**Final Testing Report** for **MUMMY MAZE PLUS**

**[Group 9]**

*Basil Issac, Bradley Robinson, Kanishka Garg, Shubham Jain*

1. **Unit testing**, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level.

|  |  |
| --- | --- |
| **Mummy AI**  Total Number of Test Cases = 12  Test Cases Passed = 12  Test Cases Failed = 0 |  |
| **Graphics**  Total Number of Test Cases = 14  Test Cases Passed = 14  Test Cases Failed = 0 |  |
| **Player Movement Special Cell**  Total Number of Test Cases = 10  Test Cases Passed = 10  Test Cases Failed = 0 |  |
| **Movement** **Player Villain**  Total Number of Test Cases = 11  Test Cases Passed = 11  Test Cases Failed = 0 |  |

*(Double click the file icons to open)*

**Mummy AI Unit Test Cases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Author of the Component: Shubham Jain  Inspection Team Members: Basil Issac, Bradley Robinson, Kanishka Garg  Date: 30th November 2013  **UNIT TESTING**  Component/method/function signature: public void VM\_AI(Villain v)  I/P for the method: An object of type Villain  O/P: Move made by the Villain character according to the move made by the Player character object  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*code snippet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*Mummy AI \*/  **public** **void** VM(Villain v){  **int** deltaX = p.getTileX() - v.getVillainX();  **int** deltaY = p.getTileY() - v.getVillainY();  deltaX = -deltaX;  deltaY = -deltaY;  System.*out*.println("DeltaX = "+deltaX);  System.*out*.println("DeltaY = "+deltaY);  **if**(Math.*abs*(deltaX) < Math.*abs*(deltaY))  {  **if**(deltaX < 0)  moveRight(deltaX, deltaY);  **else**  {  **if**(deltaX > 0)  moveLeft(deltaX, deltaY);  **else**  {  **if**(deltaY > 0)  moveUp(deltaX, deltaY);  **else**  moveDown(deltaX, deltaY);  }  }  }  **else** **if**(Math.*abs*(deltaX) > Math.*abs*(deltaY))  {  **if**(deltaY < 0)  moveDown(deltaX, deltaY);  **else**  {  **if**(deltaY > 0)  moveUp(deltaX, deltaY);  **else**  {  **if**(deltaX > 0)  moveLeft(deltaX, deltaY);  **else**  moveRight(deltaX, deltaY);  }  }  }  **else**  {  System.*out*.println("Exception Error Thrown");  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end of method\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  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.     **INPUT**   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | TEST CASE | deltaX | deltaY | Expected Output | Actual Output | Result | | TC01 | -2 | 4 | Villain character should move to the right. | Mummy moves right | PASS | | TC02 | 4 | 8 | Villain character should move to the left. | Mummy moves left | PASS | | TC03 | 0 | 10 | Villain character should move to the up. | Mummy moves up | PASS | | TC04 | 0 | -8 | Villain character should move to the down. | Mummy moves down | PASS | | TC05 | 3 | -2 | Villain character should move to the down. | Mummy moves down | PASS | | TC06 | 5 | 2 | Villain character should move to the up. | Mummy moves up | PASS | | TC07 | 10 | 0 | Villain character should move to the left. | Mummy moves left | PASS | | TC08 | -8 | 0 | Villain character should move to the right. | Mummy moves right | PASS |  1. **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:  deltaX:{-infinity, 0}, {0, infinity}  deltaY:{-infinity, 0}, {0, infinity}  **Step 2: Selection of the test inputs**  We have already included the test cases for valid inputs in the path testing and the only possible invalid case in case deltaX = 0 and deltaY = 0, in which case an exception gets raised. Other invalid cases can occur when deltaX and deltaY takes very large values and the logic fails due to overflow errors in hardware.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **TEST CASE** | **deltaX** | **deltaY** | **Expected Output** | **Actual Output** | **Result** | | TC09 | 0 | 0 | VillainandPlayerPositionException shall be raised. | As expected | PASS | | TC10 | Any long int  (like, 999880098) | -2 | IntegerOutofBound Exception shall be raised. | As expected | PASS | | TC11 | 9 | 111009832388 | IntegerOutofBound Exception shall be raised. | As expected | PASS | | TC12 | 98190091809 | 98190091809 | IntegerOutofBound Exception shall be raised. | As expected | PASS |  1. **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 TC 03, 04, 07, 08 in **Path Testing** and TC 09 in **Equivalence Testing.** |

