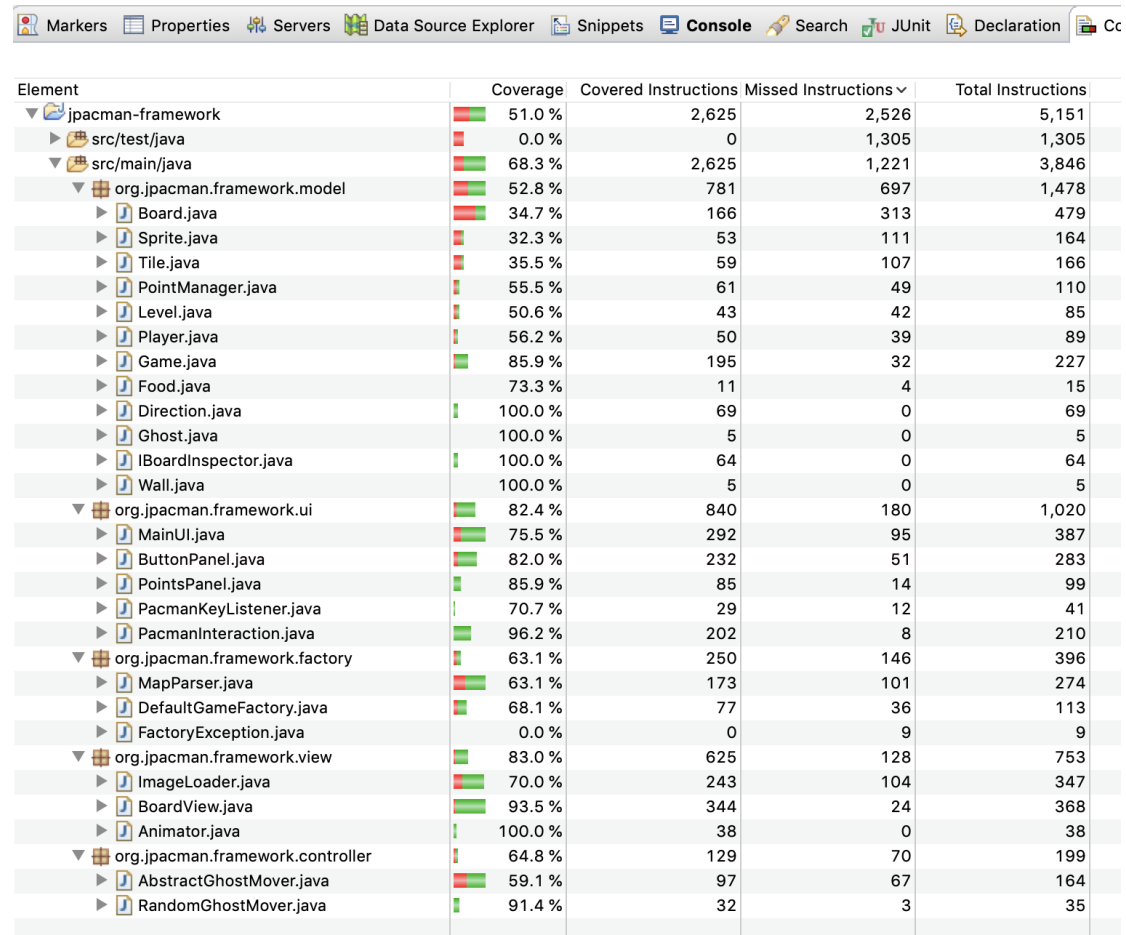


Assignment 2

2.1

The chosen scenario is in the first chapter that goes from the start of the game to losing the game.

The screenshot of the coverage view after the tour is as follows:



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	51.0 %	2,625	2,526	5,151
src/test/java	0.0 %	0	1,305	1,305
src/main/java	68.3 %	2,625	1,221	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	82.4 %	840	180	1,020
MainUI.java	75.5 %	292	95	387
ButtonPanel.java	82.0 %	232	51	283
PointsPanel.java	85.9 %	85	14	99
PacmanKeyListener.java	70.7 %	29	12	41
PacmanInteraction.java	96.2 %	202	8	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

From the coverage view and details in the application code, interesting findings are listed below:

1. PointManager.js : 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).

```

    */
    protected boolean invariant() {
        return pointsEarned >= 0
            && pointsEarned <= pointsPutOnBoard;
    }

    /**
     * While building the game, keep track of the
     * total number of points.
     * @param delta Points to be added to the game
     */
    public void addPointsToBoard(int delta) {
        assert delta >= 0;
        pointsPutOnBoard += delta;
        assert invariant();
    }

```

Explanation: From class, we learned that the assert statement will be transformed by compiler to conditioning, so it is reasonable that it only partially covered as only the path that run with conditioning as true be executed.

