PROJECT ROBOT MOTION
Task 4: Regression Testing

COEN 6761 - Software Testing and Validation

Team Code Blooded

Rajat Rajat - 40160245
Sharul Dhiman - 40195730
Rohan Kodavalla - 40196377
Rishi Murugesan Gopalakrishnan - 40200594

Department of Electrical and Computer Engineering

Gina Cody School of Engineering and Computer Science

Winter 2023

Introduction

This project focuses on developing an application that simulates a robot which can move around an NxN floor. The robot has certain functionalities. The Robot can move around the floor based on the commands received from the user. The Robot holds the pen in two positions, up and down. When the pen is down the robot will trace the path when it moves around the floor and when the pen is up the robot moves freely without tracing anything. Initially, the robot's position is [0,0] of the floor, the robot's pen will be up and the robot will be facing north.

The following are the commands that the robot will respond to.

Command	Use
I n | i n	Initializes the n x n array floor, n >0. This command will also set the robot's initial position to (0,0) and will set the pen's position to up and the robot's direction as north.
U | u	Pen Up
D | d	Pen Down
R | r	Turn Right
L | l	Turn Left
M s | m s	This command is to make the robot move forward 's' spaces where 's' is a non-negative number.
P | p	Prints the NxN array and displays the indices in the console.
C | c	This command will print the current position of the pen and whether it is up or down and which direction it is facing.
Q | q	Terminate the program

Github URL

Requirements

The requirements for the robot simulation project are listed below, each with a unique identifier:

- Initialize the system (R1)

The program should be able to initialize the system with a specified size of the floor array, N. The array values should be set to zeros and the robot should be positioned at [0, 0], with the pen up, and facing north.

- Change pen position (R2)

The program should be able to change the position of the pen from up to down or from down to up.

- Turn the robot (R3)

The program should be able to turn the robot in either direction (left or right) depending on the user's command.

- Move the robot (R4)

The program should be able to move the robot forward a given number of spaces, s. The spaces should be non-negative integers. If the pen is down while the robot moves, the appropriate elements of the floor array should be set to 1.

- Print the floor (R5)

The program should be able to display the N-by-N array, where wherever there is a 1 in the array, an asterisk should be displayed; wherever there is a zero, a blank should be displayed.

- Current position (R6)

The program should be able to display the current state of the robot including the position of the pen and whether it is up or down and its facing direction.

- Display asterisks and blanks (R7)

The program must display the floor as the N*N array with the asterisk where the robot has traced and blanks where it has not.

- End program (R8)

The program should be able to stop running when the user gives the "Q" or "q" command.

Screenshots

```java
public void initializeArrayFloor(int n) {
    // Initializing the array
    N = n;
    if( N < 0 )
    {
        System.out.println("Invalid Number. Array Dimension should be a Positive Value");
    }
    else
    {
        floor = new int[N][N];
        penStatus = "up";
        //Initializing array values to 0
        for (int i = 0; i<N; i++)
        {
            for (int j = 0;j<N;j++)
            {
                floor[i][j]= 0;
            }
        }
        //Setting Robot Position
        x = y = 0;
        floor[x][y]=0;
        //Setting Robot Direction
        Direction = "north";
    }
}
```

Figure 1. Function for Initializing the Floor and Robot - R1

```java
public void displayFloor() {
    for(int i = N-1;i>=0;i--)
    {
        System.out.println();
        System.out.print(i+"  ");
        for (int j = 0;j<N;j++)
        {
            if(floor[j][i] == 1)
                System.out.print("*  ");
            else
                System.out.print("   ");
        }
        System.out.println();
    }
    System.out.println(" ");
    System.out.print("   ");
    for(int k = 0;k<N;k++)
    {
        System.out.print(k+"  ");
    }
}
```

Figure 2. Function for displaying the floor - R5 and R8

```java
public void currentPosition() {
    System.out.println("Position: "+x + "," + y + " - Pen: " + penStatus+" - Facing: "+Direction);
}
```

Figure 3. Function for displaying the current position of the Robot - R6

```java
public void penUp() {
    penStatus = "up";
}


public void penDown() {
    penStatus = "down";
}
```

Figure 4. Functions for changing the pen's position - R2

```java
public void turnRight() {
    if(Direction == "north")
    {
        Direction = "east";
    }
    else if(Direction == "south")
    {
        Direction = "west";
    }
    else if(Direction == "west")
    {
        Direction = "north";
    }
    else
    {
        Direction = "south";
    }
}
```

Figure 5. Function to make robot turn right - R3

```java
public void turnLeft() {
    if(Direction == "north")
    {
        Direction = "west";
    }
    else if(Direction == "south")
    {
        Direction = "east";
    }
    else if(Direction == "west")
    {
        Direction = "south";
    }
    else
    {
        Direction = "north";
    }
}
```

Figure 6. Function to make robot turn left - R3

```java
public void quit() {
    // Program Termination
    System.out.println("ROBOT MOTION TERMINATED");
    System.exit(0);
}
```