**Graphics Unit Test Cases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Author of the Component: Basil Issac  Inspection Team Members: Bradley Robinson, Kanishka Garg and Shubham Jain  Date: 30th November 2013  Testing Component/Method/Function Name: Public void paint(Graphics g)  I/P Data: displayWelcomeScreen, displayMainScreen, displaySelectLevel, displayMap  O/P: Screens for the game shall be displayed according to the input received  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **public** **void** paint(Graphics g){**super**.paint(g);  **if**(!win){  **if**(displayWelcomeScreen){  g.drawImage(m.getWelcomeScreen(), 0, 0, 640, 640, **null**);  }  **else**{  **if**(displayMainScreen){  g.drawImage(m.getMainScreen(), 0, 0, 640, 640, **null**);  }  **else**{  **if**(displaySelectLevel){  g.drawImage(m.getSelectLevelScreen(), 0, 0, 640, 640, **null**);  }  **else** **if**(displayHighScore){  g.drawImage(m.getScoreScreen(), 0, 0, 640, 640, **null**);  g.setFont(**new** Font("Arial", Font.BOLD, 18));  g.setColor(Color.white);  **int** verticalSpace = 150;  **for**(**int** i = 0; i < mapHeight; i++){  g.drawString(m.getScores(i), 5\*32, verticalSpace+i\*32);  }  }  **else**{  **if**(displayMap){  **for**(**int** i = 0; i< mapHeight; i++){  **for**(**int** j = 0; j< mapWidth; j++)  **if**(m.getMap(i,j).equals("f"))  g.drawImage(m.getFinish(), i \* 32, j \* 32, 32, 32, **null**);  **if**(m.getMap(i,j).equals("g"))  g.drawImage(m.getGrass(), i \* 32, j \* 32, 32, 32, **null**);  **if**(m.getMap(i,j).equals("w"))  g.drawImage(m.getWall(), i \* 32, j \* 32, 32, 32, **null**);  **if**(m.getMap(i,j).equals("e"))  g.drawImage(m.getWater(), i \* 32, j \* 32, 32, 32, **null**);  **if**(m.getMap(i,j).equals("l"))  g.drawImage(m.getLadder(), i\*32, j\*32, 32, 32, **null**);  **if**(m.getMap(i,j).equals("p"))  g.drawImage(m.getFirepole(), i\*32, j\*32, 32, 32, **null**);  **if**(m.getMap(i,j).equals("o"))  g.drawImage(m.getPitfall(), i\*32, j\*32, 32, 32, **null**);  //draws player  g.drawImage(p.getPlayer(),p.getTileX() \* 32, p.getTileY() \* 32, 32, 32, **null**);  //draws villain  g.drawImage(v.getVillain(),v.getVillainX() \* 32, v.getVillainY() \* 32, 32, 32, **null**);  //draws heart  g.drawImage(h1.getHeart(), 0 \* 32, 19 \* 32, 32, 32, **null**);  g.drawImage(h2.getHeart(), 1 \* 32, 19 \* 32, 32, 32, **null**);  g.drawImage(h3.getHeart(), 2 \* 32, 19 \* 32, 32, 32, **null**);  //draws Floor, Level, Score  g.setFont(**new** Font("Arial", Font.BOLD, 28));  g.setColor(Color.white);  g.drawString("Floor: "+floor, 16\*32, 1\*32);  g.drawString("Level: "+level, 0\*32, 1\*32);  g.drawString("Score: "+score, 8\*32, 1\*32);  //draws Timer  g.setFont(**new** Font("Arial", Font.BOLD, 28));  g.setColor(Color.yellow);  **long** minutes = (time/3600);  **long** seconds = (time/60)%60;  g.drawString("Time Left: "+Long.toString(minutes)+"m"+Long.toString(seconds)+"s", 12\*32, 20\*32);  }  **else**  System.out.println(“Exception Error Thrown”);  }  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.     **INPUT**   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **TEST CASE** | **WIN** | **displayWelcome Screen** | **displayMain**  **Screen** | **displaySelect**  **Level** | **displayMap** | **Expected Output** | **Actual Output** | **Result** | | TC01 | False | True | False | False | False | The Welcome screen shall be displayed. | The Welcome screen gets displayed. | PASS | | TC02 | False | False | True | False | False | The Main Screen shall be displayed. | The Main Screen gets displayed | PASS | | TC03 | False | False | False | True | False | The Select Level screen shall be displayed. | The Select Level screen gets displayed. | PASS | | TC04 | False | False | False | False | True | The High Score screen shall be displayed. | The High Score screen gets displayed. | PASS | | TC05 | True | False | False | False | False | The Win screen shall be displayed. | The Win screen gets displayed. | PASS | | TC06 | False | False | False | False | False | The displayMapError Exception shall be raised and displayed. | Exception Throwned | PASS |  1. **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:  WIN: {True,False}  displayWelcomescreen: {True,False}  displayMainscreen: {True,False}  displaySelectLevel: {True,False}  displayHighScores: {True,False}  **Step 2: Selection of the test inputs**  There can be 32 possible test cases in which 6 are valid cases and already covered in the **path testing** and we will test for a few of the rest 26 test cases in which case the code will raise a InvalidInputException.   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **TEST CASE** | **WIN** | **displayWelcome Screen** | **displayMain**  **Screen** | **displaySelect**  **Level** | **displayMap** | **Expected Output** | **Actual Output** | **Result** | | TC07 | False | True | **T**rue | False | False | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC08 | False | True | **T**rue | False | True | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC09 | True | True | **T**rue | False | False | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC10 | False | True | **T**rue | True | True | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC11 | False | False | **False** | True | True | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC12 | False | False | **T**rue | True | True | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC13 | True | True | **T**rue | True | True | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS | | TC14 | False | False | **T**rue | False | True | The InvalidInput Exception shall be raised and displayed. | Exception Throwned | PASS |  1. **Boundary Testing:** This type of testing is special case of equivalence testing which checks the code at boundary of the equivalent classes. In this case, we have Boolean type of inputs and have already covered the boundary cases in the path testing test cases. So, the test cases for Boundary testing are same as Path testing. |

