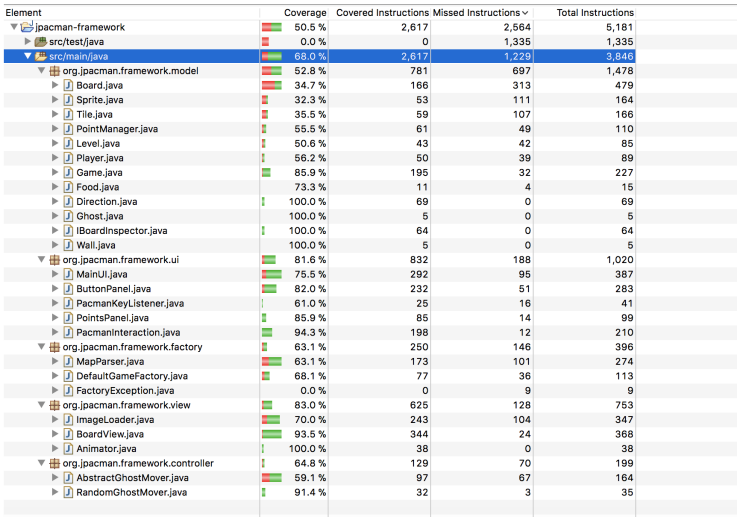
CPEN522 - Group22 - Assignment2

Yonghan Zhou 93506160 Yuzhan Yang 85929131

February 2020

Exercise 2.1

We chose scenario 1 'Analyse the start/stop/exit function of JPacman' in exploratory testing in Assignment 1. The coverage view is as follows:

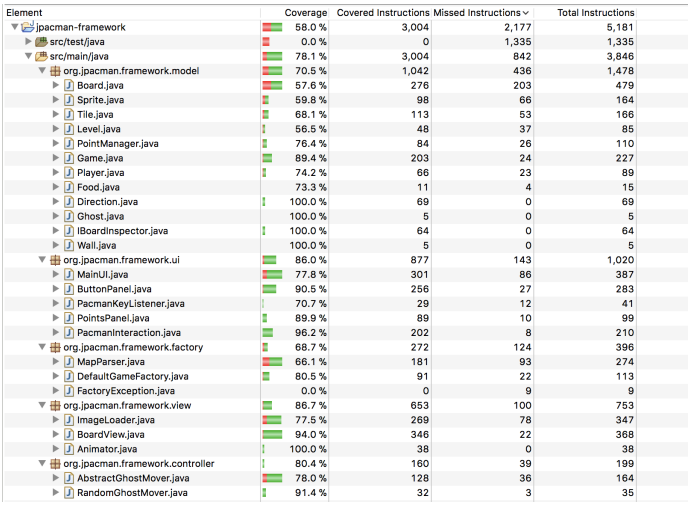


Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	50.5 %	2,617	2,564	5,181
src/test/java	0.0 %	0	1,335	1,335
src/main/java	68.0 %	2,617	1,229	3,846
org.jpacman.framework.model	52.8 %	781	697	1,478
Board.java	34.7 %	166	313	479
Sprite.java	32.3 %	53	111	164
Tile.java	35.5 %	59	107	166
PointManager.java	55.5 %	61	49	110
Level.java	50.6 %	43	42	85
Player.java	56.2 %	50	39	89
Game.java	85.9 %	195	32	227
Food.java	73.3 %	11	4	15
Direction.java	100.0 %	69	0	69
Ghost.java	100.0 %	5	0	5
IBoardInspector.java	100.0 %	64	0	64
Wall.java	100.0 %	5	0	5
org.jpacman.framework.ui	81.6 %	832	188	1,020
MainUI.java	75.5 %	292	95	387
ButtonPanel.java	82.0 %	232	51	283
PacmanKeyListener.java	61.0 %	25	16	41
PointsPanel.java	85.9 %	85	14	99
PacmanInteraction.java	94.3 %	198	12	210
org.jpacman.framework.factory	63.1 %	250	146	396
MapParser.java	63.1 %	173	101	274
DefaultGameFactory.java	68.1 %	77	36	113
FactoryException.java	0.0 %	0	9	9
org.jpacman.framework.view	83.0 %	625	128	753
ImageLoader.java	70.0 %	243	104	347
BoardView.java	93.5 %	344	24	368
Animator.java	100.0 %	38	0	38
org.jpacman.framework.controller	64.8 %	129	70	199
AbstractGhostMover.java	59.1 %	97	67	164
RandomGhostMover.java	91.4 %	32	3	35

