Software Testing Assignment 2: Category Partitioning and Boundary Values

Lars De Leeuw 20205693
Robbe Teughels 20211127

March 2025

Contents

1 CATEGORY-PARTITION AND BOUNDARY VALUES

1.1 Exercise 1

List at least three possible errors that an implementor of this method could make. - *Robbe*

- *Incorrect Boundary Comparison*
  Mismatch between $\leq$ *and* $<$
- *Swapping width and height*
  $0 \leq y < width$ *and* $0 \leq x < height$
- *Incorrect Logical Operator*
  Using `or` *instead of* `and`

1.2 Exercise 2

Identify the equivalence partitions for the withinBorders method. You should provide a table similar to Table 5.1 (see Forgács). You may assume the single fault model. - *Robbe*

Equivalence Partitions	**Attributes**	
	x Condition	**y Condition**
1	$0 \leq x < width$	$0 \leq y < height$
2	$\mathbf{x < 0}$	$0 \leq y < height$
3	$\mathbf{width \leq x}$	$0 \leq y < height$
4	$0 \leq x < width$	$\mathbf{y < 0}$
5	$0 \leq x < width$	$\mathbf{height \leq y}$
6	*Inputs outside partitions above*	

Table 1: Equivalence Partitions of withinBorders Method, single fault model

1.3 Exercise 3

Similarly, identify the equivalence partitions for the withinBorders method, but for the multiple fault model. - *Robbe*

Equivalence Partitions	Attributes	
	x Condition	**y Condition**
1	$0 \leq x < width$	$0 \leq y < height$
2	$\mathbf{x < 0}$	$0 \leq y < height$
3	$\mathbf{width \leq x}$	$0 \leq y < height$
4	$0 \leq x < width$	$\mathbf{y < 0}$
5	$0 \leq x < width$	$\mathbf{height \leq y}$
6	$\mathbf{x < 0}$	$\mathbf{y < 0}$
7	$\mathbf{x < 0}$	$\mathbf{height \leq y}$
8	$\mathbf{width \leq x}$	$\mathbf{y < 0}$
9	$\mathbf{width \leq x}$	$\mathbf{height \leq y}$

Table 2: Equivalence Partitions of withinBorders Method, multiple fault model

1.4 Exercise 4

Next, develop a test design against predicate faults (see Forgács, Table 5.2). You should have a table for predicate x and a table for predicate y. - *Lars*

Table 3: Test design against predicate faults for x. (Assuming $width > 2$)

Program version no.	**Predicate**	**Test data 1**	**Test data 2**	**Test data 3**	**Test data 4**	**Test data 5**	**Test data 6**
	x	-1	0	1	width - 1	width	width + 1
1 (correct)	$0 \leq x < width$	F	T	T	T	F	F
2	$0 \leq x < width - 1$	F	T	T	F	F	F
3	$0 \leq x < width + 1$	F	T	T	T	T	F
4	$0 \leq x \leq width$	F	T	T	T	T	F
5	$0 < x < width$	F	F	T	T	F	F
6	$0 < x < width - 1$	F	F	T	F	F	F
7	$0 < x < width + 1$	F	F	T	T	T	F
8	$0 < x \leq width$	F	F	T	T	T	F
9	$-1 \leq x < width$	T	T	T	T	F	F
10	$-1 \leq x < width - 1$	T	T	T	F	F	F
11	$-1 \leq x < width + 1$	T	T	T	T	T	F
12	$-1 \leq x \leq width$	T	T	T	T	T	F
13	$-1 < x < width$	F	T	T	T	F	F
14	$-1 < x < width - 1$	F	T	T	F	F	F
15	$-1 < x < width + 1$	F	T	T	T	T	F
16	$-1 < x \leq width$	F	T	T	T	T	F
17	$1 \leq x < width$	F	F	T	T	F	F
18	$1 \leq x < width - 1$	F	F	T	F	F	F
19	$1 \leq x < width + 1$	F	F	T	T	T	F
20	$1 \leq x \leq width$	F	F	T	T	T	F
21	$1 < x < width$	F	F	F	T	F	F
22	$1 < x < width - 1$	F	F	F	F	F	F
23	$1 < x < width + 1$	F	F	F	T	T	F
24	$1 < x \leq width$	F	F	F	T	T	F

