M4 Unit Test Write Up

PlayerTest.java

(@author: HR)

This class currently has 2 unit tests, and is essentially used to test whether or not the functionality of the “open/closed door exits” works based on the presence of monsters in the room.

testExitsClosed()

- This method is used to check and see if, when the monsters are present in a room, the exits will actually have a collideable property known as “solid.” This means that if there are monsters in the room and the player walks up to the doors, the doors should act as a solid piece of wall.

testExitsOpen()

- This method performs the exact opposite of the earlier method, and checks to see if the exits do not have the solid property associated with them when the monsters are removed from the room environment.

Both of these tests are done by generating a room in the same manner that a room would be generated in the game when walking through a door, and then checking the tile properties before and after removing the list of monsters from the room.

MonsterTest.java

(@author: NL)

atLeastOneMonster()

- This class tests if there is any monster in the non-starting room.
- The unit test does so by selecting the first exit within the starting room’s list of exits. Then once the player enters the room, the test checks that the monsters in the room are greater than 0. the starting room.

monsterMove()

- This class test if monster moves.
- The unit test does so by getting the initial position of the monsters. Waiting for a few seconds, then checking the current position of the monsters. Finally, it verifies that the initial and current positions are different.

EnemyTest.java

(@author: BM)

The unit tests verified that slimes attack the player when the player is close by, and do not when the player is out of range. This ensures that non-projectile attack functionality works.

testSlimeInRange()

- My tests puts the player in range of the enemy, ensuring the player's health decreases when the enemy attacks.

testSlimeOutOfRange()

- In this test, the player is moved out of range, and the enemy attempts to attack (and fails). Thus, the player's health stays the same.

testPlayerShoot.java

(@author: LS)

testShootProjectile()

- This tests that the player is able to shoot, and the bullets are appearing on the screen according to when the player shoots.
- The unit test verifies this by hitting Key.ENTER and checking that the bullets are objects in the room.

testLoseInGame()

- This test that the player can shoot at the monster and inflict damage.
- The unit test verifies that when the player is shooting, the monster's health decreases or causes them to die. It does so by artificially setting the player to have unlimited health so that it will not die before the monsters. Then the player turns in a full circle shooting bullets. The test then checks that damage is inflicted on the monsters.

LoseScreenTest.java

testLoseInGame()

- This test checks that the player's health will decrease when the monsters shoot at it. Once the player's health is zero, the game will stop.
- The unit test moves the player from the starting room to an adjacent room with monsters. Then the monsters are then moved to be directly next to the player, and they shoot at the player. The test then checks that the losing screen has the reset and exit button.

testLoseScreenRestart()

- This test checks that once the player's health is zero, and the losing screen is reached, the restart button can be clicked.
- The unit test sets the player's health to zero, then clicks on the "restart" button.