RAPPORT PROJET LONG
DEUXIEME ITERATION

LOUDYI Ayoub
LAANAIYA Mahmoud
LOTFI Mohamed Hamza
JAAFARI Amine
IKICH Mohamed
LAAOUINA Chouaib
JANAH Akram
Groupe IJ-5

Département Sciences du Numérique - Première année
2020-2021

1 Introduction

Ce document représente le rapport final du projet long réalisé par le groupe IJ-5. Il comprend un rappel du sujet abordé, des explications et des élaborations relatives aux différentes implantations et réalisations mises en oeuvre au cours du projet. Sans oublier bien sûr un petit aperçu sur l'organisation de l'équipe ainsi que les éléments élaborés suite aux séances gestion de projet.

2 Rappel

L'idée du projet est de créer un clone du jeu "Bomberman" en 2D. Ce jeu dans lequel un joueur se déplace et pose des bombes sur une ¨map¨ pour détruire des murs et vaincre des ennemis, qui à leurs tours vont ¨drop¨ des bonus pour le joueur. Les bombes peuvent également infliger des dégâts au joueur si il se trouve dans leur zone.
Le clone sera le plus fidèle **possible** au gameplay du vrai ¨Bomberman¨, avec une musique de fond (BGM).

3 Packages, Classes et Interfaces

Plusieurs packages ont été définis dans l'implantation :

3.1 Le package par défaut (default package)

Il contient la classe suivante :

Launcher.java : La classe principale qui lance la partie, elle contient la méthode main et met toutes les classes suivantes en lien.

3.2 Controller

C'est un package qui contient des classes qui relèvent du traitement des touches saisies par l'utilisateur et leur mise en exécution :

Controller.java : Interface qui contient les méthodes relatives aux touches pressées par l'utilisateur , ces méthodes sont **public boolean isUp()** le boolean de sortie est mis en vrai si l'utilisateur a pressé sur la flèche en haut, **public boolean isDown()**le boolean de sortie est mis en vrai si l'utilisateur a pressé sur la flèche en bas, **public boolean isRight()**le boolean de sortie est mis en vrai si l'utilisateur a pressé sur la flèche à droite et la méthode **public boolean isLeft()**le boolean de sortie est mis en vrai si l'utilisateur a pressé sur la flèche

à gauche.

Input.java : Implémente l'interface **KeyListener** des actions de l'utilisateur.

PlayerController.java : Classe qui implémente l'interface **Player.java** et renvoie les entrées clavier de l'utilisateur, selon sa saisie dans **Input.java** .

3.3 affichage

C'est le paquet regroupant les classes qui affichent le jeu sur l'écran :

Fenetre.java : La classe qui contient le canvas sur lequel on va dessiner , la fenêtre est creé avec une longueur et un largeur et on met le couleur de l'arrière-plan en noir, on ajoute le menu, le titre de cette fenêtre est **Bombermanv0** zt on ajoute à cette fenêtre un écouteur de clavier. La méthode **public void render(Game game, GameMap gameMap)** affiche cette fenêtre.

Renderer.java : La classe qui dessine l'état du jeu à chaque mise à jour.

msg.java : une classe implémente l'interface **WindowListener**.

JImagePanel.java : une classe hérite de la classe **Jpanel.java**

3.4 gameObjects

Ce paquet est un regroupement de tout ce que le joueur pourra intéragir avec, et il contient les classes :

GameObject.java : Classe abstraite des caractéristiques des objets du jeu créant un objet avec une taille de 32 pixels sur 32 pixels

MovingObject.java : Hérite de GameObject, elle est dédiée aux objets pouvant se déplacer sur la carte

Player.java : Hérite de la classre **MovingObject.java**, elle est dédiée au joueur principal manipulé par l'utilisateur.

Block.java : Interface spécialisant les blocs de la carte.

BreakableBlock.java : Implémente l'interface **Block.java**, dédiée aux blocs destructibles de la carte qui se casse si se trouve dans l'intervalle dans l'explosion du bombe et il ne va plus paraître avec la mise du type de tuile au **Null**.

UnbreakableBlock.java : Implémente l'interface **Block.java**, dédiée aux blocs non destructibles de la carte.