2. FactoryException.js: 0% code coverage.

Explanation: It is reasonable since there is no exception raised during the runtime.

3. There are many variable/constant declarations that are not labeled in green/yellow/red. For example, in “DefaultGameFactory” class, it declares a game reference without any color.

```

    */
    public class DefaultGameFactory implements IGameFactory {

        private transient Game theGame;

```

Explanation: Only when objects or values assigned to the reference, it will be counted in term of coverage. Declarations are not counted in coverage.

2.2

The coverage rises from 51% to 78%. The PointManager stated before witness a rise of coverage from 55 to 76.4%.

MainUI (1) (13-Feb-2019 2:04:44 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jpacman-framework	58.3 %	3,004	2,147	5,151
src/test/java	0.0 %	0	1,305	1,305
src/main/java	78.1 %	3,004	842	3,846
org.jpacman.framework.model	70.5 %	1,042	436	1,478
org.jpacman.framework.ui	86.0 %	877	143	1,020
org.jpacman.framework.factory	68.7 %	272	124	396
org.jpacman.framework.view	86.7 %	653	100	753
org.jpacman.framework.controller	80.4 %	160	39	199

All assertions are covered. This is the reason for the rise of coverage.

2.3

The whole coverage is 81.9%, test suite coverage is 98.3% and the application code coverage is 76.4%.

java (13-Feb-2019 2:12:18 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ jpacman-framework	81.9 %	4,221	930	5,151
▼ 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	98.3 %	1,283	22	1,305
▶ org.jpacman.test.framework.model	98.7 %	901	12	913
▶ org.jpacman.test.framework.accept	94.8 %	184	10	194
▶ 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

The application code coverage is more important. It is easy to write a 100%-coverage test suite with very low application code coverage, which does not make any sense. But a high application code coverage will show how well your test suite go through the application.

2.4

Test cases were added to test. The basic functionality of all public class methods, some potential edge cases like when invalid input is provided and some special cases were considered here, e.g., multiple sprites are on the same tile and a tile move across the boundary. The coverage of the board class could reach 83.9% with the test class.

Explanations on the test class and test cases:

1. Create many constants for use in the testing. Create a board with the height and width before testing.
2. Test the edge cases when using the constructor to create a board with negative values.

```

@Test
public void testConstructor(){
    try{
        new Board(-1,10);
    }catch(AssertionError e){
    }
    try{
        new Board(10,-1);
    }catch(AssertionError e){
    }
}

```

3. Test getHeight/Width() method to see whether the return height and width are the same as those used in constructing the board.

```

private Board board;
private final int height = 10;
private final int width = 10;
private final Sprite sprite = new Sprite() { };
private final int x = 0;
private final int y = 0;

/**
 * Initialize a board with default height and width
 */
@Before
public void setUp(){
    board = new Board(width, height);
}

```

4. Test put() method which put sprite at specific place of the board. The first one is using valid value of sprite, position value x and y within board; The second one tests when put a sprite out of the board; The third one tests put a sprite which is null on the tile; The fourth one tests when the sprite is already on one tile, it is to put on another tile.

```

/**
 * Test the put() method, put sprite on tile without any assertions
 */
@Test
public void testSpritePutOnTile(){
    board.put(sprite, x, y);
    assertEquals(board.tileAt(x, y).topSprite(), sprite);
}

/**
 * Test the put() method, put sprite on tile that is without board
 */
@Test
public void testSpritePutOnTileOffBoard(){
    try{
        board.put(sprite, 11, 1);
    }catch(AssertionError e){
    }
}

/**
 * Test the put() method, put sprite null on tile
 */
@Test
public void testSpritePutNullOnTile(){
    Sprite sprite = null;
    try{
        board.put(sprite, x, y);
    }catch(AssertionError e){
    }
}

/**
 * Test the put() method, put sprite that is already on tile onto tile
 */
@Test
public void testExistSpritePutOnTile(){
    board.put(sprite, x, y);
    try{
        board.put(sprite, 2, 1);
    }catch(AssertionError e){
    }
}

```

5. Test `spriteAt()` method that check the sprite at specific position. The first one tests the sprite at a valid position; The second one tests the sprite at a invalid position.

```

/**
 * Test the spriteAt() method
 */
@Test
public void testSpriteAt(){
    board.put(sprite, x, y);
    assertEquals(board.spriteAt(x, y), sprite);
}

/**
 * Test the spriteAt() method with withinboarders() false
 */
@Test
public void testSpriteAtOutBoarders(){
    try{
        board.spriteAt(11, 1);
    }catch(AssertionError e){
    }
}