Table 4: Test design against predicate faults for y. (Assuming $height > 2$)

Program version no.	**Predicate**	**Test data 1**	**Test data 2**	**Test data 3**	**Test data 4**	**Test data 5**	**Test data 6**
	y	-1	0	1	height - 1	height	height + 1
1 (correct)	$0 \leq y < height$	F	T	T	T	F	F
2	$0 \leq y < height - 1$	F	T	T	F	F	F
3	$0 \leq y < height + 1$	F	T	T	T	T	F
4	$0 \leq y \leq height$	F	T	T	T	T	F
5	$0 < y < height$	F	F	T	T	F	F
6	$0 < y < height - 1$	F	F	T	F	F	F
7	$0 < y < height + 1$	F	F	T	T	T	F
8	$0 < y \leq height$	F	F	T	T	T	F
9	$-1 \leq y < height$	T	T	T	T	F	F
10	$-1 \leq y < height - 1$	T	T	T	F	F	F
11	$-1 \leq y < height + 1$	T	T	T	T	T	F
12	$-1 \leq y \leq height$	T	T	T	T	T	F
13	$-1 < y < height$	F	T	T	T	F	F
14	$-1 < y < height - 1$	F	T	T	F	F	F
15	$-1 < y < height + 1$	F	T	T	T	T	F
16	$-1 < y \leq height$	F	T	T	T	T	F
17	$1 \leq y < height$	F	F	T	T	F	F
18	$1 \leq y < height - 1$	F	F	T	F	F	F
19	$1 \leq y < height + 1$	F	F	T	T	T	F
20	$1 \leq y \leq height$	F	F	T	T	T	F
21	$1 < y < height$	F	F	F	T	F	F
22	$1 < y < height - 1$	F	F	F	F	F	F
23	$1 < y < height + 1$	F	F	F	T	T	F
24	$1 < y \leq height$	F	F	F	T	T	F

1.5 Exercise 5

Implement your test case specs into Board Test class and run the test suite. - *Robbe*

```
//     // Equivalence partitions for x
//     private int x_1 = -1;
//     private int x_2 = 0; // width -1
//     private int x_3 = width;
//     // Equivalence partitions for y
//     private int y_1 = -1;
//     private int y_2 = 0; // height -1
//     private int y_3 = theBoard.getHeight();

    @Test
    public void testWithinBoarders22() {
        assertTrue(theBoard.withinBorders(0,0));
        assertTrue(theBoard.withinBorders(width -1,0));
        assertTrue(theBoard.withinBorders(0,height - 1));
        assertTrue(theBoard.withinBorders(width -1,height -1));
    }

    @Test
    public void testWithinBoarders12() {
        assertFalse(theBoard.withinBorders(-1,0));
        assertFalse(theBoard.withinBorders(-1,height -1));
    }

    @Test
    public void testWithinBoarders32() {
        assertFalse(theBoard.withinBorders(width,0));
        assertFalse(theBoard.withinBorders(width,height -1));
    }

    @Test
    public void testWithinBoarders21() {
        assertFalse(theBoard.withinBorders(0,-1));
        assertFalse(theBoard.withinBorders(width -1,-1));
    }


    @Test
    public void testWithinBoarders23() {
        assertFalse(theBoard.withinBorders(0,height));
        assertFalse(theBoard.withinBorders(width -1,height));
    }

    @Test
    public void testWithinBoarders11() {
        assertFalse(theBoard.withinBorders(-1,-1));
    }

    @Test
    public void testWithinBoarders13() {
        assertFalse(theBoard.withinBorders(-1,height));
    }

```

```
53      @Test
54      public void testWithinBoarders31() {
55          assertFalse(theBoard.withinBorders(width,-1));
56      }
57
58      @Test
59      public void testWithinBoarders33() {
60          assertFalse(theBoard.withinBorders(width,height));
61      }
```

Listing 1: "JUnit tests for withinBoarders methode."