Bomb.java : une classe héritant de la classe **GameObject.java** implémentant une bombe qui a une portée d'explosion de trois cases dans les 4 sens. Le déroulement d'explosion est expliqué dans une méthode **public void expolde()** , cette classe contient aussi une méthode **public void update()** qui fait une mise à jour que si la bombe est posé. de plus on a donné à cette bombe une image par la méthode public Image getSprite() en ajoutant un "Timer" à cette bombe et on peut l'avoir par la méthode **public double getTimer()**.

Ennemi.java : une classe créant des ennemis et il hérite de la classe **MovingObject.java** , ces ennemis sont générés avec une position initiale et une vitesse qu'on veut qui est dans la classe **VariablesGlobales** qui se trouve dans le paquetage**game** et un mouvement prédeterminé , un ennemi fait un mouvement dans le sens des abcisses jusqu'à ce qu'il trouve un obstacle et il retourne en arrière , la chose qu'on a réalisé avec la méthode **public void appliquerEnnemi1(Gamemap map)** qui existe dans la classe **Position.java** dans le paquetage **Physique** on peut avoir l'image de ces ennemis avec la méthode **public Image getSprite()**

3.5 game

Ce paquet contient deux classes :

GameLoop.java : La classe responsable du déroulement de la partie, à chaque mise à jour.

Game.java : La classe qui configure le début de la partie. (Taille de la fenêtre, plus tard : le niveau du jeu, ...)

3.6 map

Ce paquet contient tout ce qui est en lien avec la carte du jeu :

GameMap.java : Classe de la carte du jeu.

Tile.java : Classe des cases/tuiles constituant la carte du jeu.

3.7 Menu

Ce paquet contient les classes relevant de l'interface utilisateur :

3.7.1 Menu de départ :

Le menu de départ est traité dans **Launcher.java** et dans **JImagePanel.java**. Il y a le boutton Jouer, Options et Quitter. Une fois qu'on clique sur Jouer, la fenêtre du menu de départ se ferme, et à sa place s'ouvre la fenêtre du jeu.

3.7.2 Menu à l'interieur du jeu :

Une fois le jeu lancé, on voit trois menus/boutons apparaître en haut de la fenêtre : Partie,Aide. Ils sont traités dans trois classes, et chaque classe contient une sous classe nommée **MenuActionListener** :

Partie.java : une classe dans laquelle il y a deux fonctionnalités, redémarrer le jeu et quitter le jeu que si on clique sur Partie qui est une barre dans le menu deux boutons nommés "Nouveu Jeu" et "Quitter" s'afficheront et on peut redémarrer le jeu avec un clic sur ce bouton ou clique sur Ctrl+N , la même chose pour quitter le jeu qu'on peut le faire soit en cliquant sur son bouton soit en cliquant sur Ctrl+A.

Aide.java : une classe dans laquelle il y a deux fonctionnalités, affichage du tutoriel et du contact que si on clique sur Aide qui est une barre dans le menu deux boutons nommés "Tutoriel" et "contact" s'afficheront et on peut avoir ce tutoriel avec un clic sur ce bouton ou clique sur Ctrl+T , la même chose pour le contact qu'on peut l'avoir soit en cliquant sur son bouton soit en cliquant sur

Ctrl+A.

Menu.java : une classe qui rassemble les deux barres constituant ce menu "Partie" et "Aide".

3.7.3 Informations à propos du joueur

Dans ce paquetage il y a aussi une autre classe qui est :
InfosJoueur.java est une classe qui affiche le temps qui reste dans le jeu qui est fixé au début du jeu à 200 et nous donne des informations sur le joueur en affichant sa durée de vie sous forme de pourcentage qui est fixé au début du jeu à 100%.

3.8 physique

C'est le paquet responsable de tout ce qui est mouvement :

Size.java : Classe définissant la taille d'un objet

VectorDE.java : Classe du vecteur déplacement élémentaire d'un objet

Position.java : Classe de la position cartésienne d'un objet

Mouvement.java : Classe du mouvement de l'objet. Toutes les entrées de l'utilisateur concernant le mouvement sont traitées ici.