```

6. Test `spriteTypeAt()` method that check 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.

```

/**
 * Test the spriteTypeAt() with new sprite method
 */
@Test
public void testSpriteTypeAt(){
    board.put(sprite, x, y);
    assertEquals(board.spriteTypeAt(x, y), sprite.getSpriteType());
}

/**
 * Test the spriteTypeAt() method with withinboarders() false
 */
@Test
public void testSpriteTypeAtOutBoarders(){
    try{
        board.spriteTypeAt(11, 1);
    }catch(AssertionError e){
    }
}

@Test
public void testNullSpriteTypeAt(){
    assertEquals(board.spriteTypeAt(x, y), SpriteType.EMPTY);
}

```

7. Test the case when multiple sprites put on the same tile.

```

/**
 * Test what happens if there are multiple sprites on one tile.
 */
@Test
public void multipleSprites() {
    Sprite top = new Sprite() {};
    board.put(top, x, y);

    // now 'top' is the top most sprite.
    assertEquals(board.spriteAt(0, 0), equalTo(top));
}

```

8. 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.

```

@Test
public void testNullSpriteTypeAt(){
    assertEquals(board.spriteTypeAt(x, y), SpriteType.EMPTY);
}

/**
 * Test the tileAt() method
 */
@Test
public void testTileAt(){
    Tile tile = board.tileAt(x,y);
    assertEquals(tile.getX(), x);
    assertEquals(tile.getY(), y);
}

/**
 * Test the tileAt() method with withinboarders() false
 */
@Test
public void testTileAtOutBoarder(){
    try{
        Tile tile = board.tileAt(11,11);
    }catch(AssertionError e){
    }
}

```

9. Test withinBoarders() method which check whether the given position is within the board.

```

/**
 * Test the withinBoarders() method
 */
@Test
public void testWithinBorders(){
    assertEquals(false, board.withinBorders(height, width));
}

```

10. Test `tileAtOffside()` method which return the tile at the offside of another tile. The first one tests the case with valid value; the second one tests when the start tile is not a valid one (it is null).

```
/**
 * Test the tileAtOffside() method
 */
@Test
public void testTileAtOffside(){
    Tile start= board.tileAt(x,y);
    Tile tile = board.tileAtOffset(start, 1, 1);
    assertEquals(tile, board.tileAt(x+1, y+1));
}

/**
 * Test the tileAtOffside() method with Start as null
 */
@Test
public void testTileAtOffsideStartNull(){
    try{
        Tile start= null;
        Tile tile = board.tileAtOffset(start, 1, 1);
    }catch(AssertionError e){
    }
}
```

11. Test `tileAtDirection()` method which return the tile at specific direction. The first one tests the tile one step left to the start tile; the second one tests the ones across the boundary at two side of the board.


```

/**
 * Test tileAtDirection() method
 */
@Test
public void testTileAtDirection(){
    Tile start= board.tileAt(1,1);
    Tile tileEnd = board.tileAtDirection(start, Direction.LEFT);
    Tile expectEnd = board.tileAt(0, 1);
    assertEquals(tileEnd, expectEnd);
}

/**
 * Test what happens if a tile moves across the boundary
 */

@Test
public void testTileAtDirectionAcrossBound(){
    Tile start= board.tileAt(x, y);
    Tile actual = board.tileAtDirection(start, Direction.LEFT);
    Tile desired = board.tileAt(width-1, y);
    assertEquals(actual, desired);
}

```

2.5

The domain matrix is as follow:

Boundary conditions for "0<=x<10 && 0<=y<10 "											
Variable	Condition	type	t1	t2	t3	t4	t5	t6	t7	t8	t9
x	>=0	on	0								
		off		-1							
	<10	on			10						
		off				9					
y	typical	in					5	6	7	8	4
	>=0	on					0				
		off						-1			
	<10	on							10		
		off								9	
	typical	in	1	2	3	4					5
Expected results			T	F	F	T	T	F	F	T	T

Code:

```

@RunWith(Parameterized.class)
public class BoardWithinBoardersTest {

    private Board board;
    private final int height = 10;
    private final int width = 10;
    private int tx;
    private int ty;
    private boolean flag;

    /**
     * Initialize a board with default height and width,
     * and create x, y, expected flag from parameters
     */
}

```

```

public BoardWithinBordersTest(int x, int y, boolean f){
    board = new Board(width, height);
    tx = x;
    ty = y;
    flag = f;
}

```

```

/**
 * Test WithinBorders
 */
@Test
public void testWithinBorders(){
    assertEquals(board.withinBorders(tx, ty), flag);
}

```

```

/**
 * Provide parameters to constructor of class
 */
@Parameters
public static Collection<Object[]> data() {
    Object[][] values = new Object[][] {
        //on point
        {0, 1, true},
        //off point
        {-1, 2, false},
        //off point
        {10, 3, false},
        //on point
        {9, 4, true},
        //on point
        {5, 0, true},
        //off point
        {6, -1, false},
        //off point
        {7, 10, false},
        //on point
        {8, 9, true},
        //in point
        {4, 5, true},
        //out point
        {20, 20, false}
    };
}

```

```

        return Arrays.asList(values);
    }

}

```

2.6

The coverage of Board.java increased to 83.9%. 100% is not necessary, because some assertions in the code would never be called.