**Player Movement Special Cell Unit Test Cases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Author of the Component: Bradley Robinson  Inspection Team Members: Kanishka Garg, Basil Issac, Shubham Jain  Date: 29th November 2013  **UNIT TESTS**  Component/method/function signature: public void playerMovesSpecialCell ()  I/P: The character in the Map file.  O/P: Processing done in the system/game on the basis of the data in the Map file.  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*code snippet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  public void playerMovesSpecialCell(){  if(m.getMap(p.getTileX(), p.getTileY()).equals("o")){  decreaseHealth();  p.resetPlayer();  v.resetVillian();  System.out.println("Health Decreases");  }  if(m.getMap(p.getTileX(), p.getTileY()).equals("f")){  win = true;  System.out.println("WON LEVEL: "+level);  level++;  score+=100;  floor=1;  m.openFile("Map"+level+"-1");  p.resetPlayer();  v.resetVillian();  }  if(m.getMap(p.getTileX(), p.getTileY()).equals("p")){  System.out.println("PLAYER IS MOVING FLOOR DOWN");  floor--;  m.openFile("Map1-"+floor);  v.resetVillian();  }  //l is ladder  if(m.getMap(p.getTileX(), p.getTileY()).equals("l")){  System.out.println("PLAYER IS MOVING FLOOR UP");  floor++;  m.openFile("Map1-"+floor);  v.resetVillian();  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  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** | **char** | **Expected Output** | **Actual Output** | **Result** | | TC01 | ‘o’ | The Player’s health should be decreased and The Player and Villain should be reset to initial position. | Player loses health;  Player position reset;  Villain position reset; | PASS | | TC02 | ‘f’ | The Player shall win that level, level should increase by 1, score shall increase by 100, floor shall increase by 1, and the Player and Villain shall reset after opening the new file. | Player finishes particular level, displays a winning screen and moves to next level. | PASS | | TC03 | ‘p’ | The Player shall move to the lower floor by decreasing the floor number by 1 and Villain shall reset after opening the new file of the floor. | Player moves to the floor below. | PASS | | TC04 | ‘l’ | The Player shall move to the upper floor by decreasing the floor number by 1 and Villain shall reset after opening the new file of the floor. | Player moves to the floor above. | 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:  **Char**: {‘o’,’f’,’p’,’l’,’g’,’w’,’e’}🡨 Valid  {0-9}, {special symbols}, {other characters other than above 7}🡨 Invalid  **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 character is included in the map file in which case there can be umpteen number of test cases.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **TEST CASE** | **char** | **Expected Output** | **Actual Output** | **Result** | | TC05 | ‘g’ | The Player shall move to the next cell. | Player moves on next cell having grass. | PASS | | TC06 | ‘w’ | The Player shall not be able to move beyond. | Player cannot move since wall is there. | PASS | | TC07 | ‘e’ | The Player shall not be able to move beyond. | Player cannot move since water is there. | PASS | | TC08 | ‘4’ | The InvalidCharacterException shall be raised. | Exception thrown. | PASS | | TC09 | ‘#’ | The InvalidCharacterException shall be raised. | Exception thrown. | PASS | | TC10 | ‘\*’ | The InvalidCharacterException shall be raised. | Exception thrown. | 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.** |