Figure 7. Function for terminating the program - R7

```java
public boolean moveForward(int S) {
    s = S;
    if((s<0 || s>=N) == true)
    {
        System.out.println("'s' should be a positive number and should be within the Floor");
        return false;
    }
    else
    {
        if(Direction == "north" && (y+s < N))
        {
            int m = y;
            y = y + s; // Setting new coordinates
            if(penStatus == "down")
            {
                for(int l=m;l<=y;l++)
                {
                    floor[x][l] = 1;
                }
            }
        }
        else if(Direction == "east" && (x+s < N))
        {
            int m = x;
            x = x + s;
            if(penStatus == "down")
            {
                for(int l=m;l<=x;l++)
                {
                    floor[l][y] = 1;
                }
            }
        }

            else if(Direction == "west" && (x-s > 0))
            {
                int m = x;
                x = x - s;
                if(penStatus == "down")
                {
                    for(int l=m;l<=y;l++)
                    {
                        floor[l][y] = 1;
                    }
                }
            }
            else if(Direction == "south" && (y-s > 0))
            {
                int m = y;
                y = y - s;
                if(penStatus == "down")
                {
                    for(int l=m;l<=y;l++)
                    {
                        floor[x][l] = 1;
                    }
                }
            }
            else {
                System.out.println("Robot Cannot leave the floor");
                return false;
            }
        }
        return true;
    }
}
```

Figure 8. Function for moving the robot - R4

Test Cases

- **Test Case ID** : 1
  Tester's Name : Rishi Murugesan Gopalakrishnan
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testTurnRight()
  Test Input Data: 'R' (or) 'r'
  Test Case Description : Program to test turning right functionality of the robot.
  Expected Output: The robot will turn right

```java
@Test
public void testTurnRight()
{
    robot.Direction = "north";
    robot.turnRight();
    assertEquals("east",robot.Direction);

    robot.Direction = "south";
    robot.turnRight();
    assertEquals("west",robot.Direction);

    robot.Direction = "west";
    robot.turnRight();
    assertEquals("north",robot.Direction);

    robot.Direction = "east";
    robot.turnRight();
    assertEquals("south",robot.Direction);
}
```

- **Test Case ID** : 2
  Tester's Name : Rishi Murugesan Gopalakrishnan
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testTurnLeft()
  Test Input Data:'L' (or) 'l'
  Test Case Description : Program to test turning left functionality of the robot.
  Expected Output: The Robot will turn to its left

```java
@Test
public void testTurnLeft()
{
    robot.Direction = "north";
    robot.turnLeft();
    assertEquals("west",robot.Direction);

    robot.Direction = "south";
    robot.turnLeft();
    assertEquals("east",robot.Direction);

    robot.Direction = "west";
    robot.turnLeft();
    assertEquals("south",robot.Direction);

    robot.Direction = "east";
    robot.turnLeft();
    assertEquals("north",robot.Direction);
}
```

- **Test Case ID** : 3
  Tester's Name : Rishi Murugesan
  Gopalakrishnan
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testFloorValues()
  Test Input Data: "I 5", 'd',"m 2",'r', "m 2"
  Test Case Description : Program to test the whole floor to check whether the robot is tracing the floor when the pen is down by assigning 1 and 0 in the array locations.
  Expected Output:
    - 1 in array locations to where the robot has moved
    - 0 where the robot has not moved

```java
@Test
public void testFloorValues() {
    robot.initializeArrayFloor(n);
    robot.penDown();
    robot.moveForward(s);
    robot.turnRight();
    robot.moveForward(s);

    assertEquals(1,robot.floor[0][0]);
    assertEquals(1,robot.floor[0][1]);
    assertEquals(1,robot.floor[0][2]);
    assertEquals(1,robot.floor[1][2]);
    assertEquals(1,robot.floor[2][2]);
    assertEquals(0,robot.floor[0][3]);
    assertEquals(0,robot.floor[0][4]);
    assertEquals(0,robot.floor[1][0]);
    assertEquals(1,robot.floor[1][2]);
    assertEquals(0,robot.floor[1][3]);
    assertEquals(0,robot.floor[1][4]);
    assertEquals(0,robot.floor[2][0]);
    assertEquals(0,robot.floor[2][1]);
    assertEquals(0,robot.floor[2][3]);
    assertEquals(0,robot.floor[2][4]);
    assertEquals(0,robot.floor[3][0]);
    assertEquals(0,robot.floor[3][1]);
    assertEquals(0,robot.floor[3][2]);
    assertEquals(0,robot.floor[3][3]);
    assertEquals(0,robot.floor[3][4]);
    assertEquals(0,robot.floor[4][0]);
    assertEquals(0,robot.floor[4][1]);
    assertEquals(0,robot.floor[4][3]);
    assertEquals(0,robot.floor[4][4]);
}
```

- **Test Case ID** : 4
  Tester's Name : Rishi Murugesan Gopalakrishnan
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testFloorOutput()
  Test Input Data: "I 5", 'd',"m 2",'r', "m 2"
  Test Case Description : Program to test whether * (asterix) and " "(a blank space) are printed
  Expected Output:

```
4

3

2   *   *   *

1   *

0   *

    0  1  2  3  4
```

```java
@Test
public void testFloorOutput() {
    robot.initializeArrayFloor(n);
    robot.penDown();
    robot.moveForward(s);
    robot.turnRight();
    robot.moveForward(s);
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    System.setOut(new PrintStream(output));

    robot.displayFloor();
    String expectedOutput = "4                \n" +
                            "\n3              \n" +
                            "\n2   *   *   *  \n" +
                            "\n1   *          \n" +
                            "\n0   *          \n \n" +
                            "   0  1  2  3  4";

    //assertEquals("\n4              \n3              \n2   *   *   *   \n1   *              \n0   *
    assertEquals(expectedOutput.trim().replaceAll("\\s",""),output.toString().trim().replaceAll("\\s",""));
}
```

