioFileException e1) {
259                 e1.printStackTrace();
260             }
261         }
262         @Override
263         public void mouseExited(MouseEvent e) {
264             retourLabel.setIcon(new ImageIcon(getClass().getResource("Ressources
265             /RetourB.png")));
266
267         }
268     );
269
270     //Label du bouton "Exit" + ses event handlers
271     exitLabel = new JLabel("");
272     exitLabel.setBounds(1000, 600, 120, 49);
273     exitLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/exit.png
274     ))));
275     interface1Label.add(exitLabel); //On oublie pas d'ajouter le bouton "Exit" à
276     la première interface de démarrage.
277

```

```

270     exitLabel.addMouseListener(new MouseAdapter() {
271
272         @Override
273         //On ferme la fenêtre si on clique sur "Exit".
274         public void mouseClicked(MouseEvent e) {
275             System.exit(0);
276         }
277
278         @Override
279         public void mouseEntered(MouseEvent e) {
280             exitLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
281             Exit_hover.png")));
282             try {
283                 playSound(file2 ,clipHoverSound);
284             } catch (LineUnavailableException | IOException |
285             UnsupportedAudioFileException e1) {
286                 // TODO Auto-generated catch block
287                 e1.printStackTrace();
288             }
289         }
290
291         @Override
292         public void mouseExited(MouseEvent e) {
293             exitLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
294             exit.png")));
295         }
296     );
297
298
299     //Label du bouton "Musique" + ses event handlers.
300
301     musicLabel = new JLabel("");
302     musicLabel.setBounds(1200, 10, 60, 60);
303     musicLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/music_on
304     .png")));
305     interface1Label.add(musicLabel); //On oublie pas d'ajouter le bouton "Musique" à la première interface de démarrage.
306
307
308     musicLabel.addMouseListener(new MouseAdapter() {
309         @Override
310         public void mouseClicked(MouseEvent e) {
311             // Arrêter ou Lancer la musique en cliquant sur l'icône du son.
312             if(soundOn == true)
313             {
314                 musicLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/music_off.png")));
315                 clipBgSound.stop();
316                 soundOn = false;
317             }
318             else if(soundOn == false) {
319                 musicLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/music_on.png")));
320                 clipBgSound.start();
321                 soundOn = true;
322             }
323         }
324     );
325 }
```

```

323
324     /*
325      * Après avoir incrusté les icônes "Jouer", "Exit", "Musique" et "Règle" sur
326      * l'interface de démarrage,
327      * on s'occupe maintenant de la deuxième interface qui permet de préciser le
328      * nombre de joueurs.
329      */
330
331
332     //Label du bouton "1 joueur" + ajout à la deuxième interface + ses event
333     //handlers.
334
335     joueurs1Label = new JLabel("");
336     joueurs1Label.setBounds(350, 300, 200, 200);
337     joueurs1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/1jj.
338     png")));
339     interface2Label.add(joueurs1Label);
340
341
342     joueurs1Label.addMouseListener(new MouseAdapter() {
343
344         @Override
345         /* Si on clique sur "1Joueur", alors sa couleur change et "2Joueur" ré
346         cupère sa couleur de base
347         * s'il était déjà sélectionné.
348         */
349         public void mouseClicked(MouseEvent e) {
350             try {
351                 playSound(file3 , clipCommencerSound);
352             } catch (LineUnavailableException | IOException |
353             UnsupportedAudioFileException e1) {
354                 e1.printStackTrace();
355             }
356             //variables utiles pour selectionner plus tard le bon mode lors du
357             //lancement du jeu.
358             nbrJoueurs = 1;
359             selected1 = true;
360             selected2 = false;
361             //
362             joueurs1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/1jjh.png")));
363             joueurs2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/2jj.png")));
364             joueurs1Label.add(check1Label); //Ajout d'un GIF de sélection.
365
366         }
367
368         @Override
369         //La couleur change si on place le curseur sur le Bouton "1Joueur".
370         public void mouseEntered(MouseEvent e) {
371             joueurs1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/1jjh.png")));
372             try {
373                 playSound(file2 ,clipHoverSound);
374             } catch (LineUnavailableException | IOException |
375             UnsupportedAudioFileException e1) {
376                 // TODO Auto-generated catch block
377                 e1.printStackTrace();
378             }
379

```