3.9 graphique :

Ce package traite les côté visuels des objets de la carte et du joueur. Ce package contient les classes suivantes :

ImageUtils.java : Classe qui permet de charger une image depuis le lien d'un fichier présent dans les ressources du projet

SpriteSet.java : Cette classe définit un spriteSet qui est une collection d'images associés à des noms

SpriteLibrary.java : Cette classe qui définit une spriteLibrary qui est une collection de spriteSet associés à des noms de fichiers. Toutes les images et tous les noms de fichiers utilisés figurent dans le lien ressource du projet : "/res/sprites/units/perso"

AnimationManager.java : Classe qui gère la synchronisation des mouvements du joueur avec les sprites utilisés Elle associe donc à chaque action et à chaque direction du joueur une image donnée.

direction : Classe définissant la direction du joueur depuis son vecteur déplacement. Elle sera utiles pour la classe AnimationManager.

3.10 audio :

Ce package est dédié au traitement de la musique qu'on écoute lors de notre jeu et il contient trois classes :

AudioClip.java : est une classe abstraite dans laquelle on règle le volume de la musique et on l'arrête avec la méthode **public void cleanUp()**.

AudioPlayer.java : /résevé pour chouaib laaouina/.

MusicClip.java : /réservé pour Chouaib Laaouina/.

4 Organisation de l'équipe

La répartition suivante ne décrit pas rigoureusement la répartition de l'équipe. En effet chaque membre a contribué aux tâches des autres équipes. En gros, trois équipes sont constitués :

Sprites, Musique LOTFI Mohamed Hamza et LAAOUINA CHOUAIB

Moteur de Jeu LOUDYI AYOUB et MOHAMED IKICH

Menus, Touches de contrôle et différentes implémentations LAANAIYA MAHMOUD et JAAFARI AMINE

5 Difficultés rencontrées

La première difficulté classique qu'on avait rencontrée a été bien sûr la conception des classes et des interfaces à utiliser. Mais nous avions réussi finalement après plusieurs essais à implanter un diagramme UML.
Une deuxième difficulté résidait dans le déplacement et les collisions entre le joueur et les blocs. En effet la position du joueur était repérée en pixels, mais il fallait toujours faire attention à sa position relativement à l'indice (i,j) en entier des tuiles sur la carte. Plus précisement, au niveau des collisions nous avons cru au début que la position en pixels correspondait auc centre du carré représentant le Joueur, mais en réalité elle représentait le sommet en haut à gauche du carré. Prenant ceci en considération nous avions orienté les collisions vers un détecteur de dépassement des quatre sommets du carré, ainsi dès que l'un des sommets après un déplacement chevauche une structure à laquelle ils n'ont pas accès le déplacement est annulé.
Une autre diffculté trouvée était liée au téléchargement des Sprites des objets du jeu : Au début, à chaque fois que le jeu était mis à jour (à l'aide de la fonction Update), la classe **Tile** téléchargait à chaque fois les images du fichier ressource, et ceci dans une durée trop courte correspondant à la durée de mise à jour, ce qui créait un bug dans le jeu et le jeu devenait trop long. La solution établie a été de définir des images constantes au début du jeu, et la méthode **getSprite** qui affectait les Sprites correspondants aux objets correspondants ne faisait que faire appel à ces constantes.

6 Annexe :

6.1 Diagramme UML :

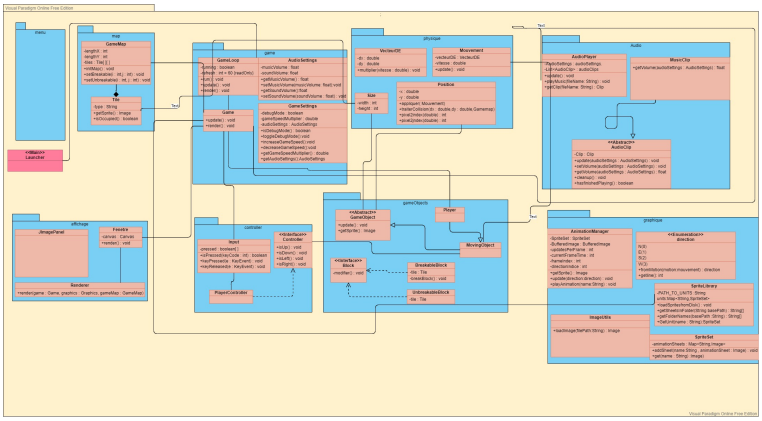

FIGURE 1 – Diagramme UML