Tests Included in This Build

# RoomUtilsTest.java

## checkRoomExists()

Checks if a room exists in particular maze coordinates. The room that is tested is the room with coordinates (5, 6) left of the start room. Prior to inserting this left room, the hasRoom() method is expected to return false for the left room’s coordinates. After inserting this room to the left of the start room, the method hasRoom() is expected to return true at the left room’s coordinates. This method ensures that the maze correctly stores rooms in the correct coordinates and that the hasRoom() method returns true if a room exists at a some coordinates in the maze.

## checkDoorGenerationRequirements()

This test tries to find out the doors that are required/forbidden when generating a new room that already has neighbors. The method tries to generate a new room to the top right of the start room with the following maze layout:



The red room is expected to require a bottom door and forbid a left door from spawning. The top and right doors are optional. This test was written to ensure that the correct doors will be generated in the correct spaces and that the player won’t generate a room that has a door that a player can enter but not exit

## checkEdges()

This test iterates through both axis in our 2d room map and does boundary checking to ensure that the proper return value (null) is given for out of bounds get room calls. This is very important because it helps validate the helper functions that this test are calling, which are used in the main room generation algorithm. They keep the map from spilling out past the total size of the grid and prevents calls to rooms that don’t exist.

## testHasLeft(), testHasRight(), testHasTop(), testHasBottom()

These tests check that the expected orientation is followed in the Room[][] maze object that keeps track of the maze and rooms within it. Mapping from Cartesian coordinates to a double-array/matrix structure must follow a couple of rules that might seem counterintuitive at first (to move left, for example, you must subtract 1 from the columns, and to move up you must subtract 1 from the rows), and as such these tests stand as a check that the logic is not getting flipped or changed in this regard.

## testFinalRooms()

This test checks that if rooms are generated on the edge of the Room[][] maze that they are final rooms, and the correct directional final room. This test uses a copy of the RoomComponent.generateNewRoomIfNeeded, that is adjusted to work in the test file.

## goThroughDoorLocation()

This test simulates a character going through a door similar to the RoomComponent method, except it only does the location part of it. It then checks that the new location is correct. This is to make sure that as the player moves

# GameScreenTest.java

## checkGameOver()

This test asserts the game end screen, which occurs when the player dies or meets endgame conditions without beating the boss. It loads into memory the GameOverSubScene. This subscene handles the text, background, and buttons of the game over scene. A simulated click of the Quit game button closes the JavaFX game and an if statement checks active memory to see whether the GameWorld ID for “handle” is still equal to the new active top level GameWorldID. In effect, this test is able to simulate the continued functionality of a player losing the game then closing out of the JavaFX game.

# RoomTest.java

## testListDifferenceAndEquality()

Room generation in our maze is heavily supported by configurations in the Room class that map the levels we have designed to certain characteristics a room might have (a left door, right door, etc.). It is important that these methods get returned properly, and also that there is not perfect overlap between the lists. This test checks that there is a difference between a set of lists, which consequently helps us guarantee that we can meet all scenarios for room generation.

## 