**Movement** **Player Villain Unit Test Cases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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.** |

1. **Integration testing** is the type of software testing that seeks to verify the interfaces between components against a software design.



*(Double click the file to open)*

|  |  |  |  |
| --- | --- | --- | --- |
| Total Number of Test Cases | Total Number of Test Cases Passed | Total Number of Test Cases Failed | Total Number of Test Cases ( No Run ) |
| 15 | 15 | 0 | 0 |

**Test Cases**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **INTEGRATION TESTING**  Unit testing focuses in individual components. Integration testing focuses on small groups of components. Two or more components are tested together in this type of testing.  We have selected to conduct **Big Bang Testing** which is a **Horizontal Integration** testing strategy. The reason for selecting this strategy is because the size of our system is small and it is easy to test a small system using this strategy.  We have a total of seven classes in our system along with their primary responsibilities:   1. **Board.java**: The main logic of the game in inside this class. It is responsible for the display output of the game. Other following classes are instantiated in the Board.java 2. **Map.java:** Responsible for the creation and loading of the Map designs. 3. **Maze.java** (init class): Initializes the applet and refers to Board.java 4. **Sound.java**: Responsible for background music of the game. 5. **Heart.java:** Responsible for the management of lives of the Player character. 6. **Player.java**: Responsible for the Player character, its movement and display. 7. **Villain.java**: Responsible for the Villain character, its movement and display.  |  |  |  |  |  | | --- | --- | --- | --- | --- | | **TEST CASE** | **TEST DESCRIPTION** | **EXPECTED OUTPUT** | **ACTUAL OUTPUT** | **RESULT** | | INT\_TC\_01 | **Precondition:** The game is in test mode and the tester invokes player position reset function by making mummy AI catch the Player. | The Player position shall get reset. | Player position gets set to initial position. | PASS | | INT\_TC\_02 | **Precondition:** The game is in test mode and tester invokes Villain Move function by moving the player by one block. The mummy AI mirrors the move made by the Player. | The Villain character shall move. | Villain gets moved according to AI algorithm. | PASS | | INT\_TC\_03 | **Precondition:** The game is in test mode and the tester & level designer decides to test the functionality of loading of the maps. | The Map shall be successfully created and displayed. | Map gets displayed according to the designs made by the level designer. | PASS | | INT\_TC\_04 | **Precondition:** The game is in test mode and the tester decides to test the sound integration with the game. | The sound shall be played when Mummy Maze Plus is running. | Sound gets played in the background. | PASS | | INT\_TC\_05 | **Precondition:** The game is in test mode and the tester decides to run the game as a java applet. | The Mummy Maze Plus shall work as a java applet calling all the other java classes. | All the classes’ gets loaded and the game starts. | PASS | | INT\_TC\_06 | **Precondition:** The game is in test mode, the tester selects the “Select Level” to check the integration of screens. | The Select Level screen shall be displayed. | Select Level screen gets displayed. | PASS | | INT\_TC\_07 | **Precondition:** The game is in test mode, the tester presses “P” to pause the game. | The pause screen shall be displayed showing options to go to the main menu/quit. | Game gets paused and A screen gets displayed. | PASS | | INT\_TC\_08 | **Precondition**: The game is in test mode, the tester presses up, down, left, right arrow keys to move the player. Tester gives a valid command using keyboard. | The Player character shall move up/down/right/left | Player gets moved Up, Down, Right, Left. | PASS | | INT\_TC\_09 | **Precondition**: The game is in test mode, the tester moves player in the pitfall. | The Player shall lose a life and Player life counter shall be decremented by 1. | The life counter gets decreased by 1 and player loses a life. | PASS | | INT\_TC\_10 | **Precondition**: The game is in test mode, the tester moves player to the finishing point on a particular level. | The system shall load the next level on completion of the current level. | The go to the next level screen gets displayed. | PASS | | INT\_TC\_11 | **Precondition**: The game is in test mode, the tester moves player over the firepole. | The Player shall move down the firepole and reach the lower floor. | Player gets moved down to the lower floor. | PASS | | INT\_TC\_12 | **Precondition**: The game is in test mode, the tester moves player over the ladder. | The Player shall move up the ladder and reach the upper floor. | Player gets moves up to the next floor. | PASS | | INT\_TC\_13 | **Precondition**: The game is in test mode, the tester makes the player complete all the levels. | A win screen shall be displayed. | A win screen gets displayed. | PASS | | INT\_TC\_14 | **Precondition**: The game is in test mode, the tester makes the player move to the next level to reset the timer. | The timer shall be successfully updated as the game progresses. | Timer gets reset at new level. | PASS | | INT\_TC\_15 | **Precondition**: The game is in test mode, the tester selects high score option from the main menu screen. | A table of users & high scores shall be displayed. | High scores gets displayed in a tabulated format. | PASS | |

