Délivrable 3.2

Rapport de test et de développement

Auteur(s) : Maxime T., Nathan P. v 1.4
Group 2.4

Sommaire:

I. Synthèse de développement

1. Comparaison objectifs et résultat final

Premièrement, nous avons pu réaliser tout ce que nous voulions faire au départ, c'est à dire un jeu Rogue-Like où nous allons de salle en salle pour tuer des monstres de différentes sortes.

Secondement, nous avons pu ajouter des choses qui n'étaient pas prévues initialement, étant donné que certains développeurs étaient inspirés, et avaient du temps devant eux. De ce fait, nous avons mis plus d'objets que prévu, créé un système d'inventaire pour stocker des potions ou des clés. Par la suite, nous avons aussi ajouté des décorations aux salles, ou encore créé deux systèmes d'attaques : Une lourde et une rapide afin de diversifier un peu le gameplay.

Enfin, nous avons pu créer un but au jeu avec un scénario, qui est de récupérer des clés au fur et à mesure que nous parcourons des salles, afin de pouvoir déverrouiller la salle du boss et le combattre. Tuer ce boss est alors le but final de notre jeu.

2. Aide à l'utilisation de l'application

Afin de pouvoir lancer le jeu, vous avez juste à cliquer sur le dossier du jeu dans votre package explorer sur eclipse, puis cliquer sur l'icône "Run" de Eclipse.

Mais vous avez aussi la possibilité d'utiliser le Jar fourni, et simplement double cliquer dessus pour le lancer.

Nous vous conseillons, une fois votre jeu lancé, d'aller dans la partie help afin de consulter toutes les touches pour jouer confortablement.

II. Description des tests unitaires réalisés

1. Room

On crée deux salles avec des portes définis, et on test leurs connexions entre elles afin de vérifier que les connexions entre les salles soient bien fonctionnelles lors de la génération du monde.
Pour tester cette partie du code, nous avons utilisé le constructeur de room qui prend une configuration sous forme d'une chaîne de caractère. Chaîne de quatre caractères qui sont soit des zéros soit des uns (ex : "0101", 0 équivaut à un mur et 1 à une porte selon la direction). on utilise ensuite la méthode `isOpen(Direction)` qui vérifie que la porte existe dans la direction donnée.
Évidemment nous aurons préalablement crée des salles qui ont des positions adjacentes.

Voici donc le code du test :

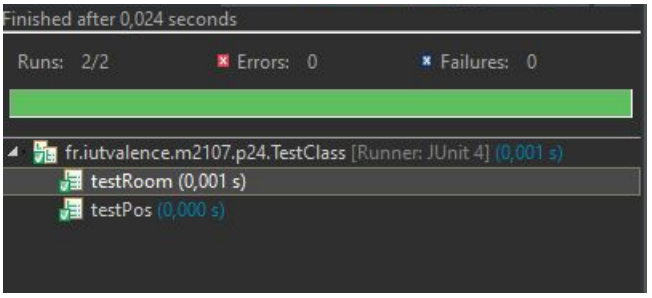
```
public void testRoom() {
    Room room1 = new Room(new Position(100, 200), "0110");
    Room room2 = new Room(new Position(101, 200), "0001");
    room1.addDoors(room2);
    assertEquals("Les portes sont reliées", room1.isOpen(Direction.RIGHT), room2.isOpen(Direction.LEFT));
}
```

2. Position

On crée une position, on la modifie, et on teste si elle est bien modifiée pour vérifier que le joueur puisse se déplacer correctement.
Pour tester cela, on a créé une position puis on l'a modifié via la méthode `move(x,y)` puis on a vérifié que la position était égale à celle que l'on vient de modifier.
Voici donc le code de ce test :

```
public void testPos() {
    Position pos1 = new Position(0,1);
    pos1.move(2, 0);
    assertEquals("Position égale", pos1, new Position(2,1));
}
```

3. Résultats



Les tests sont positifs. Ainsi nous pouvons dire que nos deux classes sont bien fonctionnelles. Nous avons choisi de tester ces deux classes, car nous pensons que le fait de pouvoir changer de salle et de pouvoir se déplacer sont des choses primordiales dans notre jeu.