Interesting findings:
1. PointManager.java : 55.5% code coverage. It is interesting to see that the method "invariant()" is not covered and in red (all branches missed). And assertions that call the method "invariant()" are in yellow (some branches missed).
2. FactoryException.java: 0% code coverage. It is reasonable since there is no exception raised during the runtime.
3. There are many variable/constant declarations that are not labeled in any color. For example, in "DefaultGameFactory" class, it declares a game reference without any color. That is because only when objects or values assigned to the reference, it will be counted in term of coverage. Declarations are not counted in coverage.

Exercise 2.2

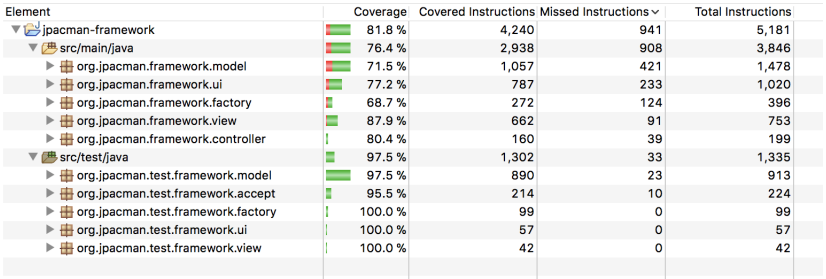
The coverage of application code increases from 68% to 78.1%. The coverage of PointManager class rises from 55.5% to 76.4%. The reason for the rise of coverage is that all assertions are covered.



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	58.0 %	3,004	2,177	5,181
src/test/java	0.0 %	0	1,335	1,335
src/main/java	78.1 %	3,004	842	3,846
org.jpacman.framework.model	70.5 %	1,042	436	1,478
Board.java	57.6 %	276	203	479
Sprite.java	59.8 %	98	66	164
Tile.java	68.1 %	113	53	166
Level.java	56.5 %	48	37	85
PointManager.java	76.4 %	84	26	110
Game.java	89.4 %	203	24	227
Player.java	74.2 %	66	23	89
Food.java	73.3 %	11	4	15
Direction.java	100.0 %	69	0	69
Ghost.java	100.0 %	5	0	5
IBoardInspector.java	100.0 %	64	0	64
Wall.java	100.0 %	5	0	5
org.jpacman.framework.ui	86.0 %	877	143	1,020
MainUI.java	77.8 %	301	86	387
ButtonPanel.java	90.5 %	256	27	283
PacmanKeyListener.java	70.7 %	29	12	41
PointsPanel.java	89.9 %	89	10	99
PacmanInteraction.java	96.2 %	202	8	210
org.jpacman.framework.factory	68.7 %	272	124	396
MapParser.java	66.1 %	181	93	274
DefaultGameFactory.java	80.5 %	91	22	113
FactoryException.java	0.0 %	0	9	9
org.jpacman.framework.view	86.7 %	653	100	753
ImageLoader.java	77.5 %	269	78	347
BoardView.java	94.0 %	346	22	368
Animator.java	100.0 %	38	0	38
org.jpacman.framework.controller	80.4 %	160	39	199
AbstractGhostMover.java	78.0 %	128	36	164
RandomGhostMover.java	91.4 %	32	3	35

Exercise 2.3

The whole coverage is 81.8%. The coverage for the application code is 76.4%. The coverage for test cases is 97.5%. The best representative of the test suite coverage is the application code coverage. Because a high application code coverage will show how well your test cases go through the application.



