4. Software Testing Report

Group 11 - 11 Musketeers

Osama Azaz
Adam Dawtry
Tom Jackson
Brendan Liew
Holly Reed
Harry Ryan

Testing Methods and Approaches

The testing we used and applied can be split into the two main principle techniques of Dynamic and Static methods of testing. Where we could we enforced dynamic techniques within our code to ensure its integrity and in the cases where visual textures and visual acknowledgement were needed we enforced static methods. The methods used can be detailed and briefly explained below:

Unit testing [2]

- Since manual testing is expensive and time consuming, we chose to automate as many test cases as possible. This ensures the components that we test are doing what we want them to do and if there are any defects or bugs in the code it should be revealed by the test cases.
- Unit Tests are particularly useful when dealing with numerical code values such as damage numbers and health changes where a visual indicator is not always a representation of what the game is doing.

Integration testing/component testing [2]

- Integration testing was employed mostly when it came to testing the validity of our loading and saving functions since we had to communicate with our file system from our game application.

System testing [2]

- Some of our system functionality and characteristics may only become obvious when the system is put together which causes an emergent behaviour which could be unplanned and unwanted. System testing is to ensure that the system is only what it is supposed to do and in order to do this we had to enforce a manual technique of play testing.
- This gives us the most realistic view of the system under test and also helps to capture the user's perspective.

  Black box testing (requirements based testing)
- Black box testing is used to test techniques that do not require knowledge of the underlying implementation to derive test cases. Requirement-driven tests.

  Steps
  - Read the requirements.
  - Identify the input and output variables in play, together with their types, and their ranges.
  - Identify how input variables influence the output variable.
  - Perform equivalent class analysis (valid and invalid classes). Explore the boundaries of these classes.
  - Generate a set of test cases that should be executed against the system under test.
  - Repeat for the rest of requirements or features.

  Grey box testing
  - Similar to black box testing however both methods of testing automatic and manual are completed by the programmers. [3]

Manual Testing

- The manual testing was an especially large part of the project since a large proportion of the game is graphical and to understand how the user would perceive the game it is a good approach to play through as a player would. We linked this method with our black box requirements based approach to ensure that our intended product met the expectations.

Testing Report

The testing of our game consisted of primarily manual testing in the form of playtesting for the reason that many of our components existed in a visual and interactive manner and could not be manipulated through the usage of coded Junit tests. Therefore this section will be split into two sections, one to show the details of our Junit tests and the other to show the outcome of the manual testing.

Automatic JUnit Tests (27% Method Coverage)

Classes Tested (10 / 22)

- Boat (All Tests Passed - 70% Method Coverage)
    - Does the Boat take damage? (True)
    - Does the Boat move? (True)
    - Does the Boat collide with terrain? (True)
    - Does the Boats data (Location, Team, Direction, Health) get saved? (True)
- College (All Tests Passed - 55% Method Coverage)
    - Does the college take damage? (True)
    - Can boats be added to the college? (True)
    - Does the college's data (Location, Team, Direction, Health) get saved? (True)
- Enemy_Wave (All Tests Passed - 57% Method Coverage)
    - Does the wave save and load from the Json file? (True)
- GameObject (All Tests Passed - 75% Method Coverage)
    - Can we set an object's max health? (True)
    - Can we set an object's hitbox? (True)
    - Does the object take damage? (True)
    - Does the object collide with another gameObject? (True)
    - Does the object save its data values (Size, Location, Health, Team)? (True)
- Indicator (All Tests Passed - 80% Method Coverage)
    - Does the indicator point at the correct gradient to the college? (True)
- Player (All Tests Passed - 80% Method Coverage)
    - Does the player collide with terrain? (True)
    - Can the player move? (True)
    - Does the player take damage when hit by a projectile? (True)
    - Does the player know it's dead when it has no health left? (True)
    - Does the player not take damage when invincible? (True)
    - Does the player's attack speed increase (powerup)? (True)
    - Does the player's attack damage increase (powerup)? (True)
    - Does the player's health increase (powerup)? (True)
    - Does the player's speed increase (powerup)? (True)
    - Does the attack speed increase when bought? (True)
    - Does the attack damage increase when bought? (True)
    - Does the player's speed increase when bought? (True)
    - Does the player's data (Location, Powerup status, health, upgrades) save to the Json correctly? (True)