```

372     }
373
374     @Override
375     //La couleur redevient normale si on enlève le curseur du bouton "1
376     Joueur".
377     public void mouseExited(MouseEvent e) {
378         if(selected1 == false)
379             joueurs1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/1jj.png")));
380     }
381
382 );
383
384
385
386     /* On fait exactement la même chose mais cette fois ci avec un mode "2
387     Joueurs".
388     * Voir les commentaire du mode "1Joueurs" ci-dessus en cas de questions*/
389
390     joueurs2Label = new JLabel("");
391     joueurs2Label.setBounds(750, 300, 200, 200);
392     joueurs2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/2jj.
393     png")));
394     interface2Label.add(joueurs2Label);
395
396     joueurs2Label.addMouseListener(new MouseAdapter() {
397
398         @Override
399         public void mouseClicked(MouseEvent e) {
400             try {
401                 playSound(file3 , clipCommencerSound);
402             } catch (LineUnavailableException | IOException |
403             UnsupportedAudioFileException e1) {
404                 e1.printStackTrace();
405             }
406             nbrJoueurs = 2;
407             selected2 = true;
408             selected1 = false;
409             joueurs2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/2jjh.
410             png")));
411             joueurs1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/1jj.
412             png")));
413             joueurs2Label.add(check1Label);
414         }
415         @Override
416         public void mouseEntered(MouseEvent e) {
417             joueurs2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/2jjh.
418             png")));
419             try {
420                 playSound(file2 ,clipHoverSound);
421             } catch (LineUnavailableException | IOException |
422             UnsupportedAudioFileException e1) {
423                 // TODO Auto-generated catch block
424                 e1.printStackTrace();
425             }
426         }
427         @Override
428         public void mouseExited(MouseEvent e) {

```

```

423         if(selected2 == false)
424             joueurs2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/2jj.png")));
425     }
426 }
427 );
428
429
430
431 //Création du titre "Nombre de Joueurs" et ajout à la deuxième interface de sélection.
432 choixNbrJoueursLabel = new JLabel("");
433 choixNbrJoueursLabel.setBounds(200, 150, 892, 87);
434 choixNbrJoueursLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/nbr_joueurs.png")));
435 interface2Label.add(choixNbrJoueursLabel);
436
437
438 /*Création du Bouton "Continuer" pour passer à l'interface suivante une fois le nombre de joueurs choisi,
439 * ajout à l'interface 2 de selection du nombre de joueurs ,
440 * et mise en place de ses event handlers
441 */
442 continueLabel = new JLabel("");
443 continueLabel.setBounds(500, 600, 250, 51);
444 continueLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/continue.png")));
445 interface2Label.add(continueLabel);
446
447 continueLabel.addMouseListener(new MouseAdapter() {
448     @Override
449     public void mouseClicked(MouseEvent e) {
450         try {
451             playSound(file3, clipCommencerSound);
452         } catch (LineUnavailableException | IOException |
453 UnsupportedAudioFileException e1) {
454             e1.printStackTrace();
455         }
456         if(selected1 || selected2) {
457             /*On passe à l'interface suivante si le nombre de joueurs a été choisi et
458             * que l'on clique sur continuer. On oublie pas de cacher les interfaces précédentes avant de passer à l'interface suivante
459             * et on ajoute les boutons "Exit" et "Music" à cette troisième interface
460             */
461             System.out.println("nbr de joueurs choisi :"+ nbrJoueurs);
462             repaint();
463             interface1Label.setVisible(false);
464             interface2Label.setVisible(false);
465             interface3Label.setVisible(true);
466             interface3Label.add(exitLabel);
467             interface3Label.add(musicLabel);
468
469             // Ajout des zones de texte pour insérer les pseudos des Joueurs
470             .
471             if(nbrJoueurs == 1) {
472                 interface3Label.add(joueurPseudoLabel);
473             }
474             else if(nbrJoueurs == 2) {

```