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	81.8 %	4,240	941	5,181
src/main/java	76.4 %	2,938	908	3,846
org.jpacman.framework.model	71.5 %	1,057	421	1,478
org.jpacman.framework.ui	77.2 %	787	233	1,020
org.jpacman.framework.factory	68.7 %	272	124	396
org.jpacman.framework.view	87.9 %	662	91	753
org.jpacman.framework.controller	80.4 %	160	39	199
src/test/java	97.5 %	1,302	33	1,335
org.jpacman.test.framework.model	97.5 %	890	23	913
org.jpacman.test.framework.accept	95.5 %	214	10	224
org.jpacman.test.framework.factory	100.0 %	99	0	99
org.jpacman.test.framework.ui	100.0 %	57	0	57
org.jpacman.test.framework.view	100.0 %	42	0	42

Exercise 2.4

A new test class BoardTest was created. The basic functionality of all public class methods was tested. Additionally, some potential edge cases like when invalid input is provided and some special cases were considered. The coverage of Board class could reach 80.6% with the test class BoardTest.

```java
package org.jpacman.test.framework.model;

import static org.junit.Assert.*;

public class BoardTest {

    private Board board;
    private int x, y;
    private Sprite sprite;



    private static final int WIDTH = 10;
    private static final int HEIGHT = 20;

    /**
     * Initial a board with WIDTH and HEIGHT
     */
    @Before
    public void setUp() {
        board = new Board(WIDTH, HEIGHT);
        x = y = 0;
        sprite = new Sprite() { };
    }


    /**
     * Test the constructor with negative width and height
     */
    @Test(expected = AssertionError.class)
    public void testConstructor() {
        new Board(-10, -20);
    }



    /**
     * Test the getWidth() method
     */
    @Test
    public void testGetWidth() {
        assertEquals(WIDTH, board.getWidth());
    }
```

```
54
55
56
57      /**
58       * Test the getHeight() method
59       */
60      @Test
61      public void testGetHeight() {
62          assertEquals(HEIGHT, board.getHeight());
63      }
64
65
66
67      /**
68       * Test the put() method by putting a sprite on tile (x,y)
    within the board
69       */
70      @Test
71      public void testPutWithinBoard() {
72          board.put(sprite, x, y);
73          assertEquals(board.tileAt(x, y).topSprite(), sprite);
74      }
75
76      /**
77       * Test the put() method by putting a sprite on tile out of
    the  board
78       */
79      @Test(expected = AssertionError.class)
80      public void testPutOutOfBoard() {
81          board.put(sprite, 20, 20);
82      }
83
84      /**
85       * Test the put() method by putting a sprite = null on tile
    (x,y)
86       */
87      @Test
88      public void testPutNull() {
89          Sprite sprite = null;
90          AssertionError e = assertThrows(AssertionError.class,
    ()->board.put(sprite, x, y));
91          assertEquals("PRE2: Sprite not null", e.getMessage());
92      }
```

```
 93
 94     /**
 95      * Test the put() method by putting a sprite that is already
    on tile
 96      */
 97     @Test
 98     public void testPutOccupied() {
 99         board.put(sprite, x, y);
100         AssertionError e = assertThrows(AssertionError.class,
    ()->board.put(sprite, 1, 1));
101         assertEquals("PRE3: Sprite should not occupy" +
    sprite.getTile(), e.getMessage());
102     }
103
104
105
106     /**
107      * Test the withinBorders() method
108      */
109     @Test
110     public void testWithinBorders() {
111         assertFalse(board.withinBorders(WIDTH, HEIGHT));
112     }
113
114
115
116     /**
117      * Test the spriteAt() method within board
118      */
119     @Test
120     public void testSpriteAt() {
121         board.put(sprite, x, y);
122         assertEquals(sprite, board.spriteAt(x, y));
123     }
124
125     /**
126      * Test the spriteAt() method out of board
127      */
128     @Test(expected = AssertionError.class)
129     public void testSpriteAtOutOfBoard() {
130         board.spriteAt(20, 20);
131     }
132
```

```
133
134
135     /**
136      * Test the spriteTypeAt() method within board
137      */
138     @Test
139     public void testSpriteTypeAt() {
140         board.put(sprite, x, y);
141         assertEquals(sprite.getSpriteType(),
    board.spriteTypeAt(x, y));
142     }
143
144     /**
145      * Test the spriteTypeAt() method out of board
146      */
147     @Test(expected = AssertionError.class)
148     public void testSpriteTypeAtOutOfBoard() {
149         board.spriteTypeAt(20, 20);
150     }
151
152     /**
153      * Test the spriteTypeAt() method with empty sprite
154      */
155     @Test
156     public void testEmptySpriteType() {
157         assertEquals(SpriteType.EMPTY, board.spriteTypeAt(x,
    y));
158     }
159
160
161
162     /**
163      * Test the tileAt() method within board
164      */
165     @Test
166     public void testTileAt() {
167         Tile tile = board.tileAt(x, y);
168         assertEquals(x, tile.getX());
169         assertEquals(y, tile.getY());
170     }
171
172     /**
173      * Test the tileAt() method out of board
```

```
174     */
175    @Test(expected = AssertionError.class)
176    public void testTileAtOutOfBoard() {
177        board.tileAt(20, 20);
178    }
179
180
181
182    /**
183     * Test the tileAtOffSide() method
184     */
185    @Test
186    public void testTileAtOffSide() {
187        Tile start = board.tileAt(x,y);
188        Tile actual = board.tileAtOffset(start, 1, 1);
189        Tile desired = board.tileAt(x+1, y+1);
190
191        assertEquals(actual, desired);
192    }
193
194
195
196    /**
197     * Test the tileAtDirection() method
198     */
199    @Test
200    public void testTileAtDirection() {
201        Tile start = board.tileAt(x, y);
202        Tile actual = board.tileAtDirection(start,
   Direction.RIGHT);
203        Tile desired = board.tileAt(x+1, y);
204        assertEquals(actual,desired);
205    }
206 }
207
208
```

