DO Tran Minh Chau
DRION Léa
A2

Ce document présente les principales modifications architecturales apportées au projet POO 2025.

Tout d'abord, plutôt que de créer deux classes distinctes **DoorOpened** et **DoorClosed**, nous avons choisi de les factoriser dans une seule classe **Door**. Héritant de **Decor**, une **Door** peut être ouverte ou fermée, et indique si elle mène au niveau suivant ou précédent. Cette approche réduit la duplication de code et permet de gérer dynamiquement l'état de la porte ainsi que sa direction.

Nous avons ajouté directement dans la classe **GameLauncher** les méthodes de lecture et de décompression des cartes au lieu d'implémenter dans une autre classe comme **MapRepoFile** (ce que j'ai fait au début). Comme cette classe est déjà responsable du chargement initial du jeu, l'ajout de ces méthodes permet de centraliser tout le processus d'initialisation et la création d'une classe supplémentaire n'était pas nécessaire pour un projet de cette taille et aurait alourdi inutilement l'organisation du code.

Dans la classe **Game**, un système de gestion du changement de niveau a été créé pour permettre au jardinier de passer d'un niveau à l'autre via les portes. Parallèlement, l'affichage des portes a été amélioré en utilisant la méthode **updateImage()** dans la classe **Sprite** pour afficher dynamiquement une porte ouverte ou fermée selon son état.

En outre, la classe **GameEngine** a été adaptée pour assurer le changement de niveau avec **checkLevel()**, repositionner le jardinier et réinitialiser l'ensemble de la scène.

Afin de factoriser les classes **Wasp** et **Hornet** qui sont très similaires, une classe **Insect** a été créée. **Wasp** et **Hornet** en héritent donc. De même, **HornetNest** et **WaspNest** héritent de **Nest**.

Un attribut **lifePoints** a été ajouté dans la classe **Insect** afin de gérer la mort des insectes en cas de collision. Un **insecte** a **1 point** de vie par défaut. Les **frelons** modifient cet attribut à **2** lors de leur création.

Dans la classe **GameEngine**, **spawnInsecticide()** permet de faire apparaître des insecticides aléatoirement. Nous avons choisit de l'implémenter dans cette classe (et pas dans Insecticide par exemple) afin d'accéder facilement à la liste des décors (**game.world().getGrid().values()**)

Ensuite, **update()** permet de mettre à jour les différents composants de notre jeu : le jardinier, les insectes et les nids qui font apparaître des insectes autour du nid.

Afin de faciliter et factoriser la génération de position aléatoire, dans **spawnInsecticide()** et dans **Nest**, une méthode **randomPos(..)** a été ajoutée dans la classe Position.

La génération d'insectes et leur déplacement aléatoire fonctionnent. Ils peuvent se déplacer sur les fleurs mais pas sur les portes. Lorsqu'ils se déplacent sur une case où se trouve un insecticide, ils perdent 1 point de vie et l'insecticide est enlevé pour éviter qu'il y en ait trop sur la map.

La collision avec le jardinier fonctionne également, si il possède au moins le nombre d'insecticides nécessaires pour l'insecte qu'il rencontre alors ce dernier est tué sinon seul le jardinier perd des points de vie. La collision est gérée par la méthode **checkCollision()** dans la classe **GameEngine**.

Des classes **SpriteWasp** et **SpriteHornet** ont été ajoutées dans la package view afin de gérer l'affichage des insectes en fonction de leur direction, à la manière de **SpriteGardener**. Les classes sont très similaires mais n'ont pas été factorisées car cela n'aurait pas été très utile, leur code n'etant pas très long.