

## Part 1 functional Testing

For this project we have the game environment set up and have a test background as well. The only functionalities we have is moving the player with w, a, s, and d. In the table below, we tested each input variable as the keys were pressed to ensure it was the proper key. Also, we tested to see if the player did move a distance with each button pressed.

**Table 1: The specification of equivalence partitioning**

Partition ID	Input Variable	Valid Partition	Invalid Partition
1	w	yes	no
2	a	yes	no
3	s	yes	no
4	d	yes	no

**Table 2: The test case specifications for each partition**

Test ID	Test Inputs	Expected Output	Partition ID Covered
1	w	Object moves up	1
2	a	Object moves left	2
3	s	Object moves down	3
4	d	Object moves right	4

## JUnit implementation of the test cases implemented

```
7  import org.junit.After;
8  import org.junit.AfterClass;
9  import org.junit.Before;
10 import org.junit.BeforeClass;
11 import org.junit.Test;
12 import static org.junit.Assert.*;
13
14 import MyGame.*;
15 import java.awt.AWTException;
16 import java.awt.event.KeyEvent;
17 import static junit.framework.Assert.assertEquals;
18 import java.awt.event.KeyAdapter;
19 import java.awt.Robot;
20
21 /**
22  *
23  * @author aaronramirez
24  */
25 public class UnitTest {
26     Game GameDriverInstance;
27     KeyInput MyKeyInput;
28
29
30
31     @Before
32     public void setUpClass() {
33         System.out.println("Gamer test before: Before method test()");
34         GameDriverInstance = new Game();
35     }
36
37     @After
38     public void tearDownClass() {
39         System.out.println("Gamer test after: After method test()");
40         GameDriverInstance = null;
41     }
42
43     @Test
44     public void TestDriverA1() throws AWTException{
45         System.out.println("Test A1 begin");
46         Robot Jess = new Robot();
```

```

46     Robot Jess = new Robot();
47     Jess.keyPress(65);
48     Jess.delay(300);
49     assertEquals(GameDriverInstance.handler.isLeft(), true);
50 }
51
52 @Test
53 public void TestDriverA2() throws AWTEException {
54     System.out.println("Test A2 begin");
55     GameObject G = GameDriverInstance.handler.object.get(64);
56     int xBefore = G.getX();
57     Robot Jess = new Robot();
58     Jess.keyPress(65);
59     Jess.delay(1000);
60     int xAfter = G.getX();
61     boolean ans = false;
62     if (xBefore != xAfter) {
63         ans = true;
64     }
65     assertEquals(ans, true);
66 }
67
68 @Test
69 public void TestDriverW1() throws AWTEException{
70     System.out.println("Test W1 begin");
71     Robot Jess = new Robot();
72     Jess.keyPress(87);
73     Jess.delay(1000);
74     assertEquals(GameDriverInstance.handler.isUp(), true);
75 }
76
77 @Test
78 public void TestDriverW2() throws AWTEException {
79     System.out.println("Test W2 begin");
80     GameObject G = GameDriverInstance.handler.object.get(64);
81     int yBefore = G.getY();
82     Robot Jess = new Robot();
83     Jess.keyPress(87);
84     Jess.delay(1000);
85     int yAfter = G.getY();

```

```

85         int yAfter = G.getY();
86         boolean ans = false;
87         if (yBefore != yAfter) {
88             ans = true;
89         }
90         assertEquals(ans, true);
91     }
92
93     @Test
94     public void TestDriverS1() throws AWTEException{
95         System.out.println("Test S1 begin");
96         Robot Jess = new Robot();
97         Jess.keyPress(83);
98         Jess.delay(1000);
99         assertEquals(GameDriverInstance.handler.isDown(), true);
100     }
101
102     @Test
103     public void TestDriverS2() throws AWTEException {
104         System.out.println("Test S2 begin");
105         GameObject G = GameDriverInstance.handler.object.get(64);
106         int yBefore = G.getY();
107         Robot Jess = new Robot();
108         Jess.keyPress(83);
109         Jess.delay(1000);
110         int yAfter = G.getY();
111         boolean ans = false;
112         if (yBefore != yAfter) {
113             ans = true;
114         }
115         assertEquals(ans, true);
116     }
117
118     @Test
119     public void TestDriverD1() throws AWTEException{
120         System.out.println("Test D1 begin");
121         Robot Jess = new Robot();
122         Jess.keyPress(68);
123         Jess.delay(1000);
124         assertEquals(GameDriverInstance.handler.isRight(), true);