```

473                     interface3Label.add(joueurPseudo1Label);
474                     interface3Label.add(joueurPseudo2Label);
475                 }
476             }
477         }
478     }
479     @Override
480     public void mouseEntered(MouseEvent e) {
481         continueLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/continue_hover.png")));
482         try {
483             playSound(file2 ,clipHoverSound);
484         } catch (LineUnavailableException | IOException |
485 UnsupportedAudioFileException e1) {
486             e1.printStackTrace();
487         }
488     }
489     @Override
490     public void mouseExited(MouseEvent e) {
491         continueLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/continue.png")));
492     }
493 });
494
495 /**
496  * On est maintenant sur la troisième interface qui permet d'insérer les
497 pseudo.s
498  * , il faut créer tous ses boutons et ajouter leurs évènements.
499 */
500
501 //Titre de la troisième interface
502 pseudosLabel = new JLabel("");
503 pseudosLabel.setBounds(350, 150, 583, 87);
504 pseudosLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/pseudo.png")));
505 interface3Label.add(pseudosLabel);
506
507 /*Bouton qui va permettre de passer à la dernière interface de selection des
508 modes.
509  * Création du Label et de ses event handlers
510 */
511
512 continue1Label = new JLabel("");
513 continue1Label.setBounds(500, 600, 250, 51);
514 continue1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/continue.png")));
515 interface3Label.add(continue1Label);
516
517 continue1Label.addMouseListener(new MouseAdapter() {
518     @Override
519     //On passe à la dernière interface lorsque l'on clique sur continuer.
520     public void mouseClicked(MouseEvent e) {
521         try {
522             playSound(file3 , clipCommencerSound);
523         } catch (LineUnavailableException | IOException |
524 UnsupportedAudioFileException e1) {
525             // TODO Auto-generated catch block

```

```

525         e1.printStackTrace();
526     }
527     /*On cache les interfaces précédentes et on affiche la dernière
528      * interface. On lui ajoute les labels de bases.
529      */
530     interface1Label.setVisible(false);
531     interface2Label.setVisible(false);
532     interface3Label.setVisible(false);
533     interface4Label.setVisible(true);
534     interface4Label.add(exitLabel);
535     interface4Label.add(musicLabel);
536     interface4Label.add(commencerLabel);

537
538     pseudos = new ArrayList<String>();
539     /*Ajouter les pseudos entrées à la liste des pseudos
540      * On ajoute à la dernière interface, les différents modes. Les
541      modes ne sont pas les mêmes en
542      * fonction de si il y a un ou deux joueurs.
543      */
544     if(nbrJoueurs == 2) {
545         pseudos.add(joueur1PseudoTxt.getText());
546         pseudos.add(joueur2PseudoTxt.getText());
547         interface4Label.add(modeLabel);
548         interface4Label.add(bombermanLabel);
549         interface4Label.add(portailLabel);
550         interface4Label.add(pouvoirLabel);

551     }
552     else if(nbrJoueurs == 1) {
553         pseudos.add(joueurPseudoTxt.getText());
554         interface4Label.add(pouvoirLabel);
555         interface4Label.add(portailLabel);
556         interface4Label.add(modeLabel);

557
558     }
559 }

560

561 }

562 @Override
563 public void mouseEntered(MouseEvent e) {
564     continue1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/continue_hover.png")));
565     try {
566         playSound(file2 ,clipHoverSound);
567     } catch (LineUnavailableException | IOException |
568 UnsupportedAudioFileException e1) {
569         // TODO Auto-generated catch block
570         e1.printStackTrace();
571     }
572 }
573 @Override
574 public void mouseExited(MouseEvent e) {
575     continue1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/continue.png")));
576 }
577 }
578 */

