GOSSET Séverin – BIGUENET Denis

Manuel développeur

Modifications apportées depuis la bêta :

- les constructeurs ont été divisés en plusieurs méthodes pour qu'il n'y ait plus de calculs dedans.

- le fichier est lu ligne par ligne et non plus lettre par lettre.

- création d'une interface displayable pour dessiner les objets affichables.

- certains objets comme les détritus ou les bombes ne sont pas dans le tableau grid, mais une arrayList.

- Ajout de la gestion des explosions et des collisions avec Jbox2D.

- Suppression de getType, correction d'incrementTimeLeft

- un seul renderFrame avec calcul de threadSleep pour éviter le clignotement.

- le chemin affiché après A* est maintenant géré avec des objets (Path).

- gestion de victoire/défaite, et de changement de niveau.

- les touches de clavier disponibles en jeu sont stockées dans un HashMap.

Choix des structures :

Pour représenter les différents éléments (murs, détritus…) nous avons créé une classe abstraite DisplayableCell pour factoriser le code de ces éléments, ainsi qu'une interface Cell pour les manipuler.

Les cellules statiques sont stockées dans un tableau de Cell : on sait qu'ils ne bougeront pas, tandis que les cellules dynamiques sont stockées dans une ArrayList (pour les détritus) ou dans un HashMap (pour les bombes) pour pouvoir ajouter/supprimer plus facilement.

Le tableau ainsi que les listes sont stockés dans un Board, qui représente le niveau et son contenu.

On a aussi créé une interface Displayable qui permet d'afficher tout ce qui peut l'être, que ce soit les murs, le joueur, ou le chemin calculé par A*.

Classes et package:

fr.umlv.wallj.bomb : Contient les classes liées aux bombes du jeu.

Bomb : représente une bombe avec ses coordonnées, un temps avant explosion, un body, et une explosion.

Permet de gérer toutes les opérations liées aux bombes, comme leur temps avant explosion ou l'explosion. Bomb implémente l'interface Displayable, car une bombe peut-être affichée.

GOSSET Séverin – BIGUENET Denis

Explosion : représente une explosion par sa position, le nombre de rayons, et une arrayList de rayons. La méthode reportFixture permet de donner une impulsion aux détritus selon l'angle et la distance à laquelle ils sont de l'explosion. Contient aussi une méthode pour afficher l'explosion générée.

Explosion implémente l'interface Displayable et aussi RayCastCallBack pour la méthode fixtureReport.

fr.umlv.wallj.display : Contiens les classes utilisée pour l'affichage du jeu.

Displayable : interface pour appliquer une méthode de dessin sur un objet.

Displayer : contient des méthodes pour dessiner l'interface, attendre des événements, afficher le chemin pris par WallJ…

Un displayer contient un board (celui du niveau actuel), un ApplicationContext et aussi les coordonnées du dernier clic effectué par le joueur.

fr.umlv.wallj.game: Contiens les classes principales du jeu.

Board : représente le niveau par la position du joueur, un world, une longueur, une largeur, un tableau de cellules, une liste de bombes, une liste de détritus ainsi qu'un nombre de bombes restantes. Cette classe implémente l'interface Displayable pour afficher son contenu.

Gère la validité d'un niveau, la construction du tableau du niveau, ainsi que certaines actions du joueur comme poser une bombe ou leur activation.

Cell : interface qui gère l'initialisation d'un DisplaybaleCell selon les arguments. Contient aussi certaines constantes pour différencier les différentes cellules (wall, trash etc…) et permet d'appliquer certaines méthodes, comme vérifier s'il s'agit d'une case vide.

CollisionHandler : méthodes pour gérer la collision entre deux body. Implémente l'interface Jbox2D ContactListener.

DisplayableCell : classe abstraite pour factoriser le code des murs, détritus…

Un DisplayableCell est représenté par ses coordonnées, un type (mur, détritus…) et un body.

Empty : représente une case vide dans le niveau, une case vide n'a pas de body. Empty hérite de DisplayableCell.

GOSSET Séverin – BIGUENET Denis

Game : représente la partie par un Board, un numéro de niveau, un Displayer, une liste de touches de clavier interagibles, un indicateur de victoire (un booléen) et aussi par certaines action du joueur comme un nœud de départ et arrivée ainsi que le chemin entre ces deux nœuds. Game permet de bouger son personnage, de mettre à jour la partie et sa physique, indique aussi si le joueur a gagné la partie.

Garbage : représente un détritus, le détritus est dynamique. Garbage hérite de DisplayableCell.

Player : représente l'état du joueur (sa position), de l'afficher et de lui donner une novelle position. Implémente Displayable.

Sizes : classe contenant diverses constantes, comme l'espace entre le bord de la fenêtre et le niveau, la largeur d'une case ou la taille de la police.

Trashcan : représente une poubelle, une poubelle est statique. Hérite de DisplayableCell.

Wall : représente un mur, un mur est statique. Hérite de DisplayableCell.

fr.umlv.wallj.pathfinding : Contiens les classes liées au pathfinding et à l'algorithme A*

Node : gestion de l'algorithme A*, un nœud est représenté par ses coordonnées, un cout, un coup théorique jusqu'à l'arrivée, ainsi qu'un précédent (la case visitée juste avant).

Contient différentes méthodes pour trouver le chemin le plus court d'un point A vers un point B, éviter les obstacles, trouver le nœud le plus proche…

Path : gestion de l'affichage du meilleur chemin trouvé par la méthode Node.shortestWay. Un chemin est représenté par une arrayList de nœuds et un index. Path implémente l'interface Displayable.