- **Test Case ID** : 5
  Tester's Name : Rohan Kodavalla
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name :
  testPenDownAndTracing()
  Test Input Data: "I 5", 'd',"m 2"
  Test Case Description : Program to test whether
  the robot is tracing the floor when the pen is down.
  Expected Output: Position: 0,2 - Pen: down - Facing: north

```java
@Test
public void testPenDownAndTracing() {
    robot.initializeArrayFloor(n);
    robot.penDown();
    robot.moveForward(s);
    assertEquals(1,robot.floor[0][2]);
}
```

- **Test Case ID** : 6
  Tester's Name : Rohan Kodavalla
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name :
  testMovingOutOfBounds()
  Test Input Data: "I 5"
  Test Case Description : Program to test
  whether the robot is tracing only inside the floor when the pen is down and does not go out of
  boundaries.
  Expected Output: 's' should be a positive number and should be within the Floor (if False)

```java
@Test
public void testMovingOutOfBounds() {
    robot.initializeArrayFloor(n);
    robot.penDown();
    assertEquals(true, robot.moveForward(n-1)); /
    assertEquals(false, robot.moveForward(n)); //
}
```

- **Test Case ID** : 7
  Tester's Name : Rishi Murugesan
  Gopalakrishnan
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name :
  testCurrentPosition()
  Test Input Data: "I 5"
  Test Case Description : Program to
  test whether the program displays
  the current output of the robot
  Expected Output: Position: 0,0 - Pen: up - Facing: north

```java
@Test
public void testCurrentPosition() {
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    System.setOut(new PrintStream(output));

    robot.initializeArrayFloor(n);

    robot.currentPosition();
    assertEquals("Position: 0,0 - Pen: up - Facing: north", output.toString().trim());
}
```

- **Test Case ID** : 8
  Tester's Name : Rajat Rajat
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testPenUp()
  Test Input Data: "I 5", 'd', 'u'
  Test Case Description : Program to test
  whether the program changes pen's position to "up" when command 'u' is entered
  Expected Output: up

```java
@Test
public void testPenUp() {
    robot.penUp();
    assertEquals("up",robot.penStatus);
}
```

- **Test Case ID** : 9
  Tester's Name : Rajat Rajat
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testPenDown()
  Test Input Data: "I 5", 'd'
  Test Case Description : Program to test
  whether the program changes pen's position to "down" when command 'u' is entered
  Expected Output: down

```java
@Test
public void testPenDown() {
    robot.penDown();
    assertEquals("down",robot.penStatus);
}
```

- **Test Case ID** : 10
  Tester's Name : Sharul Dhiman
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name :
  testInitializeSystem()
  Test Input Data: "I 5"
  Test Case Description : Program to
  test whether the program initializes
  the floor and the robot when
  command i is used
  Expected Output: up, 0, Position: 0,0 - Pen: up - Facing: north

```java
@Test
public void testInitializeSystem() {
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    System.setOut(new PrintStream(output));

    robot.initializeArrayFloor(n);
    assertEquals("up",robot.penStatus);
    assertEquals(0, robot.floor[0][0]); //return true if y,x values = [x],[y]
    robot.currentPosition();
    assertEquals("Position: 0,0 - Pen: up - Facing: north", output.toString().trim());
}
```

- **Test Case ID** : 11
  Tester's Name : Sharul Dhiman
  Date : 02/11/2023
  Test Type : Unit Testing
  Test Function Name : testMoveForward()
  Test Input Data: "I 5", 'd',"m 2"
  Test Case Description : Program to test whether the move forward functionality works perfectly
  Expected Output: Position: 0,2 - Pen: down - Facing: north , 1

```java
@Test
public void testMoveForward() {
    ByteArrayOutputStream output = new ByteArrayOutputStream();
    System.setOut(new PrintStream(output));

    robot.initializeArrayFloor(n);
    robot.penDown();
    robot.moveForward(s);
    robot.currentPosition();
    assertEquals("Position: 0,2 - Pen: down - Facing: north", output.toString().trim());
    assertEquals(1,robot.floor[0][2]);
}
```

Below is the table mapping requirements and unit test cases

Requirement ID	Requirement	Test Case Scenario	Unit Test Case Function	Execution Result (Pass/Fail)
R1	Initialize the system	Test that the floor array is correctly initialized to zeros, that the robot is positioned at [0, 0], with the pen up, and facing north.	Test Case ID : 10	Pass
R2	Change pen position	Test that the pen can be changed from up to down and from down to up.	Test Case ID : 8 Test Case ID : 9	Pass
R3	Turn the robot	Test that the robot can be turned in either direction (left or right) depending on the user's command.	Test Case ID : 1 Test Case ID : 2	Pass
R4	Move the robot	Test that the robot can move forward a given	Test Case ID : 11	Pass

