Megan Weller
ESOF 322 – A3
Tests

AllTests

Door

---

testEnterDoor() - Tests Door enter()

For player to enter the door, they must have the key; it will print that the player does not have the key and stay in the current room. If they do have a key, it will print that they entered the door, enter the new room and set the new location. This test will test the 4 cases when the does/does not have the key and if they are entering Room1 and Room2. The initial method enters doors and checks if the Player has a key.

Initial State:

- One instance of Player
- Two instances of Room
- One instance of Key
- One instance Door

For Test 1 and 2, the Player is in Room1 which contains a key that the Player can pick up. A Door connects Room1 and Room2. For Test 3 and 4, the Player is in Room2 which contains a key that the Player can pick up.

Input Data Used:

The method Door enter() uses:

- Player haveItem()
- Player setDesc()
- Player getLoc()
- Cavesite enter()

The test case uses:

- Room setSide()
- Player set Room()
- Room addItem()
- Player pickup()
- Player drop()
- Player getLoc()

Output Cases:

Test 1: The player is in Room1 without a key trying to enter the Door into Room2

I know that the Player will not be able to enter the Door so the Room expectedRoom is Room1. The player attempts to enter the door and the test gets the location of the player. It tests:

assertTrue(actualRoom.equals(expectedRoom)).

It checks whether the location of the player is the expected location, which in this case came out true.

Test 2: The player is in Room1 with a key trying to enter the Door into Room2

Expected Room: Room2
Actual Room: Room2
assertTrue(actualRoom.equals(expectedRoom)) which is true because the location of the Player is the expected Room because the Player has a Key

Test 3: The player is in Room2 with a key trying to enter the Door into Room1

Expected Room: Room1
Actual Room: Room1
assertTrue(actualRoom.equals(expectedRoom)) which is true because the location of the Player is the expected Room because the Player has a Key

Test 4: The player is in Room2 without a key trying to enter the Door into Room1

Expected Room: Room2
Actual Room: Room2
assertTrue(actualRoom.equals(expectedRoom)) which is true because the location of the Player is the expected Room because the Player does not have a Key

Player

---

testGoPlayer() – Player go()

The method moves the Player into the new location based on the current room and the direction desired. This test has one test that will test the coverage. The initial method helps the Player move in the direction desired.

Initial State:

- One instance of Player
- Two instances of Room

The Player is in Room1 and there is a Room2 is in the direction 4, or down.

Input Data Used:

The method Player go() uses:
- Room exit()

The test case uses:
- Room setLoc()
- Room exit()
- Room setSide()

Output Cases:

Test 1: The Player is in Room1 and is going down in the direction of Room2

I set a value of expectedRoom to be Room2 since I know that it is down from Room1. I used the method Player go() in the direction 4 and assigned the room the Player is to be actualRoom. To see if the actualRoom is the expectedRoom I used:

assertTrue(actualRoom.equals(expectedRoom)

It asserts that the statement is true, otherwise it will throw an error. The statement came out to be true, therefore the test successfully ran.

testPickUpItem() – Player pickUp()

The method picks up items and places it in the Player's inventory. There are two test cases for total coverage.

Initial State:
- One instance of Player
- One instance of Room
- Three instances of Keys

The Player is in a Room and there are three Keys that the Player can pick up and place in their inventory.

Input Data Used:

The method Player pickUp() uses:
- Room removeItem()
- Array myThings[]

The test cases uses:
- Player setLoc()
- Room addItem()
- Player numItemsCarried()

Test 1: The Player picks up Key1

The numExpected1 is 1 because we are expecting that only one item will be in the inventory. The Player pickUp() Key1 and I test:

assertEquals(thePlayer.numItemsCarried() == numExpected1)

to see if the actual number of items in the inventory is equal to the expected number. It tests to be true, so that part of the coverage is complete.

Test 2: The Player attempts to pick up Key2 and Key3 with Key1 being in their inventory

I know that the inventory can only two items so I covered if the inventory already had one item. I know the numExpected2 of the inventory should be 2, therefore when the Player picks up Key2 and then Key3, I tested:

assertEquals(thePlayer.numItemsCarried() == numExpected2)

to see if the method did not accept Key3 in the inventory. It asserts the two objects are equal, otherwise it will throw an error. The statement turned out to be true, therefore the method has completed its coverage.

testDropItem() – Player drop()

The method drops the items in the Player's inventory and places it back into the Room. There are 4 tests for the method to test all coverage possible.

Initial State:
- One instance of Player
- One instance of Room
- Two instances of Key

The Player picks up Key1 and Key2 from Room and then drops the Items.