Element	Coverage	Covered Instructions	Missed Ins...	Total Instr...
Board.java	83.9 %	402	77	479
Board	83.9 %	402	77	479
tunnelledCoordina	40.5 %	30	44	74
tileAtOffset(Tile, in	67.5 %	52	25	77
Board(int, int)	93.8 %	76	5	81
tileInvariant()	97.5 %	39	1	40
getHeight()	100.0 %	3	0	3
getWidth()	100.0 %	3	0	3
onBoardMessage(i	100.0 %	24	0	24
put(Sprite, int, int)	100.0 %	54	0	54
spriteAt(int, int)	100.0 %	27	0	27
spriteTypeAt(int, ir	100.0 %	36	0	36
tileAt(int, int)	100.0 %	28	0	28
tileAtDirection(Tile	100.0 %	8	0	8
withinBorders(int,	100.0 %	16	0	16

From the above, we also know that “withinBorders” are completely coverage.

2.7

After the modification, the total coverage increased from 81.9% to 84.7%, and the coverage of application code increased from 76.4% to 80.7%. These are the benefit of the two newly added test cases.

Element	Coverage	Covered Instructions	Missed Ins...	Total Instr...
jpacman-framework	85.5 %	5,093	864	5,957
src/main/java	80.7 %	3,167	756	3,923
src/test/java	94.7 %	1,926	108	2,034

2.8

The least two classes are FactoryException (0%) and PacmanKeyListener (14.6%). For FactoryException.java, increase the coverage to 100% by calling two constructors and check the message.

For PacmanKeyListener.java, after trying key event and key release, the coverage increased to 100%.

FactoryException.java	100.0 %	9	0	9
PacmanKeyListener.java	100.0 %	41	0	41

```
public class FactoryExceptionTest {
```

```
    private MapParser parser;
```

```

private String[] map = {};

/**
 * Initialization
 */
@Before
public void setUp() {
    IGameFactory factory = new DefaultGameFactory();
    parser = new MapParser(factory);
}

/**
 * Common FactoryException testing without cause
 * @throws FactoryException
 */
@Test(expected=FactoryException.class)
public void testFactoryException() throws FactoryException {
    parser.parseMap(map);
}

/**
 * Common FactoryException constructor testing without cause
 */
@Test
public void testFactoryExceptionConstructorWithoutCause() {
    String message = "Problem reading file ";
    try {
        FactoryException f = new FactoryException(message);
    } catch (Exception e) {assertEquals(e.getMessage(),message);}
}

/**
 * Common FactoryException constructor testing with cause
 */
@Test
public void testFactoryExceptionConstructorWithCause() {
    String message = "Problem reading file ";
    Throwable t = new Throwable();
    try {
        FactoryException f = new FactoryException(message, t);
    } catch (Exception e) {
        assertEquals(e.getMessage(),message);
        assertEquals(e.getCause(),t);
    }
}

```

```

    }

}

public class PacmanKeyListenerTest {
    KeyEvent up;
    KeyEvent down;
    KeyEvent left;
    KeyEvent right;
    MainUI ui = new MainUI();
    @Before
    public void setUp(){
        up = new KeyEvent(ui, 0, 0, 0, KeyEvent.VK_UP, 'a');
        down = new KeyEvent(ui, 0, 0, 0, KeyEvent.VK_DOWN, 'b');
        left = new KeyEvent(ui, 0, 0, 0, KeyEvent.VK_LEFT, 'c');
        right = new KeyEvent(ui, 0, 0, 0, KeyEvent.VK_RIGHT, 'd');
    }
    @Test
    public void testUIActions() {

        try {
            Robot robot = new Robot();
            ui.initialize();
            ui.start();
            robot.keyPress(KeyEvent.VK_S);
            Thread.sleep(3000);
            robot.keyPress(KeyEvent.VK_K);
            robot.keyPress(KeyEvent.VK_J);
            robot.keyPress(KeyEvent.VK_H);
            robot.keyPress(KeyEvent.VK_L);
            robot.keyPress(KeyEvent.VK_Q);
            Thread.sleep(3000);
            robot.keyPress(KeyEvent.VK_X);
        } catch (Exception e) {fail("Keytype failed!");}
    }
}

```