|		number of spaces and that the appropriate elements of the floor array are set to 1 when the pen is down while the robot moves.		
R5	Print the floor	Test that the N-by-N array is displayed correctly with asterisks where there is a 1 in the array and blanks where there is a zero.	Test Case ID : 3 Test Case ID : 5	Pass
R6	Display current position	Test that the current position of the pen, whether it is up or down, and its facing direction can be displayed correctly.	Test Case ID : 7	Pass
R7	Display asterisks and blanks	Test that the program can display * and blank space to show the path that the robot has traced	Test Case ID : 4	Pass
R8	End program	Test that the program stops running when the user gives the "Q" or "q" command.	Test Case ID : 12	Pass
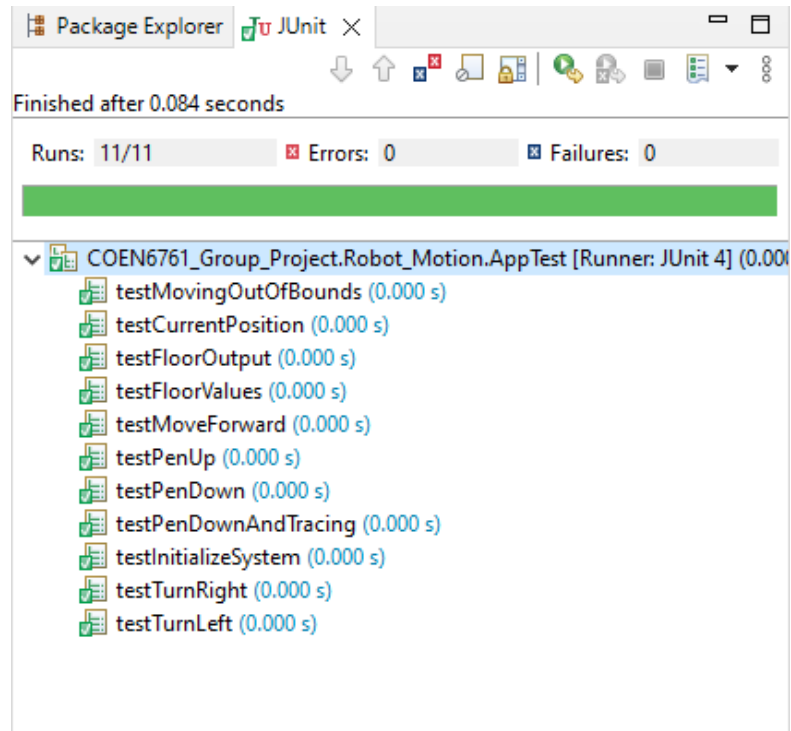
Figure 9. JUnit Test Result Window

Code Analysis

Code coverage is a metric used to measure percentage of code lines, statements, methods, branches or conditions which are executed during testing of an application. It indicates how much of the code has been tested by the test cases and can also help identify parts of the code that weren't tested properly.

Usually code coverage is measured using a code coverage tool that commonly comes with the IDE. Code coverage tools track the lines of code that are executed during the test run of test cases. The results from these tools are typically in percentages.
Different type of code coverage can be measured such as
- Methods or Functions
- Statements
- Paths (If statements or Loops)
- Conditions (Boolean Expressions)
- Number of lines

Based on our code complexity and the functionalities covered, we have decided to set our code coverage threshold to be 90%.

Functional Coverage

Functional coverage determines how much of the code has been executed by tests that were designed to test specific functionalities of the application.

Functions in Robot.java	Test cases that covers functions	Function Coverage Percentage
command ( )	NA	0 %
initializeArrayFloor ( )	testInitializeSystem ( )	100 %
displayFloor ( )	testFloorOutput ( )	100 %
currentPosition ( )	testCurrentPosition ( )	100 %
penUp ( )	testPenUp ( )	100 %
penDown ( )	testPenDown ( )	100 %
turnRight ( )	testTurnRight ( )	100 %

turnLeft ( )	testTurnLeft ( )	100 %
moveForward ( )	testMoveForwardPenDown( ) testMoveForwardPenUp ( )	100 %
quit ( )	NA	0 %
TOTAL		**90.9 %**

Statement Coverage

Statement coverage is a specific type of code coverage that measures the proportion of executable statements in a program that are executed during the execution of a test suite.

In order to calculate statement coverage, the code is instrumented with special instructions that count the number of times each statement is executed during the running of a test suite. After the test suite has been executed, the coverage tool calculates the percentage of statements that have been executed at least once.It helps to assess the thoroughness of the testing process by indicating which parts of the code have been exercised and which parts have not.

Robot.java methods	No. of Statements in each method	No. of Statements covered by test cases	Statement Coverage Percentage
command ( )	67	0	0 %
moveForward ( )	191	182	95.3 %
currentPosition ( )	26	26	100 %
displayFloor ( )	71	71	100 %
initializeArrayFloor()	62	62	100 %
penDown ( )	4	4	100 %
penUp ( )	4	4	100 %
turnLeft ( )	28	28	100 %
turnRight ( )	28	28	100 %
quit ( )	2	2	100 %
TOTAL	**492**	**76**	**84.6 %**

Path Coverage

In order to measure the proportion of all possible execution paths we use path coverage, which is a type of coverage metrics. All the possible paths in the program have been executed during testing. To ensure that code behaves as expected in all possible situations we have to identify and test all possible execution paths.

All these execution paths are identified and tested through the software code which includes conditional statements, exceptional handling code and loops. Path coverage gives more detailed assessment towards the quality of the code as compared with simpler metrics such as line coverage or statement coverage. It identifies potentially hidden bugs by ensuring that all possible paths are executed and tested.