Input Data Used:

The method Player drop() uses:
- Room addItem()
- Array myThings[]

The test cases uses:
- Room addItem()
- Player setLoc()
- Player pickUp()

When the Player drops() the item, it goes through the if-statement

if (itemNum > 0 & itemNum <= itemCount)

therefore, I tested dropping item 3, 2, 1, 0 to cover all cases within the if- statement

Test 1: The Player has two items in the inventory and drops the second item

When the Player drops the second item, the number of carried items should be 1, numExpectedTest1, since the limit carried is two. I tested with the statement:

assertEquals(thePlayer.numItemsCarried(), numExpectedTest1);

to test whether the two are equal, which turned out to be true. It asserts that the statement is true, otherwise it will throw an error.

Test 2: The Player attempts to drop item '3' from the inventory

To fully cover the if-statement, I had the player drop the third item from the inventory. It won't pass the if – statement so the number of items carried should still be 1, numExpectedTest1. I tested with the statement:

assertEquals(thePlayer.numItemsCarried(), numExpectedTest1);

to test whether the two are equal, which turned out to be true. It asserts that the statement is true, otherwise it will throw an error.

Test 3: The Player attempts to drop the 'zeroth' item from the inventory

To fully cover the if-statement, I had the player drop the 'zeroth' item from the inventory. It won't pass the if – statement so the number of items carried should still be 1, numExpectedTest1. I tested with the statement:

assertEquals(thePlayer.numItemsCarried(), numExpectedTest1);

to test whether the two are equal, which turned out to be true. It asserts that the statement is true, otherwise it will throw an error.

Test 4: The Player has one item in the inventory and drops the item

When the Player drops the item, the number of carried items should be 0, numExpectedTest0, since there is one item in the inventory. I tested with the statement:

assertEquals(thePlayer.numItemsCarried(), numExpectedTest0);

to test whether the two are equal, which turned out to be true. It asserts that the statement is true, otherwise it will throw an error.

testAddItem() – Room addItem()

The method adds an item into the Room. There is one test case for this method.

Initial State:
- One instance of Room
- One instance of Key

Input Data Used:

The method Room addItem() uses:
- Array contents[]

The test case uses:
- Room getRoomContents()
- Array Item actual

Test 1: Add the item Key to the Room

To test whether the key added is in the room contents, I assigned a key expected Key to the Key being added. Once the key is added to the room, I assigned the first spot of the array actual[] to the item in getRoomContents(). I then used the test statement:

assertEquals(actual[0], expectedKey);

to see if the key in the room is the expectedKey. It is checking whether the item in the array equals the key we added. It turned out to be true, therefore it passed.

testRemoveItem() – Room removeItem()

The method removes items from the room. There is one test case to test all coverage.

Initial State:
- One instance of Room
- One instance of Key

The key is currently added to the Room.

Input Data Used:

The method Room removeItem() uses:
- Array contents()

The test case uses:
- Room addItem()
- Room roomEmpty()

Test 1: Remove item Key from the Room

To test if the key is fully removed from the Room, I first added the key the room. I then made the Boolean expectedEmpty to be true because I know once the key is removed, it should be empty. I made a second Boolean empty and assigned it to r1.roomEmpty() to see what the condition is. I tested with the statement:

assertTrue(empty == expectedEmpty);

to see if it is indeed empty. It asserts that the statement is true, otherwise it will throw an error. The two Booleans are true therefore the test passed and the coverage is complete.

testEnterRoom() – Room enter()

The player enters a Room and sets the new location. There is one test case to complete full coverage.

Initial State:

- One instance of Player
- Two instances of Room

The Player is in Room1 and there is a room below it, Room2.

Input Data Used:

The method Room enter() uses:

- Player setLoc()

The test case uses:

- Room setSide()
- Room setRoom()
- Player getLoc()

Test 1: The Player is in Room1 and is entering Room2

I set a Room expectedRoom to be Room2 since that is the room I know the Player should be in once they entered. Once the Player enters the room, the Room actualRoom is assigned the Player's current location. I tested with the statement:

assertTrue(actualRoom.equals(expectedRoom));

to check for correctness. It asserts that the statement is true, otherwise it will throw an error. The test is true therefore the test succeeds.

testExitRoom() – Room exit()

The player exits one room into the direction of another Room. There is one test case to complete all coverage.

Initial State:

- One instance of Player
- Two instances of Room

The Player is set in Room1 and there is a room below it, Room2.

Test 1: The Player is in Room1 and is exiting to Room2

I set a Room expectedRoom to be Room2 since I know that's where the Player should be when they exit Room1. Once the Player exits the room using the directional value of 4, Room actualRoom is assigned to the Player's location. I tested with the statement:

assertTrue(actualRoom.equals(expectedRoom));

to check for correctness. It asserts that the statement is true, otherwise it will throw an error. The test is true; therefore the coverage is complete.