`setUp()`:
Initialize a board with default height and width before testing.

`testConstructor()`:
Test the edge cases when invoking the constructor to create a board with negative width and height.

`testGetWidth()/testGetHeight()`:
Test the returned width/height is equal to the default width/height.

`testPutWithinBoard(), testPutOutOfBoard(), testPutNull(), testPutOccupied()`:
Test put() method that puts a sprite at specific place of the board. The first one is using valid value of sprite, position x and y within board. The second one tests the case that puts a sprite out of the board. The third one tests when putting null on the tile; The fourth one tests when the sprite is already on one tile.

`testWithinBoarders()`:
Test withinBoarders() method which checks whether the given position is within the board.

`testSpriteAt(), testSpriteAtOutOfBoard()`:
Test spriteAt() method that checks the sprite at a specific position. The first one tests the sprite at a valid position; The second one tests the sprite at an invalid position.

`testSpriteTypeAt(), testSpriteTypeAtOutOfBoard(), testEmptySpriteType()`:
Test spriteTypeAt() method that checks the sprite type at specific position. The first one tests the sprite type at a valid position; The second one tests the sprite type at a invalid position; The third one tests the sprite type at a position without any sprite on it.

`testTileAt(), testTileAtOutOfBoard()`:
Test tileAt() method which checks the tile at specific position of the board. The first one tests the tile with valid position value. The second one tests when the given position is out of the board.

`testTileAtOffSize()`:
Test tileAtOffside() method which returns the tile at the offside of another tile.

`testTileAtDirection()`:
Test tileAtDirection() method which returns the tile in specific direction. Here it tests the tile one step right to the start tile.

Exercise 2.5

The domain matrix is as follows:

Boundary conditions for "x >= 0 && x < 10 && y >= 0 && y < 20"														
Variable	Condition	Type	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12
x	>= 0	on	0											
		off		-1										
	< 10	on			10									
		off				9								
	Typical	in					3	4	5	6			7	8
		out									12	14		
y	>= 0	on					0							
		off						-1						
	< 20	on							20					
		off								19				
	Typical	in	4	8	12	14					16		18	
		out										22		24
Result			TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE

```java
package org.jpacman.test.framework.model;

import static org.junit.Assert.*;

import java.util.Arrays;
import java.util.Collection;

import org.jpacman.framework.model.Board;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;


@RunWith(Parameterized.class)
public class BoardWithinBordersTest {

    private Board board;
    private int x, y;
    private boolean isWithin;

    private static final int WIDTH = 10;
    private static final int HEIGHT = 20;

    /**
     * Create a new test case check if within borders at point(x,y)
     */
    public BoardWithinBordersTest(int x, int y, boolean isWithin) {
        board = new Board(WIDTH, HEIGHT);
        this.x = x;
        this.y = y;
        this.isWithin = isWithin;
    }

    /**
     * Test withinBorders() method
     */
    @Test
    public void testWithinBorders() {
        assertEquals(isWithin, board.withinBorders(x, y));
    }
```

```
42
43      /**
44       * One-by-one domain testing
45       * x >= 0 && x < 10 && y >= 0 && y < 20
46       */
47      @Parameters
48      public static Collection<Object[]> data() {
49          Object[][] values = new Object[][] {
50              {0,4,true},
51
52              {-1,8,false},
53
54              {10,12,false},
55
56              {9,14,true},
57
58              {3,0,true},
59
60              {4,-1,false},
61
62              {5,20,false},
63
64              {6,19,true},
65
66              {12,16,false},
67
68              {14,22,false},
69
70              {7,18,true},
71
72              {8,24,false}
73          };
74
75          return Arrays.asList(values);
76      }
77
78
79
80  }
81
```