- PowerUps (All Tests Passed - 50% Method Coverage)
    - Are the powerups that exist in the game world saved and loaded correctly? (True)
- Projectile (All Tests Passed - 60% Method Coverage)
    - Can we create a projectile and does its speed correspond to what created it, player or enemy? (True)
    - Does the projectile save and load correctly? (True)
- WeatherManager (All Tests Passed - 71% Method Coverage)
    - Can we create wave obstacles that target the player? (True)
    - Does the location of the waves save to the Json? (True)
    - Is the state of the weather saved and loaded correctly? (True)

Manual Testing

The aim of the manual testing was to ensure that all of the user requirements were tested and met without any bugs or problems when the game was played. Manual testing allowed us to enforce system testing and see that the game is visually appealing and behaving as it should before giving it to the client. To design these manual tests we created a traceability matrix as shown on our website under testing the below tests summary uses Requirement ID and Test ID as a referral key to this.

Requirements Testing (Black Box / Grey Box)

- Obstacles / Weather Testing (Req_01)
    - (Test_01) Do waves do damage? (True)
    - (Test_02) Does terrain stop waves? (False)
    - (Test_03) Does bad weather occur? (True)
    - (Test_04) Does bad weather have visual indicators? (True)
- Ship Combat (Req_02)
    - (Test_01) Do enemy ships engage the player (True)
    - (Test_02) Can enemy ships be damaged / destroyed (True)
- Enemy Colleges (Req_03)
    - (Test_01) Are the colleges spawned correctly? (True)
    - (Test_02) Do they deal and take damage? (True)
- Points and gold (Req_04)
    - (Test_01) Are points and loot visually displayed correctly? (True)
    - (Test_02) Are the points accumulated through various tasks? (True)
- Game Objectives (Req_05)
    - (Test_01) Are the game objectives displayed to the player correctly? (True)
- Spending Loot (Req_06)
    - (Test_01) Can the shop be accessed? (True)
    - (Test_02) Can the player buy upgrades through the shop? (True)
    - (Test_03) Does the upgrade work? (True)
    - (Test_04) Does the shop prevent more than one upgrade to be bought? (True)
- Powerups (Req_07)
    - (Test_01) Is there 5 powerup types? (True)
    - (Test_02) Does each powerup have its intended effect? (True)
    - (Test_03) Does each powerup have its own timer? (True)
- Difficulty (Req_08)
    - (Test_01) Is the player forced to choose a difficulty? (True)

    - (Test_02) Does the difficulty selected affect the gameplay? (True)
- Save / Loading Functionality (Req_09)
    - (Test_01) Is player data saved and loaded? (True)
    - (Test_02) Are player power ups saved and loaded? (True)
    - (Test_03) Are objectives and scores saved and loaded? (True)
    - (Test_04) Are obstacles and weather saved and loaded? (True)
    - (Test_05) Are projectiles saved and loaded? (True)
    -

The only test that failed across both testing methods was the manual test (Req_01)(Test_02) which was that on a visual level the wave obstacles could pass through terrain. This was not intended as part of our design and failed as a result of that. In order to fix this and enable all tests to be passed we would have to find a way to relay the Tiled Map layout to the obstruction and pass the edges of terrain to it so that it knows where the terrain is. However this could not be attained by the final implementation without needing to refactor how the entire map is produced which would affect multiple other classes and requirements.

Comments about testing

- Aside from the single failed test, our testing methods cover all of the clients requirements as detailed in the product brief and the added requirements brought in when we took over this project.
- When testing we strived to obtain 100% coverage of our entire code, ideally this could have been done via automated testing however, this is impossible due to our code being heavily reliant on textures and visuals. Therefore many of the tests have been done manually in order to obtain a complete set of tests.
- Manual tests also meant we could test from a user perspective as detailed in The practical test pyramid [4] . It is useful to test the User Interface to check that the game is visually pleasing when trying to appeal to a certain audience.
- Additionally there are a few tests we decided not to automate even where we could due to the simplicity of them, we found testing getters and setters in our code would be a waste of resources and time as we could clearly determine they were working if the code would compile correctly.

All Testing Design and Evidencing can be found at the following link:
https://uoy-eng1-team-11.github.io/testing

References:

[1] I. Somerville, *Software Engineering*, Global Edition. Pearson Education, 2016. [E-book] Available: https://ebookcentral.proquest.com [Accessed: 22 April 2022].

[2] N. Matragkas, Engineering 1, “Software Testing.” ENG1, Computer Science, University of York, York, 2021.

[3] “Manual testing - javatpoint,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/manual-testing. [Accessed: 03-May-2022].

[4] “The practical test pyramid,” *martinfowler.com*. [Online]. Available: https://martinfowler.com/articles/practical-test-pyramid.html. [Accessed: 03-May-2022].