1. **System testing** tests a completely integrated system to verify that it meets its requirements.



*(Double click the file to open)*

|  |  |  |  |
| --- | --- | --- | --- |
| Total Number of Test Cases | Total Number of Test Cases Passed | Total Number of Test Cases Failed | Total Number of Test Cases ( No Run ) |
| 41 | 38 | 2 | 1 |

**TEST CASES**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **TEST CASE** | **TEST DESCRIPTION** | **EXPECTED OUTPUT** | **ACTUAL OUTPUT** | **RESULT** | **COMMENTS** | | BB\_TC\_01 | **Precondition:**Mummy Maze Plus is in test mode and A user is opening the game,and the game is to be available as a Java web Applet. | The game shall be hosted on a local web server as a Java web applet. |  | FAIL | Not authorized to create signed java applet | | BB\_TC\_02 | **Precondition:**Mummy Maze Plus is in test mode and A user is opening the game,and he/she can access the game applet using a local web address. | The player shall open a local web address to access the Mummy Maze Plus game applet. |  | FAIL | In the modern browsers, an unsigned java applet cannot be hosted locally. | | BB\_TC\_03 | **Precondition:** BB\_TC\_02 is successfully completed and the game loads on the screen. | A welcome screen shall be displayed and the player shall be prompted prompted to press any key to continue. | As Expected | PASS |  | | BB\_TC\_04 | **Precondition:** BB\_TC\_03 is successfully completed and the next screen loads on the screen. | A menu shall appear showing three options to choose from : 1. “Select Level” (Level-1, Level-2, Level-3, Level-4 or Level-5), 2. “High Scores” for viewing players scores, and 3. “Quit Game” for quitting the game. | As Expected | PASS |  | | BB\_TC\_05 | **Precondition:** BB\_TC\_04 is successfully completed and the user chooses the "Select Level" option. | The "Select Level" screen shall load and the player shall be able to choose among five different levels of varying difficulties. | As Expected | PASS |  | | BB\_TC\_06 | **Precondition:** BB\_TC\_05 is successfully completed and the player uses keyboard to select the level. | The player shall be able to select a level via keyboard input and the floor maps shall then be loaded according to the level selected. | As Expected | PASS |  | | BB\_TC\_07 | **Precondition:** BB\_TC\_06 is successfully completed and the user waits for the loading of the play mode. | The level shall load from a top view perspective. | As Expected | PASS |  | | BB\_TC\_08 | **Precondition:** BB\_TC\_06 is successfully completed and the user waits for the loading of the play mode. | Each level shall contain different number of floors based on the level of difficulty | As Expected | PASS |  | | BB\_TC\_09 | **Precondition:** BB\_TC\_06 is successfully completed and the user waits for the loading of the play mode. | The grid size of the loaded maps shall be 20x20 blocks | As Expected | PASS |  | | BB\_TC\_10 | **Precondition:** BB\_TC\_07 is successfully completed, the game is in play mode and maps are successfully loaded. | All floors in a map shall contain obstacles (such as water, pitfalls, etc), walls, ladders (to go above floor), fire-poles (to go lower floor) and an exit (finishing point). | As Expected | PASS |  | | BB\_TC\_11 | **Precondition:** BB\_TC\_07 is successfully completed, the game is in play mode and maps are successfully loaded. | The ground floor in the map shall not have any fire-pole and the top most floor in the map shall not have any ladders. | As Expected | PASS |  | | BB\_TC\_12 | **Precondition:** BB\_TC\_11 is successfully completed and the player decides to explore the maps in all the levels. | All five levels shall be fully designed with different map layouts and shall have multiple floors each with their own tricks and traps. | As Expected | PASS |  | | BB\_TC\_13 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode and the player decides to play the game. ( user is called 'player' in the play mode) | The human character (player) shall move up, down, left or right one step at a time, by using the keyboard. | As Expected | PASS |  | | BB\_TC\_14 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode and the player decides to play the game. | The default keyboard inputs used to move the character shall be “Up Arrow” for up, “Left Arrow” for left, “Down Arrow” for down and “Right Arrow” for right. | As Expected | PASS |  | | BB\_TC\_15 | **Precondition:** BB\_TC\_14 is successfully completed and the player decides to pause the game. | The player shall be able to pause the game by pressing “P” from keyboard | As Expected | PASS |  | | BB\_TC\_16 | **Precondition:** BB\_TC\_15 is successfully completed and the player decides to resume the game. | The player shall be able to resume the game by pressing “R” from keyboard | As Expected | PASS |  | | BB\_TC\_17 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode and the player presses "Escape Key" at any instant during the play mode. | A screen shall be loaded showing the option to exit to main menu by pressing “M” or quit the whole game by pressing “Q”. | As Expected | PASS |  | | BB\_TC\_18 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode, the player is navigating the floor, encounters a ladder and decides to go to the higher floor. | The player shall be able to move to the higher floor using ladder. | As Expected | PASS |  | | BB\_TC\_19 | **Precondition:** BB\_TC\_18 is successfully completed,the game is in play mode, the player is navigating the floor, encounters