```

```

117
118
119 @Test
120 public void TestDriverD1() throws AWTException{
121     System.out.println("Test D1 begin");
122     Robot Jess = new Robot();
123     Jess.keyPress(68);
124     Jess.delay(1000);
125     assertEquals(GameDriverInstance.handler.isRight(), true);
126 }
127
128 @Test
129 public void TestDriverD2() throws AWTException {
130     System.out.println("Test D2 begin");
131     GameObject G = GameDriverInstance.handler.object.get(64);
132     int xBefore = G.getX();
133     Robot Jess = new Robot();
134     Jess.keyPress(68);
135     Jess.delay(1000);
136     int xAfter = G.getX();
137     boolean ans = false;
138     if (xBefore != xAfter) {
139         ans = true;
140     }
141     assertEquals(ans, true);
142 }
143

```

End: Select Rectangle (getSounds()) No matches

Test Results x

UnitTest x

Tests passed: 100.00 %

All 8 tests passed. (9.037 s)

## Part 2 Structural Testing:

### 100% statement adequacy

JaCoCoCoverage analysis of project "Wizard\_Game" (powered by JaCoCo from EcjEmma)

JaCoCoCoverage analysis of project "Wizard\_Game" (powered by JaCoCo from EcjEmma)

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
MyGame	100%	100%	100%	100%	0 80	0 187	0 43	0 10
Total	0 of 835	100%	0 of 74	100%	0 80	0 187	0 43	0 10

Created with JaCoCo 0.7.6.201602188812

JaCoCoCoverage analysis of project "Wizard\_Game" (powered by JaCoCo from EcjEmma) > MyGame

MyGame

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods	Missed Classes
Game	100%	100%	100%	100%	0 18	0 79	0 7	0 1
KeyListener	100%	100%	100%	100%	0 15	0 21	0 3	0 1
Wizard	100%	100%	100%	100%	0 11	0 17	0 3	0 1
Handler	100%	100%	100%	100%	0 14	0 25	0 12	0 1
Camera	100%	100%	100%	100%	0 8	0 13	0 4	0 1
Window	100%	100%	n/a	n/a	0 1	0 11	0 1	0 1
ID	100%	100%	n/a	n/a	0 4	0 3	0 4	0 1
GameObject	100%	100%	n/a	n/a	0 4	0 9	0 4	0 1
Block	100%	100%	n/a	n/a	0 3	0 6	0 3	0 1
BufferImageLoader	100%	100%	n/a	n/a	0 2	0 3	0 2	0 1
Total	0 of 835	100%	0 of 74	100%	0 80	0 187	0 43	0 10

Created with JaCoCo 0.7.6.201602188812

## Part 3: Contribution of each teammate:

Harsh: He did structural and functional testing for the W button. He went and tested the positions of the start and end points when the button is pressed. He also tested if the button is pressed it, the player character would be moved up. He also helped find the

test coverage tool JaCoCo. He double checked the JaCoCo testing and got the same results as Abbey.

Jess: She did structural and function testing for the S button. She went and tested the positions of the start and end points when the button is pressed. She also tested if the button is pressed it, the player character would be moved down. She did the partition ID table.

Aaron: He did structural and functional testing for the A button. He went and tested the positions of the start and end points when the button is pressed. He also tested if the button is pressed it, the player character would be moved left. He did the Test ID table with the testing.

Abbey: She did structural and function testing for the D button. She went and tested the positions of the start and end points when the button is pressed. She also tested if the button is pressed it, the player character would be moved right. She also ran the JaCoCo testing and showed the results.