/* Création des zones de texte du mode multijoueur*/

```

```

581 // Mode multijoueur : Joueur 1
582 joueurPseudo1Label = new JLabel("");
583 joueurPseudo1Label.setBounds(270, 300, 200, 50);
584
585 joueur1PseudoTxt = new JTextField();
586 joueur1PseudoTxt.setText("Joueur 1");
587 joueur1PseudoTxt.setFont(new Font("Arial", Font.BOLD, 14));
588 joueur1PseudoTxt.setBounds(0, 0, 200, 50);
589 joueurPseudo1Label.add(joueur1PseudoTxt);
590 joueur1PseudoTxt.setColumns(10);
591
592 // Mode multijoueur : Joueur 2
593 joueurPseudo2Label = new JLabel("");
594 joueurPseudo2Label.setBounds(770, 300, 200, 50);
595
596 joueur2PseudoTxt = new JTextField();
597 joueur2PseudoTxt.setText("Joueur 2");
598 joueur2PseudoTxt.setFont(new Font("Arial", Font.BOLD, 14));
599 joueur2PseudoTxt.setBounds(0, 0, 200, 50);
600 joueurPseudo2Label.add(joueur2PseudoTxt);
601 joueur2PseudoTxt.setColumns(10);
602
603 // Mode Solo : Joueur 0
604 joueurPseudoLabel = new JLabel("");
605 joueurPseudoLabel.setBounds(550, 300, 200, 50);
606
607 joueurPseudoTxt = new JTextField();
608 joueurPseudoTxt.setText("Joueur 1");
609 joueurPseudoTxt.setFont(new Font("Arial", Font.BOLD, 14));
610 joueurPseudoTxt.setBounds(0, 0, 200, 50);
611 joueurPseudoLabel.add(joueurPseudoTxt);
612 joueurPseudoTxt.setColumns(10);
613
614
615 /*On est maintenant dans la dernière interface qui permet la selection des
différents modes
   * On va créer les Labels des différents modes activables, c'est à dire :
   * Pouvoir, Portail, Bomberman en Multijoueur
   * Pouvoir, Portail en Solo
   */
616
617 modeLabel = new JLabel("");
618 modeLabel.setBounds(400, 25, 490, 120);
619 modeLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/mode0.png")));
620
621 pouvoirLabel = new JLabel("");
622 pouvoirLabel.setBounds(500, 250, 385, 75);
623 pouvoirLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/PouvoirBleu.png")));
624
625 portailLabel = new JLabel("");
626 portailLabel.setBounds(500, 350, 385, 75);
627 portailLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/PortailBleu.png")));
628
629 bombermanLabel = new JLabel("");
630 bombermanLabel.setBounds(500, 450, 385, 75);
631 bombermanLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/bombermanbleu.png")));
632

```

```

636
637     commencerLabel = new JLabel("");
638     commencerLabel.setBounds(350, 600, 544, 49);
639     commencerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/
commencer.png")));
640
641
642     //Event Handlers du mode Pouvoir
643
644
645     pouvoirLabel.addMouseListener(new MouseAdapter() {
646         @Override
647         public void mouseClicked(MouseEvent e) {
648             try {
649                 playSound(file3 , clipCommencerSound);
650             } catch (LineUnavailableException | IOException |
651             UnsupportedAudioFileException e1) {
652                 // TODO Auto-generated catch block
653                 e1.printStackTrace();
654             }
655             if (selected_pouvoir == false) {
656                 selected_pouvoir= true;
657                 pouvoirLabel.setIcon(new ImageIcon(getClass().getResource("
Ressources/PouvoirVert.png")));
658                 pouvoirLabel.add(check2Label);
659                 check2Label.setVisible(true);
660
661             }
662             else {
663                 selected_pouvoir=false;
664                 pouvoirLabel.setIcon(new ImageIcon(getClass().getResource("
Ressources/PouvoirBleu.png")));
665                 check2Label.setVisible(false);
666
667             }
668         }
669
670     }
671     @Override
672     public void mouseEntered(MouseEvent e) {
673         pouvoirLabel.setIcon(new ImageIcon(getClass().getResource("
Ressources/PouvoirVert.png")));
674         try {
675             playSound(file2 ,clipHoverSound);
676         } catch (LineUnavailableException | IOException |
677             UnsupportedAudioFileException e1) {
678                 // TODO Auto-generated catch block
679                 e1.printStackTrace();
680             }
681         }
682         @Override
683         public void mouseExited(MouseEvent e) {
684             if(selected_pouvoir == false)
685                 pouvoirLabel.setIcon(new ImageIcon(getClass().getResource("
Ressources/PouvoirBleu.png")));
686
687         }
688     });

```