Exercise 2.6

The coverage of BoardTest class in Board class is 80.6%.

Board	80.6 %	386	93	479
tunnelledCoordinate(int, int, int)	40.5 %	30	44	74
tileAtOffset(Tile, int, int)	61.0 %	47	30	77
Board(int, int)	80.2 %	65	16	81
tileInvariant()	97.5 %	39	1	40
getHeight()	100.0 %	3	0	3
getWidth()	100.0 %	3	0	3
onBoardMessage(int, int)	100.0 %	24	0	24
put(Sprite, int, int)	100.0 %	54	0	54
spriteAt(int, int)	100.0 %	27	0	27
spriteTypeAt(int, int)	100.0 %	36	0	36
tileAt(int, int)	100.0 %	28	0	28
tileAtDirection(Tile, Direction)	100.0 %	8	0	8
withinBorders(int, int)	100.0 %	16	0	16

The coverage of BoardWithinBordersTest class in withinBorders() method is 100%.

Board	26.9 %	129	350	479
tileAtOffset(Tile, int, int)	0.0 %	0	77	77
tunnelledCoordinate(int, int, int)	0.0 %	0	74	74
put(Sprite, int, int)	0.0 %	0	54	54
spriteTypeAt(int, int)	0.0 %	0	36	36
Board(int, int)	66.7 %	54	27	81
spriteAt(int, int)	0.0 %	0	27	27
onBoardMessage(int, int)	0.0 %	0	24	24
tileAt(int, int)	50.0 %	14	14	28
tileAtDirection(Tile, Direction)	0.0 %	0	8	8
getHeight()	0.0 %	0	3	3
getWidth()	0.0 %	0	3	3
tileInvariant()	97.5 %	39	1	40
withinBorders(int, int)	100.0 %	16	0	16

100% is not necessary, because some assertions in the code would never be executed.

Exercise 2.7

Including newly created two test classes, the whole coverage increases from 81.8% to 85%. The coverage of the application code increases from 76.4% to 79.7%. The reason for the coverage rise is that the two newly added test classes cover additional code that was missed before.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	85.0 %	4,892	864	5,756
src/main/java	79.7 %	3,066	780	3,846
org.jpacman.framework.model	80.2 %	1,185	293	1,478
org.jpacman.framework.ui	77.2 %	787	233	1,020
org.jpacman.framework.factory	68.7 %	272	124	396
org.jpacman.framework.view	87.9 %	662	91	753
org.jpacman.framework.controller	80.4 %	160	39	199
src/test/java	95.6 %	1,826	84	1,910

Exercise 2.8

The least two classes are FactoryException 0% and PacmanKeyListener 14.6%.

jpacman-framework	85.0 %	4,892	864	5,756
src/main/java	79.7 %	3,066	780	3,846
org.jpacman.framework.model	80.2 %	1,185	293	1,478
Board.java	80.6 %	386	93	479
Sprite.java	61.0 %	100	64	164
Tile.java	77.7 %	129	37	166
Level.java	65.9 %	56	29	85
PointManager.java	78.2 %	86	24	110
Game.java	90.3 %	205	22	227
Player.java	77.5 %	69	20	89
Food.java	73.3 %	11	4	15
Direction.java	100.0 %	69	0	69
Ghost.java	100.0 %	5	0	5
IBoardInspector.java	100.0 %	64	0	64
Wall.java	100.0 %	5	0	5
org.jpacman.framework.ui	77.2 %	787	233	1,020
ButtonPanel.java	66.1 %	187	96	283
MainUI.java	78.3 %	303	84	387
PacmanKeyListener.java	14.6 %	6	35	41
PointsPanel.java	89.9 %	89	10	99
PacmanInteraction.java	96.2 %	202	8	210
org.jpacman.framework.factory	68.7 %	272	124	396
MapParser.java	66.1 %	181	93	274
DefaultGameFactory.java	80.5 %	91	22	113
FactoryException.java	0.0 %	0	9	9
org.jpacman.framework.view	87.9 %	662	91	753
ImageLoader.java	80.1 %	278	69	347
BoardView.java	94.0 %	346	22	368
Animator.java	100.0 %	38	0	38
org.jpacman.framework.controller	80.4 %	160	39	199
AbstractGhostMover.java	78.0 %	128	36	164
RandomGhostMover.java	91.4 %	32	3	35
src/test/java	95.6 %	1,826	84	1,910

