Author of the Component: Kanishka Garg

Inspection Team Members: Bradley Robinson, Basil Issac, Shubham Jain

Date: 30th November 2013

UNIT TESTING

Component/method/function signature: public void **movementPlayerVillian**(int keyCode)

I/P for the method: A keyboard key command; w = wall; e = water (obstacles)

O/P: Move made by the Villain and Player character in accordance to the keyboard key command and obstacle as a constraint.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*code snippet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void **movementPlayerVillian** (int keyCode)

{

//Handling moving player with controls UP, DOWN, LEFT, RIGHT

if (keyCode == KeyEvent.VK\_UP) {

//if you touch a "w" or wall, the move does not occur

if(!m.getMap(p.getTileX(), p.getTileY() - 1).equals("w") && !m.getMap(p.getTileX(), p.getTileY() - 1).equals("e") ){

p.move(0, -1); //Player Moves

VM\_AI(v); //Mummy Moves

}

}

//if you press down, move down

if(keyCode == KeyEvent.VK\_DOWN){

if(!m.getMap(p.getTileX(), p.getTileY()+1).equals("w") && !m.getMap(p.getTileX(), p.getTileY()+1).equals("e")){

p.move(0, 1); //Player Moves

VM1(v); //Mummy Moves

}

}

//if you press left

if(keyCode == KeyEvent.VK\_LEFT){

if(!m.getMap(p.getTileX()-1, p.getTileY() ).equals("w") && !m.getMap(p.getTileX()-1, p.getTileY() ).equals("e")){

p.move(-1, 0); //Player Moves

VM1(v); //Mummy Moves

}

}

//if you press right

if(keyCode == KeyEvent.VK\_RIGHT){

if(!m.getMap(p.getTileX()+1, p.getTileY() ).equals("w") && !m.getMap(p.getTileX()+1, p.getTileY() ).equals("e")){

p.move(1, 0); //Player Moves

VM1(v); //Mummy Moves

}

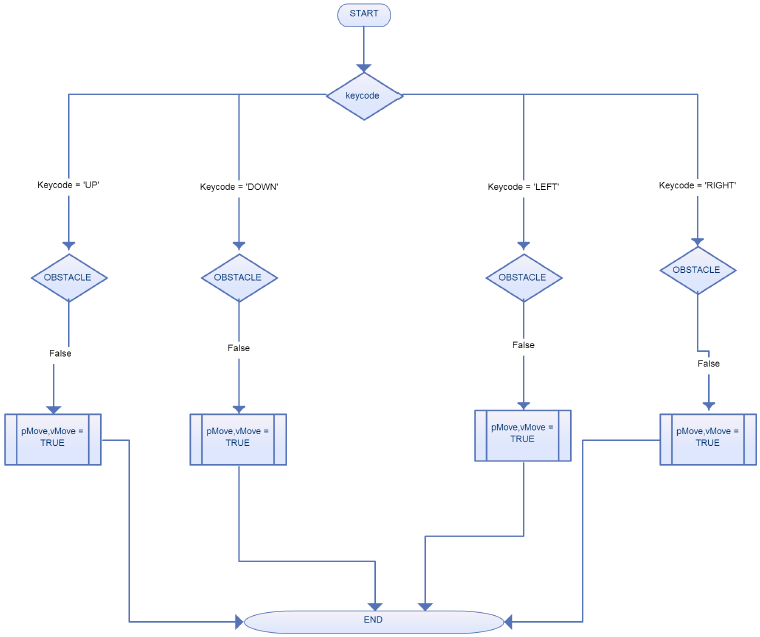
}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end of code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Testing Plan:

1. **Path Testing:** This type of testing explores the all the possible paths that our code can traverse. So, Inputs are given to test the successful execution of each and every path.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST CASE** | **Key Code** | **Water/Wall Presence** | **Expected Output** | **Actual Output** | **Result** |
| TC01 | UP | FALSE | The Player and Villain should move. | Player Moves; Villain Moves | PASS |
| TC02 | DOWN | FALSE | The Player and Villain should move. | Player Moves; Villain Moves | PASS |
| TC03 | LEFT | FALSE | The Player and Villain should move. | Player Moves; Villain Moves | PASS |
| TC04 | RIGHT | FALSE | The Player and Villain should move. | Player Moves; Villain Moves | PASS |
| TC05 | UP | TRUE | The Player or Villain should not move. | Player does not moves; Villain does not moves | PASS |
| TC06 | DOWN | TRUE | The Player or Villain should not move. | Player does not moves; Villain does not moves | PASS |
| TC07 | LEFT | TRUE | The Player or Villain should not move. | Player does not moves; Villain does not moves | PASS |
| TC08 | RIGHT | TRUE | The Player or Villain should not move. | Player does not moves; Villain does not moves | PASS |

**2. Equivalence Testing:**

**Step 1: Identification of the equivalence classes**

Divide the input variables into valid and invalid values.

Variables for this method along with the possible values:

**Key Code**: {UP, DOWN, LEFT, RIGHT}🡨 Valid

{All other keys on keyboard other than above four}🡨 Invalid

**Water/Wall Presence**: {TRUE, FALSE}

**Step 2: Selection of the test inputs**

We have already included the test cases for valid inputs in the path testing and the invalid case occurs when any Invalid Key Code is pressed in which case there can be umpteen number of test cases. There is no invalid input for the other variable, i.e. Water/Wall Presence.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST CASE** | **Key Code** | **Water/Wall Presence** | **Expected Output** | **Actual Output** | **Result** |
| TC09 | ‘K’ | FALSE | The Player and Villain should not move. | Player does not moves; Villain does not moves | PASS |
| TC10 | ‘9’ | FALSE | The Player and Villain should not move. | Player does not moves; Villain does not moves | PASS |
| TC11 | ‘Spacebar’ | FALSE | The Player and Villain should not move. | Player does not moves; Villain does not moves | PASS |

3. **Boundary Testing:**

This type of testing is special case of equivalence testing which checks the code at boundary of the equivalent classes. The boundary cases are already covered by **Path** and **Equivalence Testing.**