```

689
690
691 //Event Handlers du mode Portail
692
693     portailLabel.addMouseListener(new MouseAdapter() {
694         @Override
695         public void mouseClicked(MouseEvent e) {
696             try {
697                 playSound(file3 , clipCommencerSound);
698             } catch (LineUnavailableException | IOException |
699             UnsupportedAudioFileException e1) {
700                 // TODO Auto-generated catch block
701                 e1.printStackTrace();
702             }
703             if (selected_portail == false) {
704                 selected_portail= true;
705                 portailLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/PortailVert.png")));
706                 portailLabel.add(check3Label);
707                 check3Label.setVisible(true);
708             }
709             else {
710                 selected_portail=false;
711                 portailLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/PortailBleu.png")));
712                 check3Label.setVisible(false);
713             }
714         }
715     }
716
717     @Override
718     public void mouseEntered(MouseEvent e) {
719         portailLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/PortailVert.png")));
720         try {
721             playSound(file2 ,clipHoverSound);
722         } catch (LineUnavailableException | IOException |
723             UnsupportedAudioFileException e1) {
724             // TODO Auto-generated catch block
725             e1.printStackTrace();
726         }
727     }
728     @Override
729     public void mouseExited(MouseEvent e) {
730         if(selected_portail == false) {
731             portailLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/PortailBleu.png")));
732         }
733     });
734
735 //Event Handlers du mode Bomberman
736
737     bombermanLabel.addMouseListener(new MouseAdapter() {
738         @Override
739         public void mouseClicked(MouseEvent e) {
740             try {
741                 playSound(file3 , clipCommencerSound);

```

```

743
744         } catch (LineUnavailableException | IOException |
745 UnsupportedAudioFileException e1) {
746             // TODO Auto-generated catch block
747             e1.printStackTrace();
748         }
749         if (selected_bombe == false) {
750             selected_bombe= true;
751             bombermanLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/bombermanvert.png")));
752             bombermanLabel.add(check4Label);
753             check4Label.setVisible(true);
754         }
755         else {
756             selected_bombe=false;
757             bombermanLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/bombermanbleu.png")));
758             check4Label.setVisible(false);
759         }
760     }
761
762     @Override
763     public void mouseEntered(MouseEvent e) {
764         bombermanLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/bombermanvert.png")));
765         try {
766             playSound(file2 ,clipHoverSound);
767         } catch (LineUnavailableException | IOException |
768 UnsupportedAudioFileException e1) {
769             // TODO Auto-generated catch block
770             e1.printStackTrace();
771         }
772     }
773     @Override
774     public void mouseExited(MouseEvent e) {
775         if(selected_bombe == false)
776             bombermanLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/bombermanbleu.png")));
777     }
778 }
779 );
780
781
782 //Event Handlers du bouton de lancement de la partie.
783
784 commencerLabel.addMouseListener(new MouseAdapter() {
785     @Override
786     public void mouseClicked(MouseEvent e) {
787         try {
788             playSound(file3 , clipCommencerSound);
789         } catch (LineUnavailableException | IOException |
790 UnsupportedAudioFileException e1) {
791             e1.printStackTrace();
792         }
793
794         //Affichage des modes choisis.
795         boolean[] modes= { selected1, selected2, selected_bombe ,
796         selected_portail, selected_pouvoir};

```

