Assignment 1

Ziyan Gong 94478161
Zhen Wang 98552169

Exercise 1.2: Exploratory Testing Apply exploratory testing to JPacman.
Test Charter 1: Analyze the function of start/stop in JPacman
Actor: Normal Jpacman player
Purpose: To evaluate if the button panel works properly in this game.
Setup: A MacBook Pro 2017 with 16GB memory and Intel Iris graphics card, macOS Mojave operating system, eclipse IDE.
Priority: High, because this is a fundamental function for the whole system and used very frequently.
Data: JPacman-framework package.
Activities/ Tours:

1. Run this java application. Observation: Start button is clickable but Stop button is not.
2. Click Start button, the game begins. Observation: Keys that control the object moves can be used; the status bar shows "Playing";the Start button turns to grey and Stop button can be clicked.
3. Click Stop. Observation: The game pauses; everything in the play field stops running. Start button is available and Stop is not now.
4. Click Start again. When the object is caught by the enemy, the game is over. Observation: The status bar shows "You have lost". Only "Exit" button is clickable on the button panel.

Missing functions: No opening animation and no music after entering the game. The change of the status bar is not easy to notice, a pop-up will be better.

Test Charter 2: Analyze the action of hitting the wall in Jpacman
Actor: Normal Jpacman player
Purpose: To evaluate if the point manager works properly in this game.
Setup: A MacBook Pro 2017 with 16GB memory and Intel Iris graphics card, macOS Mojave operating system, eclipse IDE.
Priority: High, because this is an action that happens all the time through the game, it is part of the integrity of the game and it influences the validity and calculating of points.
Data: JPacman-framework package.
Activities/Tours:

1. Run this java application. Observation: The point manager shows 0/1780. The Jpacman stays on a tile not a wall.
2. Move the Jpacman up until it hits the wall. Observation: The Jpacman cannot move any more and the points do not increase as well with new "up" command.
3. Move the Jpacman down until it hits the wall. Observation: The Jpacman cannot move any more and the points do not increase as well with new "down" command.
4. Move the Jpacman left until it hits the wall. Observation: The Jpacman cannot move any more and the points do not increase as well with new "left" command.
5. Move the Jpacman right until it hits the wall. Observation: The Jpacman cannot move any more and the points do not increase as well with new "right" command.

6. Randomly repeat step 2-5 to explore more walls in the play field until the end of the game.

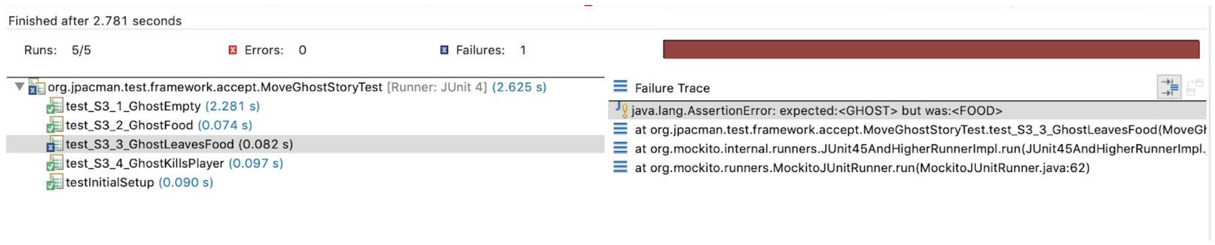
Missing function: When walking forward and the left (or right) is the wall, turn left (or right), the face of Jpacman does not turn.

Test Charter 3: Analyze the function of the point manager in Jpacman
Actor: Normal Jpacman player
Purpose: To evaluate if the point manager works properly in this game.
Setup: A MacBook Pro 2017 with 16GB memory and Intel Iris graphics card, macOS Mojave operating system, eclipse IDE.
Priority: Median, because this is a function that will run through the whole game and will be used as the judgement of winning. But the point makes little sense when the player loses the game.
Data: JPacman-framework package.
Activities/Tours:

1. Run this java application. Observation: The point manager shows 0/1780.
2. Move the Jpacman up/down/right/left to a foodtile. Observation: The Jpacman move to the new tile for each step. The point manager shows points added by 10 for each step.
3. Move the Jpacman up/down/right/left to an empty tile. Observation: The Jpacman move the the new tile for each step. The point manager shows points staying the same for each step.
4. Move the Jpacman up/down/right/left to the wall. Observation: The Jpacman does not move. The point manager shows points staying the same.
5. Be caught by the ghost. Observation: Lose the game. The point stays the same as the manually counted eaten foodtile.

Missing function: No instruction on the point manager panel. For example, users do not know how many points they can get for each step without actually playing it.

Exercise 1.3: Running the Test Suite
There are 10 test classes and 39 test cases.
Action: In the test_S3_GhostLeavesFood() function in MoveGhostStoryTest class,change assertion from FOOD to SpriteType.GHOST.
Findings: The test raised one failure caused by this function. The error message is "java.lang.AssertionError: expected: <GHOST>, but was: <FOOD>" showing in the Failure Trace followed by the methods that are called when resulting this failure.



```
Finished after 2.781 seconds

Runs:   5/5          Errors:   0          Failures:   1

org.jpacman.test.framework.accept.MoveGhostStoryTest [Runner: JUnit 4] (2.625 s)
    test_S3_1_GhostEmpty (2.281 s)
    test_S3_2_GhostFood (0.074 s)
    test_S3_3_GhostLeavesFood (0.082 s)
    test_S3_4_GhostKillsPlayer (0.097 s)
    testInitialSetup (0.090 s)

Failure Trace
java.lang.AssertionError: expected:<GHOST> but was:<FOOD>
at org.jpacman.test.framework.accept.MoveGhostStoryTest.test_S3_3_GhostLeavesFood(MoveGh
at org.mockito.internal.runners.JUnit45AndHigherRunnerImpl.run(JUnit45AndHigherRunnerImpl.
at org.mockito.runners.MockitoJUnitRunner.run(MockitoJUnitRunner.java:62)
```

Exercise 1.4
Look into MoveGhostStoryTest.

Before the testing, the class run setUp to create a simpler map with player on the left of the ghost, player, etc with testing resources. And also create one emptyTile at the location (2,2) which is one step down to the ghost and one foodTile at the location (2,0) which is one step up to the ghost.
Four moving functions of ghost are tested here.
1. Test that a ghost can move towards an empty tile.
Make the ghost moves down one step. And check the tile that the ghost locates is empty or not, using *assertEquals.*
2. Test that a ghost can move over food.
Make the ghost moves up one step. And check the tile that the ghost locates is food or not, using *assertEquals. Then check the Spritetype of the top sprite of this foodtile is ghost or not with assertEquals*
3. Test that a ghost can leave the food once he's on it.
If the second test success, move the ghost down, i.e., move away from the foodtile. Check whether the Spritetype of food tile is food or not.
4. Test that the ghost causes the player's death if the ghost bumps into the player.
Make the ghost moves left one step. Check the player is alive or not using assertFalse.

Evaluation: Test class don't test that a ghost can't keep moving if the forward is wall.
The name of test function is not good enough. Take the first one as an example. The name test_S3_2_GhostFood is hard to understand what S3, 2 and GhostFood are.