Robot.java methods	No. of Paths	Test Case covering each Path	Path Coverage Percentage
command( )	9	9	100 %
moveForward ( )	18	18	100 %
currentPosition ( )	0	0	NA
displayFloor ( )	8	8	100 %
initializeArrayFloor()	6	6	100 %
penDown ( )	0	0	NA
penUp ( )	0	0	NA
turnLeft ( )	4	4	100 %
turnRight ( )	4	4	100 %
quit ( )	0	0	NA

Condition Coverage

Condition coverage ensures that every possible condition in the code is evaluated at least once. The aim of this coverage metric is to detect the errors caused by complex logical expressions. It is a useful measure in detecting flaws in decision-making statements such as if-else statements and switch cases.
Below is the table defining the condition coverage :

Robot.java methods	Conditional Statements Type	Number of Conditional statements	Test Case covering Conditional statements	Conditional Coverage Percentage
command()	N/A	NA	NA	NA
initializeArrayFloor()	if-else, for	3	testInitializeSystem(), testFloorValues(), testFloorOutput(), testPenDownAndTracing(), testMovingOutOfBounds(), testCurrentPosition(), testMoveForwardPenDown() testMoveForwardPenUp()	100%
displayFloor()	for, if-else	4	testFloorOutput()	100%
currentPosition()	N/A	0	testInitializeSystem(), testCurrentPosition(), testMoveForwardPenDown() testMoveForwardPenUp()	N/A
penUp()	N/A	0	testPenUp(), testMoveForwardPenUp()	N/A
penDown()	N/A	0	testPenDown(), testMoveForwardPenDown() testFloorValues(), testFloorOutput(), testPenDownAndTraacing(), testMovingOutOfBounds()	N/A
turnRight()	if-else if - else	3	testTurnRight(), testFloorValues(), testFloorOutput(), testMoveForwardPenDown()	100%

			testMoveForwardPenUp()	
turnLeft()	if-else if - else	3	testTurnLeftt(), testMoveForwardPenUp(), testMoveForwardPenDown()	100%
moveForward()	if- else if- else, for	13	testFloorOutput(), testFloorValues(), testPenDownAndTracing(), testMoveForwardPenUp(), testMoveForwardPenDown()	100%
quit()	N/A	NA	NA	NA

Therefore, from above table the condition coverage for the Robot.java code is 100% , meaning every possible condition in the code is evaluated at least once.

Line Coverage

No. of Lines in Robot.java	No. of Statements covered by test cases	Statement Coverage Percentage
120	91	75.8 %

Results from EclEmma in Eclipse

Instruction Counter

Element	Coverage		Covered Instructio...	Missed Instructions	Total Instructions
∨ 🟦 Robot.java	▮▮▮	84.6 %	416	76	492
∨ 🟢 Robot	▮▮▮	84.6 %	416	76	492
🟢 command()	▮	0.0 %	0	67	67
🟢 moveForward(int)	▮▮▮	95.3 %	182	9	191
🟢 currentPosition()	▮	100.0 %	26	0	26
🟢 displayFloor()	▮	100.0 %	71	0	71
🟢 initializeArrayFloor(int)	▮	100.0 %	62	0	62
🟢 penDown()		100.0 %	4	0	4
🟢 penUp()		100.0 %	4	0	4
🟢 quit()		100.0 %	2	0	2
🟢 turnLeft()	▮	100.0 %	28	0	28
🟢 turnRight()	▮	100.0 %	28	0	28

Branch Counter

Element	Coverage		Covered Branches	Missed Branches	Total Branches
∨ 🟦 Robot.java	▮▮▮	84.7 %	61	11	72
∨ 🟢 Robot	▮▮▮	84.7 %	61	11	72
🟢 command()	▮	0.0 %	0	10	10
🟢 moveForward(int)	▮▮▮	97.2 %	35	1	36
🟢 currentPosition()	▮		0	0	0
🟢 displayFloor()	▮	100.0 %	8	0	8
🟢 initializeArrayFloor(int)	▮	100.0 %	6	0	6
🟢 penDown()			0	0	0
🟢 penUp()			0	0	0
🟢 quit()			0	0	0
🟢 turnLeft()	▮	100.0 %	6	0	6
🟢 turnRight()	▮	100.0 %	6	0	6

Line Counter

Element	Coverage	Covered Lines	Missed Lines	Total Lines
Robot.java	75.8 %	91	29	120
Robot	75.8 %	91	29	120
command()	0.0 %	0	28	28
moveForward(int)	97.1 %	34	1	35
currentPosition()	100.0 %	2	0	2
displayFloor()	100.0 %	13	0	13
initializeArrayFloor(int)	100.0 %	13	0	13
penDown()	100.0 %	2	0	2
penUp()	100.0 %	2	0	2
quit()	100.0 %	1	0	1
turnLeft()	100.0 %	11	0	11
turnRight()	100.0 %	11	0	11

Method Counter

Element	Coverage	Covered Methods	Missed Methods	Total Methods
Robot.java	90.9 %	10	1	11
Robot	90.9 %	10	1	11
command()	0.0 %	0	1	1
currentPosition()	100.0 %	1	0	1
displayFloor()	100.0 %	1	0	1
initializeArrayFloor(int)	100.0 %	1	0	1
moveForward(int)	100.0 %	1	0	1
penDown()	100.0 %	1	0	1
penUp()	100.0 %	1	0	1
quit()	100.0 %	1	0	1
turnLeft()	100.0 %	1	0	1
turnRight()	100.0 %	1	0	1

Type Counter

Element	Coverage	Covered Types	Missed Types	Total Types
Robot.java	100.0 %	1	0	1
Robot	100.0 %	1	0	1
command()		0	0	0
currentPosition()		0	0	0
displayFloor()		0	0	0
initializeArrayFloor(int)		0	0	0
moveForward(int)		0	0	0
penDown()		0	0	0
penUp()		0	0	0
quit()		0	0	0
turnLeft()		0	0	0
turnRight()		0	0	0

Complexity

Element	Coverage	Covered Complexity	Missed Complexity	Total Complexity
∨ J Robot.java	78.4 %	40	11	51
∨ G Robot	78.4 %	40	11	51
● command()	0.0 %	0	10	10
● moveForward(int)	94.7 %	18	1	19
● currentPosition()	100.0 %	1	0	1
● displayFloor()	100.0 %	5	0	5
● initializeArrayFloor(int)	100.0 %	4	0	4
● penDown()	100.0 %	1	0	1
● penUp()	100.0 %	1	0	1
● quit()	100.0 %	1	0	1
● turnLeft()	100.0 %	4	0	4
● turnRight()	100.0 %	4	0	4

Discussion of the code coverage results

In terms of threshold values for code coverage, different organizations and projects may have different requirements or expectations for code coverage, based on factors such as the criticality of the code, the complexity of the system, and the level of risk tolerance.