```

795         //selected1 et selected2 correspondent aux modes 1 et 2 joueurs.
796         for (int k=0; k<modes.length; k++) {
797             System.out.println(modes[k]);
798         }
799         Menu.this.dispose(); //Ferme la fenêtre de menu
800         System.out.println("Fermeture du menu !");
801         if(nbrJoueurs==1){
802             multijoueur = false;
803         } else {
804             multijoueur = true;
805         }
806
807         Game.launch(Game.class);
808
809     }
810
811     @Override
812     public void mouseEntered(MouseEvent e) {
813         commencerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/commencer_hover.png")));
814         try {
815             playSound(file2 ,clipHoverSound);
816         } catch (LineUnavailableException | IOException |
817 UnsupportedAudioFileException e1) {
818             // TODO Auto-generated catch block
819             e1.printStackTrace();
820         }
821     }
822     @Override
823     public void mouseExited(MouseEvent e) {
824         commencerLabel.setIcon(new ImageIcon(getClass().getResource("Ressources/commencer.png")));
825     }
826
827
828     // Le gif de la validation du mode solo ou multijoueur.
829
830     check1Label = new JLabel("");
831     check1Label.setBounds(120, 90, 120, 120);
832     check1Label.setIcon(new ImageIcon(getClass().getResource("Ressources/check.gif")));
833
834
835     //Confirmations des modes avec les gifs. ils sont indépendants.
836
837     check2Label = new JLabel("");
838     check2Label.setBounds(280, -30, 120, 120);
839     check2Label.setIcon(new ImageIcon(getClass().getResource("Ressources/check.gif")));
840
841     check3Label = new JLabel("");
842     check3Label.setBounds(280, -30, 120, 120);
843     check3Label.setIcon(new ImageIcon(getClass().getResource("Ressources/check.gif")));
844
845     check4Label = new JLabel("");
846     check4Label.setBounds(280, -30, 120, 120);
847     check4Label.setIcon(new ImageIcon(getClass().getResource("Ressources/check.gif")));

```

```

848
849
850     }
851
852     public void playSound(File file, Clip clip) throws LineUnavailableException,
853     IOException, UnsupportedAudioFileException {
854         AudioInputStream audioStream = AudioSystem.getAudioInputStream(file);
855         clip = AudioSystem.getClip();
856         clip.open(audioStream);
857         clip.start();
858     }
859 }
```

6.3.4 Tests Unitaires

```

1 package Test;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 import org.junit.jupiter.api.BeforeEach;
5 import org.junit.jupiter.api.Test;
6
7 import Modele.Map;
8 import Modele.Personnage;
9 import Modele.Position;
10
11 class PersonnageTestUnitaire {
12
13     private Personnage personnage;
14     @BeforeEach
15     public void setUp() {
16         personnage = new Personnage(2, 1); // Correspond à la position de 'P' dans la
17         carte
18     }
19
20     @Test
21     public void testGetPosition() {
22         Position position = personnage.getPosition();
23         assertEquals(2, position.getX());
24         assertEquals(1, position.getY());
25     }
26
27     @Test
28     public void testSetPosition() {
29         Position newPosition = new Position(1, 1);
30         personnage.setPosition(newPosition);
31         assertEquals(newPosition, personnage.getPosition());
32     }
33
34 }
```



```

1 package Test;
2 import Modele.Fantome;
3 import Modele.Position;
4 import static org.junit.jupiter.api.Assertions.*;
5
6 import org.junit.jupiter.api.BeforeEach;
```

```

7 import org.junit.jupiter.api.Test;
8
9 class TestUnitaireFantome {
10     private Fantome fantome;
11
12     @BeforeEach
13     void setUp() {
14         fantome=new Fantome(1,1);
15     }
16
17     @Test
18     public void testDistanceToPacman() {
19         Position pacmanPos=new Position(4,5);
20         int distance = fantome.distanceToPacman(pacmanPos);//(4-1)+(5-1)=7
21         assertEquals(7,distance,"Erreur La distance doit etre 7");
22     }
23     public void testFontomeBehind() {
24         Position pacmanPos=new Position(2,2);// Le vecteur du pacman au fontome dans le
25         // rayon
26         Position vecteurDirecteur=new Position(1,0);// Vecteur du mouvement du fantome
27         boolean pasDerriere=fantome.fantomeBehind(pacmanPos, vecteurDirecteur);// Dans
28         ce cas le pacman est devant le fantome
29         assertFalse(pasDerriere,"Le Pacman n'est pas derriere le fantome");
30     }
31 }
```

6.3.5 Tests Validations