1.6 Exercise 6

Implement withinBoarders method. Then re-run the test suite. Inspect code coverage results, and explain your findings. - *Robbe*

The test coverage of both, the old implementation and new have 100% line coverage. However the new implementation also has 100% branche coverage while the old was not applicable for this metric. The test also uncovered defects in the old version. This means that full code coverage in the reports does not guarantee a correct implementation.

1.7 Exercise 8

Would your test suite reveal all the faults you proposed in Exercise 1? If not, explain why the equivalence partitioning approach missed it, and add appropriate test cases separately to your JUnit implementation. - *Robbe*

The equivalence partitioning does not cover the "one of" mistakes by switching $<$ and $\leq$. This can be tested with the on-point values. I have strategically chosen the integer values to be on-point values in their partition class. For the partitioning with 2 on points, I have extended the test to test all possible combinations within the partitioning.
The tests accidentally covers the case of switching width and height by defining a test board with height != width. Combined with the off-point values, at least one test case will fail if they are mixed up

1.8 Exercise 10

Repeat the exercises for the methodGame.addGuestFromCode(char code, int x, int y). How many test cases do you end up with? - *Lars*

1.8.1 List at least three possible errors that an implementer of this method could make.

- *Off by one error*
  Mismatch between $\leq$ *and* $<$ *causing 0 to not be a valid value for x and y*
- *Creating the wrong type of Guest for the given char.*
  E.g. creating a food-type guest for the wall-type char.
- *Using x for the height and y for the width.*
  $0 \leq y < width$ *and* $0 \leq x < height$
- *Forgetting to check whether the cell is already occupied by a non-empty guest.*

1.8.2 Identify the equivalence partitions for the single fault model

The partitions can be seen in table 5. For the single fault model, there is only one extra partition compared to withinBorders. This is because Java is a strongly typed language, so the only two partitions we have for char are that it is either in the set of recognized characters or not.

Equivalence Partitions	**Attributes**		
	x Condition	**y Condition**	**char Condition**
1	$0 \leq x < width$	$0 \leq y < height$	$\texttt{char} \in \{'W','P','F','M','0'\}$
2	$\mathbf{x < 0}$	$0 \leq y < height$	$\texttt{char} \in \{'W','P','F','M','0'\}$
3	$\mathbf{width \leq x}$	$0 \leq y < height$	$\texttt{char} \in \{'W','P','F','M','0'\}$
4	$0 \leq x < width$	$\mathbf{y < 0}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
5	$0 \leq x < width$	$\mathbf{height \leq y}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
6	$0 \leq x < width$	$0 \leq y < height$	$\mathbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$

Table 5: Equivalence Partitions of methodGame.addGuestFromCode Method, single fault model.

1.8.3 Identify the equivalence partitions for the multiple fault model

The partitions can be seen in table 6. We end up with a total of 18 partitions. Which is expected because both x and y have 3 equivalence partitions each and char has 2, so under the multiple fault model we need a test case for every combination achieved by the cartesian product:

$$\mathbf{partitions_x} \times \mathbf{partitions_y} \times \mathbf{partitions_{char}}$$

Equivalence Partitions	Attributes		
	x Condition	**y Condition**	**char Condition**
1	$0 \leq x < width$	$0 \leq y < height$	$\texttt{char} \in \{'W','P','F','M','0'\}$
2	$0 \leq x < width$	$0 \leq y < height$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
3	$0 \leq x < width$	$\mathbf{y < 0}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
4	$0 \leq x < width$	$\mathbf{y < 0}$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
5	$0 \leq x < width$	$\mathbf{height \leq y}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
6	$0 \leq x < width$	$\mathbf{height \leq y}$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
7	$\mathbf{x < 0}$	$0 \leq y < height$	$\texttt{char} \in \{'W','P','F','M','0'\}$
8	$\mathbf{x < 0}$	$0 \leq y < height$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
9	$\mathbf{x < 0}$	$\mathbf{y < 0}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
10	$\mathbf{x < 0}$	$\mathbf{y < 0}$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
11	$\mathbf{x < 0}$	$\mathbf{height \leq y}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
12	$\mathbf{x < 0}$	$\mathbf{height \leq y}$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
13	$\mathbf{width \leq x}$	$0 \leq y < height$	$\texttt{char} \in \{'W','P','F','M','0'\}$
14	$\mathbf{width \leq x}$	$0 \leq y < height$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
15	$\mathbf{width \leq x}$	$\mathbf{y < 0}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
16	$\mathbf{width \leq x}$	$\mathbf{y < 0}$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$
17	$\mathbf{width \leq x}$	$\mathbf{height \leq y}$	$\texttt{char} \in \{'W','P','F','M','0'\}$
18	$\mathbf{width \leq x}$	$\mathbf{height \leq y}$	$\textbf{char} \notin \{'\mathbf{W}','\mathbf{P}','\mathbf{F}','\mathbf{M}','\mathbf{0}'\}$