For FactoryException class, a test class is added, including two constructor test methods checking the exception message and cause. Then the coverage increases to 100%. For PacmanKeyListener, by using a robot to trigger the keyPressed method and checking the unexpected exceptions, the coverage increases to 100%.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	86.0 %	5,071	826	5,897
src/main/java	80.9 %	3,110	736	3,846
org.jpacman.framework.model	80.2 %	1,185	293	1,478
org.jpacman.framework.ui	80.6 %	822	198	1,020
ButtonPanel.java	66.1 %	187	96	283
MainUI.java	78.3 %	303	84	387
PointsPanel.java	89.9 %	89	10	99
PacmanInteraction.java	96.2 %	202	8	210
PacmanKeyListener.java	100.0 %	41	0	41
org.jpacman.framework.factory	71.0 %	281	115	396
MapParser.java	66.1 %	181	93	274
DefaultGameFactory.java	80.5 %	91	22	113
FactoryException.java	100.0 %	9	0	9
org.jpacman.framework.view	87.9 %	662	91	753
org.jpacman.framework.controller	80.4 %	160	39	199
src/test/java	95.6 %	1,961	90	2,051

```java
package org.jpacman.test.framework.factory;

import static org.junit.Assert.assertEquals;
import org.jpacman.framework.factory.FactoryException;
import org.junit.Test;

public class FactoryExceptionTest {

    /**
    * Test FactoryException constructor with message
    */
     @Test
     public void testConstructorWithMes(){
        String message = "Factory exception message";
        try {
            FactoryException f = new FactoryException(message);
        } catch(RuntimeException e) {
            assertEquals(message, e.getMessage());
        }
     }

    /**
    * Test FactoryException constructor with message and cause
    */
     @Test
     public void testConstructorWithMesCause(){
        String message = "Factory exception message";
        Throwable cause = new Throwable();
        try {
            FactoryException f = new FactoryException(message, cause);
        } catch(RuntimeException e) {
            assertEquals(message, e.getMessage());
            assertEquals(cause, e.getCause());
        }
     }
}

```

```java
package org.jpacman.test.framework.ui;

import static org.junit.Assert.fail;

import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;

import org.jpacman.framework.factory.FactoryException;
import org.jpacman.framework.ui.MainUI;
import org.junit.Test;

public class PacmanKeyListenerTest {


    @Test
    public void testKeyPressed() throws FactoryException,
AWTException {

        MainUI ui = new MainUI();
        ui.initialize(); //call addKeyListener(new
PacmanKeyListener(pi));
        ui.start();

        Robot robot = new Robot();


        int[] keycodes = {KeyEvent.VK_S, //start
                        KeyEvent.VK_UP, //up
                        KeyEvent.VK_DOWN, //down
                        KeyEvent.VK_K, //up
                        KeyEvent.VK_J, //down
                        KeyEvent.VK_LEFT, //left
                        KeyEvent.VK_RIGHT, //right
                        KeyEvent.VK_H, //left
                        KeyEvent.VK_L, //right
                        KeyEvent.VK_Q, //stop
                        KeyEvent.VK_X //exit
                        };

        try {
            for (int keycode : keycodes) {
                robot.keyPress(keycode);
```

```
42                  robot.keyRelease(keycode);
43                  robot.delay(1000);
44              }
45          } catch(Throwable e) {
46              fail("Unexpected exception in keyListener");
47          }
48
49      }
50
51  }
52
```