```

1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4 import Modele.Pacman;
5 import Modele.Position;
6 import org.junit.jupiter.api.BeforeEach;
7 import org.junit.jupiter.api.Test;
8 import Modele.Fantome;
9 import Modele.Map;
10
11 class TestValidationFantome {
12     private Fantome fantome;
13     private Map map;
14     private Pacman pacman;
15
16     @BeforeEach
17     void setUp(){
18         String board = "%%%%%\n%P %\n% %\n% F %\n%%%"; // on donne un board carré pour faire le test
19         map = new Map(board);
20         fantome = map.fantomes.get(0); // on utilise un seul fantome
21         pacman = map.pacmans.get(0); // on utilise un seul pacman
22     }
23
24     @Test
25     void testDuFantomeAuPacman(){
26         Position pacmanPos = pacman.getPosition(); // determine la position du Pacman
```

```

27     int distanceInit = fantome.distanceToPacman(pacmanPos); // determine la
28     distance initial entre les deux
29
30     for (int i = 0; i < 10; i++) { // On bouge le fantome
31         fantome.moveRandomly(map);
32     }
33
34     int distanceFin = fantome.distanceToPacman(pacmanPos); // La distance final
35     entre les deux
36     assertTrue(distanceFin <= distanceInit, "Le fantome doit s'approcher du
pacman."); // Verification
37 }
38 }
```

```

1 package Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.BeforeEach;
6 import org.junit.jupiter.api.Test;
7
8 import Modele.Map;
9 import Modele.Personnage;
10 import Modele.Position;
11
12 class PersonnageTestValidation {
13
14     private Personnage personnage;
15     private Map map;
16
17     @BeforeEach
18     public void setUp() {
19         personnage = new Personnage(2, 1); // Correspond à la position de 'P' dans la
carte
20         String board = "%%%\\n% P%\\n% %\\n%%%";
21         map = new Map(board);
22         /* %%%\n
23          * % P%\n
24          * % %\n
25          * %%%%
26          */
27     }
28
29
30     @Test
31     public void testColisionMur() {
32         // La carte est déjà configurée correctement par le constructeur
33
34         // Appeler la méthode à tester
35         Position[] result = personnage.colisionMur(map);
36
37         // Vérifier que les positions possibles sont correctes
38         assertNull(result[0]); // Haut bloqué par un mur
39         assertNotNull(result[1]); // Bas accessible
40         assertNotNull(result[2]); // Gauche accessible
41         assertNull(result[3]); // Droite bloqué par un mur
42     }
43 }
44 }
```

6.3.6 Tests Intégrations

```
1 package Test;
2 import static org.junit.jupiter.api.Assertions.*;
3
4 import org.junit.jupiter.api.BeforeEach;
5 import org.junit.jupiter.api.Test;
6
7 import Modele.Fantome;
8 import Modele.Map;
9 import Modele.Position;
10
11 class TestIntegrationFantome {
12     private Fantome fantome;
13     private Map map;
14     @BeforeEach
15     void setUp(){
16         String board="%%%%%\n%P %\n% F%\n%%%%%";// La map est carré et chacun
17         des fantomes et pacman est dans un coté4
18         map=new Map(board);
19         fantome=map.fantomes.get(0);// On a un seul fantome
20     }
21
22     @Test
23     void testMoveRandomly() {// Le fantome ne doit pas rester fixe puisqu'il est dans
24         un carré ou il y a pas d'obstacle alors il doit se déplacer
25         Position Position0=fantome.getPosition();
26         fantome.moveRandomly(map);
27         Position newPos=fantome.getPosition();
28         assertNotEquals(Position0,newPos,"Le fantome ne doit pas rester fixe");
29     }
}
```