Table 6: Equivalence Partitions of methodGame.addGuestFromCode Method, multiple fault model.

1.8.4 Implement your test case specs into Game Test class and run the test suite.

Implementation of all 18 test-cases can be seen in 2. In all but one of the test-cases I expect AssertionErrors.

```
    // // Equivalence partitions for char
    // private char char_1 = 'W';
    // private char char_2 = 'Q';
    // // Equivalence partitions for x
    // private int x_1 = -1;
    // private int x_2 = 0;
    // private int x_3 = theGame.boardWidth();
    // // Equivalence partitions for x
```

```
    // private int y_1 = -1;
    // private int y_2 = 0;
    // private int y_3 = theGame.boardHeight();
    @Test(expected = AssertionError.class)
    public void testAddGuestFromCode111() {
        try {
            theGame.addGuestFromCode('W', -1, -1);
        } catch (Exception e)
        {
            assert false;
        }
    }
    @Test(expected = AssertionError.class)
    public void testAddGuestFromCode211() {
        try {
            theGame.addGuestFromCode('Q', -1, -1);
        } catch (Exception e)
        {
            assert false;
        }
    }
    @Test(expected = AssertionError.class)
    public void testAddGuestFromCode121() {
        try {
            theGame.addGuestFromCode('W', 0, -1);
        } catch (Exception e)
        {
            assert false;
        }
    }
    @Test(expected = AssertionError.class)
    public void testAddGuestFromCode221() {
        try {
            theGame.addGuestFromCode('Q', 0, -1);
        } catch (Exception e)
        {
            assert false;
        }
    }
    @Test(expected = AssertionError.class)
    public void testAddGuestFromCode131() {
        try {
            theGame.addGuestFromCode('W', theGame.boardWidth(), -1)
    ;
        } catch (Exception e)
        {
            assert false;
        }
    }
    @Test(expected = AssertionError.class)
    public void testAddGuestFromCode231() {
        try {
            theGame.addGuestFromCode('Q', theGame.boardWidth(), -1)
    ;
        } catch (Exception e)
        {
            assert false;
```

```
64          }
65      }
66
67      @Test(expected = AssertionError.class)
68      public void testAddGuestFromCode112() {
69          try {
70              theGame.addGuestFromCode('W', -1, 0);
71          } catch (Exception e)
72          {
73              assert false;
74          }
75      }
76      @Test(expected = AssertionError.class)
77      public void testAddGuestFromCode212() {
78          try {
79              theGame.addGuestFromCode('Q', -1, 0);
80          } catch (Exception e)
81          {
82              assert false;
83          }
84      }
85      @Test(expected = AssertionError.class)
86      public void testAddGuestFromCode113() {
87          try {
88              theGame.addGuestFromCode('W', -1, theGame.boardHeight()
    );
89          } catch (Exception e)
90          {
91              assert false;
92          }
93      }
94      @Test(expected = AssertionError.class)
95      public void testAddGuestFromCode213() {
96          try {
97              theGame.addGuestFromCode('Q', -1, theGame.boardHeight()
    );
98          } catch (Exception e)
99          {
100             assert false;
101         }
102     }
103     @Test
104     public void testAddGuestFromCode122() {
105         theGame.addGuestFromCode('W', 0, 0);
106     }
107     @Test(expected = AssertionError.class)
108     public void testAddGuestFromCode222() {
109         try {
110             theGame.addGuestFromCode('Q', 0, 0);
111         } catch (Exception e)
112         {
113             assert false;
114         }
115     }
116     @Test(expected = AssertionError.class)
117     public void testAddGuestFromCode123() {
118         try {
```

```
119             theGame.addGuestFromCode('W', 0, theGame.boardHeight())
    ;
120         } catch (Exception e)
121         {
122             assert false;
123         }
124     }
125     @Test(expected = AssertionError.class)
126     public void testAddGuestFromCode223() {
127         try {
128             theGame.addGuestFromCode('Q', 0, theGame.boardHeight())
    ;
129         } catch (Exception e)
130         {
131             assert false;
132         }
133     }
134     @Test(expected = AssertionError.class)
135     public void testAddGuestFromCode132() {
136         try {
137             theGame.addGuestFromCode('W', theGame.boardWidth(), 0);
138         } catch (Exception e)
139         {
140             assert false;
141         }
142     }
143     @Test(expected = AssertionError.class)
144     public void testAddGuestFromCode232() {
145         try {
146             theGame.addGuestFromCode('Q', theGame.boardWidth(), 0);
147         } catch (Exception e)
148         {
149             assert false;
150         }
151     }
152     @Test(expected = AssertionError.class)
153     public void testAddGuestFromCode133() {
154         try {
155             theGame.addGuestFromCode('W', theGame.boardWidth(),
    theGame.boardHeight());
156         } catch (Exception e)
157         {
158             assert false;
159         }
160     }
161     @Test(expected = AssertionError.class)
162     public void testAddGuestFromCode233() {
163         try {
164             theGame.addGuestFromCode('Q', theGame.boardWidth(),
    theGame.boardHeight());
165         } catch (Exception e)
166         {
167             assert false;
168         }
169     }
```