a firepole and decides to go to the lower floor. | The Player shall be able to move to the lower floor using firepole. | As Expected | PASS |  | | BB\_TC\_20 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode and the player is trying to move past the walls. | The player shall not be able move beyond the walls. | As Expected | PASS |  | | BB\_TC\_21 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode and the player is trying to move past the water pool. | The player shall not be able to move past pools of water. | As Expected | PASS |  | | BB\_TC\_22 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode and the player is trying to move beyond the boundaries. | The player shall not be able to go beyond the boundaries. | As Expected | PASS |  | | BB\_TC\_23 | **Precondition:** BB\_TC\_10 is successfully completed and the game is in play mode. | The mummy AI shall not go beyond the boundaries. | As Expected | PASS |  | | BB\_TC\_24 | **Precondition:** BB\_TC\_10 is successfully completed and the game is in play mode. | The mummy shall use its supernatural mummy powers to walk across water. | As Expected | PASS |  | | BB\_TC\_25 | **Precondition:** BB\_TC\_07 is successfully completed, the game is in play mode, maps are successfully loaded and the player life counter is activated. | The player shall be given 3 lives at the beginning of each level. | As Expected | PASS |  | | BB\_TC\_26 | **Precondition:** BB\_TC\_25 is successfully completed and the mummy AI catches the Player. | The player shall lose a life and the life counter shall decrease by 1. | As Expected | PASS |  | | BB\_TC\_27 | **Precondition:** BB\_TC\_25 is successfully completed and the mummy AI catches the Player. | The player’s position shall be reset back to the beginning of that particular floor. | As Expected | PASS |  | | BB\_TC\_28 | **Precondition:** BB\_TC\_10 is successfully completed,the game is in play mode, player life counter = 1 and the mummy AI catches the Player. | The "game-over"screen shall be displayed to the Player. | As Expected | PASS |  | | BB\_TC\_29 | **Precondition:** BB\_TC\_10 is successfully completed, the game is in play mode and the player decides to explore the number of ways to reach exit in different levels of the game. | The number of ways to reach exit shall be different for different levels. | As Expected | PASS |  | | BB\_TC\_30 | **Precondition:** BB\_TC\_28 is successfully completed. | The score of the player for the current instance of the game shall be displayed. | As Expected | PASS |  | | BB\_TC\_31 | **Precondition:** BB\_TC\_28 is successfully completed. | The player shall be prompted to enter his/her name to save his score for current instance of the game. | As Expected | PASS |  | | BB\_TC\_32 | **Precondition:** BB\_TC\_28 is successfully completed. | The player shall choose an option to exit to main menu by pressing “M” or quit the whole game by pressing “Q”. | As Expected | PASS |  | | BB\_TC\_33 | **Precondition:** BB\_TC\_10 is successfully completed, the game is in play mode, player life counter = 2, level number = 2, floor number = 2 and the Player reaches the finish flag. | A screen shall be displayed asking player to press "N" to move to the next subsequent level. | As Expected | PASS |  | | BB\_TC\_34 | **Precondition:** BB\_TC\_10 is successfully completed, the game is in play mode, player life counter = 1, level number = 5, floor number = 5 and the Player reaches the finish flag. | A final screen shall be displayed informing the player that he/she has obtained the mastery of the game which is the highest attainable honor in Mummy Maze Plus game. | Not Implemented | \_\_ |  | | BB\_TC\_35 | **Precondition:** BB\_TC\_10 is successfully completed and the game is in play mode. | The player shall be able to see a small display showing the attributes of current instance of game like level number, floor number, timer and remaining life count. | As Expected | PASS |  | | BB\_TC\_36 | **Precondition:** BB\_TC\_04 is successfully completed and the user chooses the "High Scores" option. | A screen shall be displayed showing the high scores in a tabulated format. | As Expected | PASS |  | | BB\_TC\_37 | **Precondition:** BB\_TC\_10 is successfully completed, the game is in play mode and the player decides to explore the high score functionality. | Weighted scores shall be calculated based on the amount of time the player takes to complete a particular level, the level difficulty and how often the player gets hit by the mummy in that level. | As Expected | PASS |  | | BB\_TC\_38 | **Precondition:** BB\_TC\_04 is successfully completed and the user chooses the "High Scores" option. | High scores shall be displayed in a tabulated format displaying the player's name in one column and his total score in the other column. | As Expected | PASS |  | | BB\_TC\_39 | **Precondition:** BB\_TC\_02 is successfully completed and the game loads on the screen. | Background music shall be played in the background. | As Expected | PASS |  | | BB\_TC\_40 | **Precondition:** BB\_TC\_10 is successfully completed and the game is in play mode. | Sound effects for critical events like changing the floors, hit by mummy etc. shall be played. | As Expected | PASS |  | | BB\_TC\_41 | **Precondition:** BB\_TC\_10 is successfully completed, the game is in play mode, music & sound effects are enabled and the player decides to disable it by pressing "S". | The background music and sound effects shall be disabled. | As Expected | PASS |  | |