Some organizations may have a specific minimum threshold that all code must meet before it can be released, while others may have different thresholds for different parts of the codebase. For example, some organizations may require 100% code coverage for critical or safety-critical code, while accepting lower levels of coverage for less critical code.

Decision making for releasing

After considering the complexity of our code and the range of functionalities it covers, we chose to establish a code coverage threshold of 90% and most of our code coverage metrics have surpassed this value.
We have covered every possible scenario and we have decided to release code based on code coverage results. It is important to note that we have consider not only the coverage percentage but also the quality of the tests cases and the specific areas of the code that are covered. In addition, we have also considered additional testing that includes manual testing and code review.

Software Release

Software Release v1.1.0

Description

This is the first software release for the Robot Motion program, version 1.0.0. The program allows users to move a robot on a 2D floor, with the option to set the pen up or down and draw on the floor. Users can initialize the floor dimensions, move the robot, turn it left or right, and print the floor to the console.

Features

- Initialize the floor with non-negative dimensions
- Move the robot forward
- Turn the robot left or right
- Set the pen up or down
- Print the current position of the pen and whether it is up or down and whether it is facing up or down
- Print the N x N array and also display indices
- Stop the program

This release includes the following changes

- Added appropriate Javadoc comments to each method to improve code readability
- Improved error handling by adding appropriate error messages
- Improved the displayFloor() method to display the indices of the array
- Improved the initialization of the array to handle negative input values
- Improved the formatting of the output in the currentPosition() method

Usage

Users can enter commands in the command prompt to interact with the robot.

Commands:
I n/i n: Initialize the system, system is reset, n is an non-negative integer
M s/ m s: Move forward s spaces - M is Char and s is a non-negative number
R/r: Turn Right - Char
L/l: Turn Left - Char
U/u: Pen Up - Char
D/d: Pen Down - Char
P/p: Print the N x N array and also display indices - Char
C/c: Print current position of the pen and whether it is up or down and whether it is facing up or down - Char
Q/q: Stop the program - Char

Improvements

Future software releases may include:
Improved error handling and input validation
Additional functionality, such as saving and loading floors to and from files
A graphical interface to allow users to interact with the robot visually

Known Issues

There are no known issues with this release.

Regression Testing

Adding New Feature

According to the project requirement, the development team was asked to develop a new feature for Task 4.

Feature: Replay all the commands entered by the user as a history
Command : H | h

This function was implemented in the already existing **Robot.java** using the following steps:

- Declaring a list to store all the commands

```java
List<String> commandHistory = new ArrayList<>();
```

- Adding the command while the corresponding command function is called

```java
public void initializeArrayFloor(int n) {
    commandHistory.add("Initialize Array "+n);


public void displayFloor() {
    commandHistory.add("Display Floor");
```

- Similarly, command was added to commandHistory list for every function.
- Now to display the history of all commands, we created a function called **history ( )**

```java
public void history() {
    commandHistory.add("Show Command History");

    System.out.println("COMMAND HISTORY");
    for(String command:commandHistory)
    {
        System.out.println(command);
    }
}
```

- When a series of commands, for example " i 5" , "d" , "c " , "h" we'll get the following output

```
Please enter your command
h
COMMAND HISTORY
Initialize Array 5
Pen Down
Current Position of the Robot
Show Command History
```

Regression Testing for New Feature

The following test case was written and added to the already existing test suite to test the new feature.

Test Case ID	Test Case Scenario	Test Function Name	Execution Result
Test Case ID :14	Executing a series of commands and then invoking "H" command to print the command history	testCommandHistory ( )	Pass

Reviewing Test Report Performed by QA Team

QA Team : Dev Team 8
Members of QA Team :
Varun Puri
Shubham Kumar
Abhishek Madhu
Krishna Chaitanya Koganti

The QA team performed the following testing methods to our Robot.java program.
- Black-Box Testing
- White-Box Testing to achieve a certain coverage
- Mutation Testing
- Data Flow Testing

Inferences from QA Report

After analyzing the QA report we have inferred the following,

Black-Box Testing

The QA team wrote 5 test cases for black box testing. According to the screenshots and testing code they have attached they are:

1. ErrorTest ( )
2. UpTest ( )
3. DownTest ( )
4. StartSystemTest ( )
5. TerminationTest ( )

According to their test execution screenshot, all the test cases passed.We imported their Black Box Testing code, we ran it in our testing suite and obtained the same.
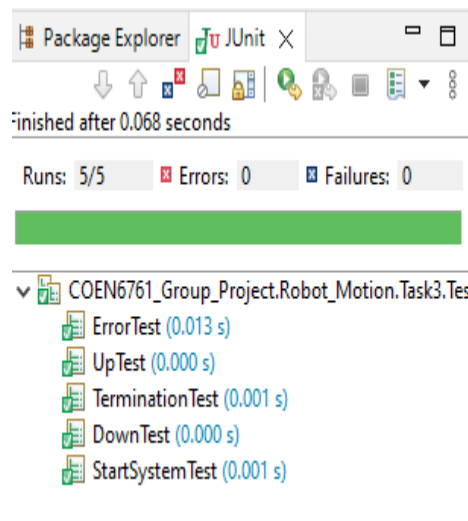
b) **Test report**

S.No	Function	Result
1	Matrix size	PASS
2	Pen Position	PASS
3	Grid boundary	PASS
4	Q command	PASS
5	Revert Back	FAIL

But in their Test report table, the QA team has marked **fail** for a test case named **Revert Back.** There is **no** test case with the name Revert Back and none of the test cases failed in the black box testing suite they provided.