Listing 2: "JUnit tests for addGuestFromCode."

1.8.5 Implement addGuestFromCode method. Then re-run the test suite. Inspect code coverage results, and explain your findings.

addGuestFromCode was already implemented so I compared the code coverage results without the additional tests and with. Without the 18 new test-cases the code coverage for the addGuestFromCode method was 72% for line coverage and 72% for branch coverage, after adding the tests it shot up to 81% for the line coverage and 77% for the branch coverage.

1.8.6 Would your test suite reveal all the faults you proposed in Exercise 1? If not, explain why the equivalence partitioning approach missed it, and add appropriate test cases separately to your JUnit implementation.

No it would not, for example the last point "Forgetting to check whether the cell is already occupied by a non-empty guest." is not covered by equivalence partition testing. For this I wrote a single test-case 3 using the simple GameTestCase map where (0,1) contains a wall, so trying to add a food-entity there should result in a fault. equivalence partitioning missed this because it requires some domain-specific knowledge about the function, specifically that the occupy method can only be used to occupy non-null empty cells.

```
1    @Test(expected = AssertionError.class)
2    public void testAddGuestFromCodeNonEmptyCell() {
3        try {
4            theGame.addGuestFromCode('F', 0, 1);
5        } catch (Exception e)
6        {
7            assert false;
8        }
9    }
```

Listing 3: "Additional JUnit test for addGuestFromCode."

1.9 Exercise 11

Imagine a more complex Board and an additional parameter that describes the shape of the Guest (like occupying multiple cells). How does an equivalence partitioning approach scale? - *Lars*

Lets imagine two scenarios one where the additional parameter is an enumerated value and one where the additional parameter is a tuple (x_size, y_size). For simplicity's sake, we will assume the equivalence partitions of x and y remain the same, meaning if a 2x2 square shape partially spills over a boundary, we will assign it to either the larger than or less than allowed partition.

1.9.1 Enumerated Value

In the enumerated value scenario we simply obtain two more equivalence partitions, one where the **parameter** $\in \{allowed\}$ and one where **parameter** $\notin \{allowed\}$. Under the single fault model this would give us an additional test-case, but under the multiple fault model we end up with twice the amount of test cases.

1.9.2 Tuple value

In this scenario, each element of the tuple will have 3 equivalence partitions, one where $tuple.value < 0$, one where $0 < tuple.value < height/width$ and one where $tuple.value > height/width$. Under the single fault model, this would give us 2 additional test cases, but under the multiple fault model, our amount of test-cases becomes nine times larger! Here we can easily see that this amount of test-cases is overkill for this method.