1. **User Acceptance Testing** consists of a process of verifying that a solution works for the customer. It is not system, but rather is there to ensure that the solution will work for the customer.



*(Double click the file to open)*

|  |  |  |  |
| --- | --- | --- | --- |
| Total Number of Test Cases | Total Number of Test Cases Passed | Total Number of Test Cases Failed | Total Number of Test Cases ( No Run ) |
| 14 | 13 | 1 | 0 |

TEST CASES

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **TEST CASE** | **TEST DESCRIPTION** | **EXPECTED OUTPUT** | **ACTUAL OUTPUT** | **RESULT** | | UAT\_TC\_01 | **Precondition:** The game is loaded on the screen. | A welcome screen shall be displayed and the player shall be prompted to press any key to continue. | As Expected | PASS | | UAT\_TC\_02 | **Precondition:** UAT\_TC\_01 is successfully completed. | A menu shall appear showing three options to choose from: 1. “Select Level” (Level-1, Level-2, Level-3, Level-4 or Level-5), 2. “High Scores” for viewing players scores, and 3. “Quit Game” for quitting the game. | As Expected | PASS | | UAT\_TC\_03 | **Precondition:** UAT\_TC\_02 is successfully completed and the customer chooses the "Select Level" option. | The "Select Level" screen shall load and the customer shall be able to choose among five different levels of varying difficulties. | As Expected | PASS | | UAT\_TC\_04 | **Precondition:** UAT\_TC\_03 is successfully completed and the customer enters "2" and waits for the loading of the play mode. | The level shall load from a top view perspective. | As Expected | PASS | | UAT\_TC\_05 | **Precondition:** UAT\_TC\_04 is successfully completed and the game is loaded on the screen.Level = 1. Floor = 1. | The grid size of the loaded maps shall be 20x20 blocks | As Expected | PASS | | UAT\_TC\_06 | **Precondition:** UAT\_TC\_05 is successfully completed and the customer moves up the floor using ladder, moves down the floor using firepole and reaches the exit. Level = 2. Floor = 3. | The customer shall be able to traverse through the floors using Ladders & Firepoles and The next level shall be loaded as the customer reaches the exit point. | As Expected | PASS | | UAT\_TC\_07 | **Precondition:** The game shall be available as a java web applet. | The customer shall be able to run Mummy Maze Plus as a Java Web Applet. | Not Implemented | FAIL | | UAT\_TC\_08 | **Precondition:** UAT\_TC\_07 is successfully completed and The game is in play mode.Level = 1. Floor = 1. | The customer shall be able to experience the music in the background. | As Expected | PASS | | UAT\_TC\_09 | **Precondition:** UAT\_TC\_07 is successfully completed and The game is in play mode.Level = 2. Floor = 2. | The customer shall experience smooth transitions on the floor and in between subsequent floors. | As Expected | PASS | | UAT\_TC\_10 | **Precondition:** UAT\_TC\_07 is successfully completed and The game is in play mode.Level = 1. Floor = 1. | The customer expects the game shall be winnable and A win screen shall be displayed after the customer wins the game. | As Expected | PASS | | UAT\_TC\_11 | **Precondition:** UAT\_TC\_07 is successfully completed and The game is in play mode.Level = 1. Floor = 1. | The customer shall be able to experience relevant graphics and shall represent the theme of the game. | As Expected | PASS | | UAT\_TC\_12 | **Precondition:** UAT\_TC\_07 is successfully completed and The game is in play mode. Level = 2. Floor = 2. | The customer shall be able to pause/quit the game at any point of time. | As Expected | PASS | | UAT\_TC\_13 | **Precondition:** UAT\_TC\_02 is successfully completed and the customer chooses the "High Scores" option. | High Scores table shall be displayed in a tabulated format. | As Expected | PASS | | UAT\_TC\_14 | **Precondition:** UAT\_TC\_04 is successfully completed and the game is loaded on the screen.Level = 1. Floor = 1. | The level number, floor number, life count and timer shall be displayed to the customer. | As Expected | PASS | |

# Final Test Results for Mummy Maze Plus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Total Number of Test Cases | Total Number of Test Cases Passed | Total Number of Test Cases Failed | Total Number of Test Cases ( No Run ) | Test Cases Pass Percentage |
| 117 | 113 | 3 | 1 | ~97.5% |