Assignment 4

Group 21

- Kelly
- Shelby
- Steven

Chosen Project

We chose spaceinvaders (https://github.com/dwmkerr/spaceinvaders) because it is was made with only HTML and JS, no third party libraries, and it has a lot of Javascript code (915 LOCs, 56 Functions), and an interactive UI, which will provide ample content for us to test with Qunit and Selenium.

Space Invaders is a web app game that uses the arrow keys to move your character, and the space bar to fire bombs at the invaders. As the game goes on, the invaders get faster and they drop more bombs.

You have 3 lives, and every time you get hit by an invaders bomb you lose a life. When you reach zero lives, the game is over. The game gives a summary of you score and what level you got to, and gives you a prompt to play again.

When you reach the end of a level, it gives you a summary of you score and the next level you are going to be on.

When you advance to the next level you do not gain back any lives.

Exercise 4.1

KeyboardEventTest

Test cases:

- press spacebar to start the game
- press spacebar to fire
- press left arrow key to move left
- press right arrow key to move right
- press left arrow key and spacebar to move left and fire
- press right arrow key and spacebar to move right and fire

MouseClickEventTest

Test cases:

- click 'mute' link to mute the game
- click 'unmute' link to unmute the game
- click 'spaceinvaders on github' to navigate to the source code repo
- click 'more experiments' to navigate to author website view more experiments
- click 'more dwmkeer.com' to navigate to author website
- click back button to navigate back to spaceinvaders

StaticContentTest

Test cases:

- check that all elements are present
- check that 'gameCanvas' size is at 600 x 800
- check that the 'info' box has width of 800
- check that 'info' is displaying the correct content
- check that all links are displaying the correct text

Exercise 4.2

The quality of the test cases are measured by assertions and page objects models.

- Assertions checks whether the each element in the UI is presented the way it is expected to be presented (in terms of size and inner text).
- Assertions are also used to check that links presented in the UI are clickable and each link navigates to the correct web pages.
- Page objects models however makes the code cleaner and easier to understand. Any changes to the UI can be changed in the page objects, this makes it less error prone.
- Page objects also makes it easier for developers/testers to write tests.

Exercise 4.3

Client-side coverage

Metric 1: Test Execution Coverage The first run of the test classes gave us %88.3 coverage.

- MouseClickEventTest.java - 42.0%
- StaticContentTest.java - 97.1%
- KeyboardEventTest.java - 98.1%
- TestSpaceInvadersMainPage.java - 100.0%
- SpaceInvadersMainPage.java - 92.8%

This was because the test in MouseClickEventTest.java e.g. testMouseClickEvent() was not executed. This seemed to be due to the rocket getting killed before the clicks had occured. A second run of the test classes gave 96.8% coverage. The only parts of the code that weren't covered were catch and fail statements. All clicking events in each part of the code were then covered.

Metric 2: Branch Coverage We only had 2 if statements in our code and since a non-executed fail statement followed them, we had 0% branch coverage.

Metric 3: Class Coverage Every test class in our suite had at least 1 method executed so we had 100% class coverage.

Assignment 5

Exercise 5.1

We are using QUnit to test our JavaScript code. We chose QUnit because it is really easy to use and our group has some experience with QUnit already.

As an example, to execute the tests, you just need to open the html file in a web browser, and all the QUnit tests are executed and the results are displayed nicely.

Pros:

- Easy to use
- Gives a nice html page with the results
- Opening the html page executes all of the tests.

Cons:

Exercise 5.2

- Check that the mute button correctly toggles the sound
- Check that the initialization method is working correctly
- Check the playstate enter and that the values change correctly with each new level.
- Check that the currentState and moveToState functions are working as intended.
- Check that the stats for stars are within the alloweable bounds (all the values are generated randomly, so it checks that the values are within the correct bounds).

Exercise 5.3

There are two JavaScript files in this project:

- Starfield.js = 64.29% Coverage.

- spaceinvaders.js = 60.7% Coverage.

The coverage is not too bad so far, and keeping in mind that there are parts of the script that are hard to test because there is not much to assert because they perform the action of running the game.

Also, this is a good example of how coverage is not a good metric for how good your tests are. The update method for the PlayState is 182 lines long and performs a lot of functionality. Therefore, if you run this method once to test one specific part of this method, it shows that the whole method has been covered even though only a small portion of this method was actually asserted.

For code coverage we used Blanket.js. Coverage is measured with this by opening the html document and checking the box saying "enable coverage" (See the readme in JS-testcases).

It gave a good indication of how much code was left untested, and made it easier to find more code to try to test.