White-Box Testing

The QA team wrote 8 test cases for white box testing. According to the screenshots and testing code they have attached they are:

1. testPenUp ( )
2. LeftTurningTest ( )
3. PenDownTest ( )
4. RightTurningTest ( )
5. PositionTest ( )
6. FloorOutputTest ( )
7. OutofBoundTest ( )
8. TracingTest ( )

According to their test execution screenshot, all the test cases passed.We imported their White Box Testing code, we ran it in our testing suite and obtained the same.
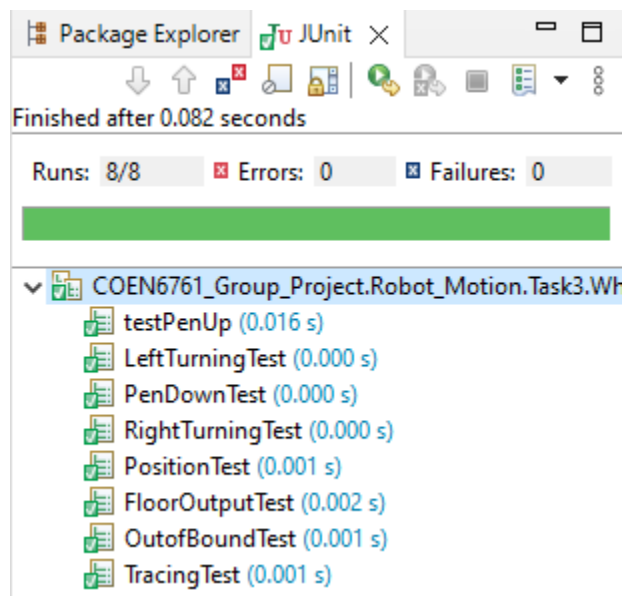
They have also performed coverage analysis from the white-box testing code and obtained the following coverage results

Metrics	Total	Missed	Percentage Covered
Branch Coverage	74	31	58%
Path Coverage	53	24	54.71%
Statement Coverage	478	161	66%
Line Coverage	116	47	59.48%
Function Coverage	12	2	83.33%

Mutation Testing

From their test suite, the QA team has performed mutation testing using PIT Mutation testing tool. Their test suite has achieved the following percentages:
- Line Coverage - 74 %
- Mutation Coverage - 69 %
- Test Strength - 79 %

Data Flow Testing

The QA team has not mentioned to which method they have performed the data flow testing.They have created C-Use P-Use table and have derived DCU and DPU table too. From these two tables we assume that they are testing the moveForward ( ) function.

The QA team wrote 2 test cases for data flow testing. According to the screenshots and testing code they have attached they are:
1. testInitializeArrayFloorWithValidInput ( )
2. testInitializeArrayFloorWithNegativeInput ( )

According to their test execution screenshot, all the test cases passed.We imported their Data Flow Testing code, we ran it in our testing suite and obtained the same.
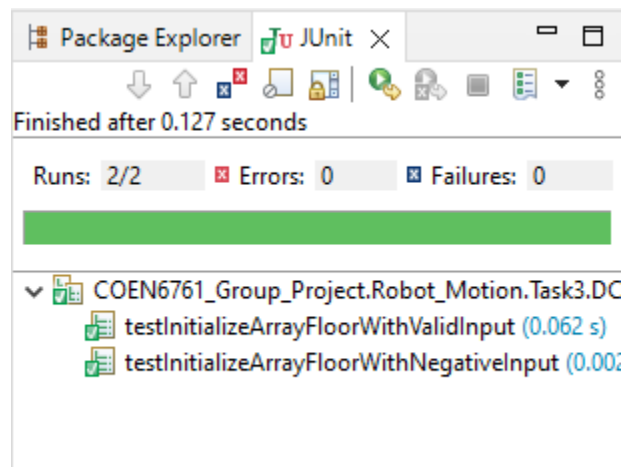
From the testing results we have inferred that all the functionalities developed by us in Robot.java are working fine. Hence we have not made any modification in our application code.

Hence we have added only the new test case created to test the command history feature.

We have executed the new test suite and have attached the execution screenshot below.
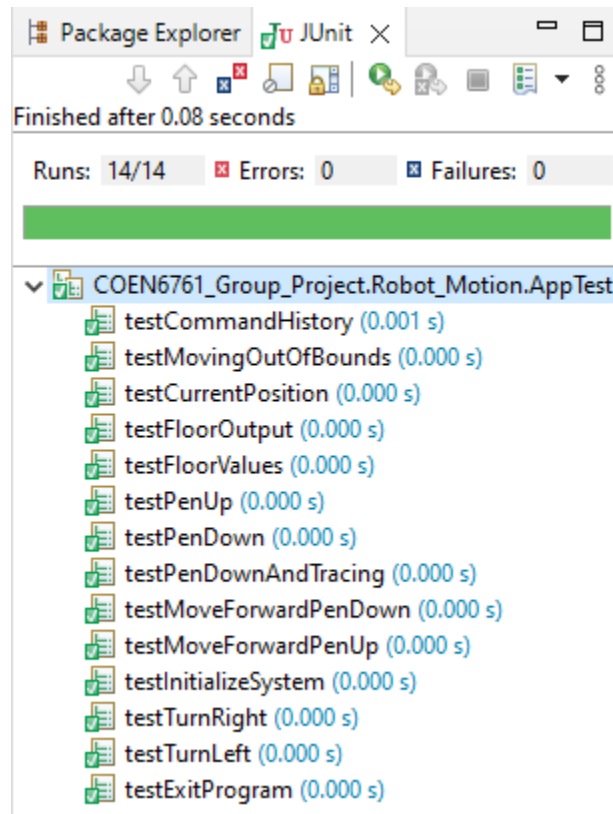
Code Coverage for the New Test Suite

Instruction Coverage

Branch Coverage

AppTest (1) (16-Apr-2023 6:28:36 pm)

Element	Coverage	Covered Branches	Missed Branches	Total Branches
∨ 🗁 src/main/java	84.0 %	63	12	75
∨ ⊞ COEN6761_Group_Project.Robot_Motion	84.0 %	63	12	75
› 🗋 App.java		0	0	0
∨ 🗋 Robot.java	84.0 %	63	12	75
∨ ⒼRobot	84.0 %	63	12	75
● command()	0.0 %	0	11	11
● currentPosition()		0	0	0
● displayFloor()	100.0 %	8	0	8
● history()	100.0 %	2	0	2
● initializeArrayFloor(int)	100.0 %	6	0	6
● moveForward(int)	97.2 %	35	1	36
● penDown()		0	0	0
● penUp()		0	0	0
● quit()		0	0	0
● turnLeft()	100.0 %	6	0	6
● turnRight()	100.0 %	6	0	6

Line Coverage

AppTest (1) (16-Apr-2023 6:28:36 pm)

Element	Coverage	Covered Lines	Missed Lines	Total Lines
∨ 🗁 src/main/java	75.0 %	105	35	140
∨ ⊞ COEN6761_Group_Project.Robot_Motion	75.0 %	105	35	140
› 🗋 App.java	0.0 %	0	4	4
∨ 🗋 Robot.java	77.2 %	105	31	136
∨ ⒼRobot	77.2 %	105	31	136
● command()	0.0 %	0	30	30
● currentPosition()	100.0 %	3	0	3
● displayFloor()	100.0 %	14	0	14
● history()	100.0 %	5	0	5
● initializeArrayFloor(int)	100.0 %	14	0	14
● moveForward(int)	97.2 %	35	1	36
● penDown()	100.0 %	3	0	3
● penUp()	100.0 %	3	0	3
● quit()	100.0 %	1	0	1
● turnLeft()	100.0 %	12	0	12
● turnRight()	100.0 %	12	0	12

Method Coverage

AppTest (1) (16-Apr-2023 6:28:36 pm)

Element	Coverage	Covered Methods	Missed Methods	Total Methods
∨ 🗁 src/main/java	78.6 %	11	3	14
∨ ⊞ COEN6761_Group_Project.Robot_Motion	78.6 %	11	3	14
› 🗋 App.java	0.0 %	0	2	2
∨ 🗋 Robot.java	91.7 %	11	1	12
∨ ⒼRobot	91.7 %	11	1	12
● command()	0.0 %	0	1	1
● currentPosition()	100.0 %	1	0	1
● displayFloor()	100.0 %	1	0	1
● history()	100.0 %	1	0	1
● initializeArrayFloor(int)	100.0 %	1	0	1
● moveForward(int)	100.0 %	1	0	1
● penDown()	100.0 %	1	0	1
● penUp()	100.0 %	1	0	1
● quit()	100.0 %	1	0	1
● turnLeft()	100.0 %	1	0	1
● turnRight()	100.0 %	1	0	1

Complexity Coverage

Element	Coverage	Covered Complexity	Missed Complexity	Total Complexity
∨ 🗁 src/main/java	75.0 %	42	14	56
∨ ⊞ COEN6761_Group_Project.Robot_Motion	75.0 %	42	14	56
› Ｊ App.java	0.0 %	0	2	2
∨ Ｊ Robot.java	77.8 %	42	12	54
∨ ⊕ Robot	77.8 %	42	12	54
● command()	0.0 %	0	11	11
● currentPosition()	100.0 %	1	0	1
● displayFloor()	100.0 %	5	0	5
● history()	100.0 %	2	0	2
● initializeArrayFloor(int)	100.0 %	4	0	4
● moveForward(int)	94.7 %	18	1	19
● penDown()	100.0 %	1	0	1
● penUp()	100.0 %	1	0	1
● quit()	100.0 %	1	0	1
● turnLeft()	100.0 %	4	0	4
● turnRight()	100.0 %	4	0	4

Conclusion

In conclusion, In Task 1 we have programmed the robot simulation project that requires the development of a Java program that can simulate the movement of a robot as it moves through a floor represented by an N-by-N array. The program should be able to initialize the system, change the pen position, turn the robot, move the robot, display the floor, display the current position of the robot and pen, and end the program. These requirements provide a comprehensive outline of the functionality expected in the final product. In Task 2, our code coverage results have provided valuable information about the quality of the robot motion program and the potential risk of releasing code with the attained code coverage. In Task 3, we tested the robot application developed by Dev Team 6 and in Task 4 we added new features to our application and also analyzed the QA report created on our program and performed regression testing on our robot application.