Team iPatch

Assessment 4: Evaluation and Testing report

Christian Pardillo Laursen
Filip Makosza
Joseph Leigh
Mingxuan Weng
Oliver Relph

Evaluation approach

Our product evaluation was mostly centred around play testing. Since the requirements specify user-facing features, the best way to evaluate whether the brief has been met to a satisfactory standard would be to play the game with the intent of experiencing each requirement as any regular player would. This also allows the person testing to make sure the final product doesn't suffer from unexpected bugs or mistakes which might otherwise be missed, like a graphic being displayed incorrectly, or the game crashing due to interactions between systems in the complete product.
The tester looks at each system requirement in turn and carries out appropriate steps in-game to see whether the requirement is met. For a simple example, there is a requirement that the player's ship must take damage when it is hit by an enemy. The tester would let themselves get damaged and check that their on-screen HP meter decreases, and make sure that when the meter gets to zero the game over screen is displayed. If everything functions as expected, the requirement can be considered to be met.

Requirements were constructed based on the original design brief [1] and discussions with the client both in person and through email, later being updated to account for additional requirements introduced in assessment 4. The list of requirements was given to the client to approve to ensure that our ideas for how to complete the project matched theirs. We used the requirements document as a checklist of completeness for the project, highlighting unfulfilled requirements in red, partially completed requirements in orange, and completed requirements in green. Only one requirement was left partially completed by the end of the project (requirement 5), which will be discussed later in this document.

Testing Report

In order to analyse the quality of the final product accurately and appropriately, we need to establish what the main aspects of a good quality software project are. In Software Testing and Quality Assurance: Theory and Practice[2] by Naik and Tripathy, it clearly listed 6 characteristics that good quality software should have: functionality, reliability, usability, efficiency, maintainability and portability. We focused on four of these six characteristics, as they are the most appropriate for our project and are essential features of a successful and good quality product:

1. Functionality: The software should provide an adequate set of functions for specified tasks and user objectives
2. Reliability: The product can be expected to perform its intended functions as the designer expected.
3. Usability: The product is considered as usable if human users find it is easy

to use, which means that the product needs intuitive user interfaces.

4.Maintainability: The code was built to be modular, extensible, readable and easy for future groups to maintain the project.

Before we started to develop the product, we have listed all the user requirements, system requirements. And we also updated the requirements to fit the new changes from the “customer”. We have cross referencing the project requirements when we developing the final product to make sure our product fits every requirements and constraints which also ensures the functionality of the product.

For evaluation of the reliability of the final product, we designed the Black-box tests that would make sure that every function that we designed are working as we expected. We also re-testing all the black-box test that the groups has tested before to insure the reliability of the product. The detailed testing result will performed in the next section of this report.

To access the maintainability, we reviewed all the codes to make sure that they are documented and commented to ensure that the future developer can easily understand the code. The software is modular and extensible and built around classes with specific functionality. We also keep update the implementation documentation to track and record new changes made to the product.

In order to access the usability, the test controller of the group navigate the game from the start menu to the end of the game, checking if there is any difficulties when playing the game. And then, we can make more features to make the user experiences better.

Testing approach

From the testing report from Team 1, it shows that the developer from Team 1 still remain to use black-box testing for the product. So In the assessment 4, we also continue to use this testing approach we used in the previous 3 assessment. Black-box testing will still be the main testing method since it allows use to test and functionality and usability of the product and it can imitate the environment observed by the player to help the developer to handle the error that the client may have.
At the first, we re-ran all the tests from the first 10 tests to check if all the basic code performing as we expected after Team 1’s implementation and changes. In addition to this, we also established a new section in the testing material which is called “ Additional changes”. The new requirements the client demand need new tests to ensure that all these features are playable and the final project has met the requirements. So the table contains the testing we design for the new features and all the testing which we absorbed from Team 1.

The result of our testing are presented as a table where each test has an ID and the requirement they met, an expected result and the actual test result and evidence. The result of each test has been highlighted in 2 colors to indicate the test status, red means “Fail”, “Pass” in green.The test evidences of the tests which have been re-ran will directly shown in the table instead of a link to the screenshots to make the material more readable and easy to update.
After all the new features implemented and several testing, we wrote 21 black-box tests for the final product. 100 percent of these tests have passed which insure the functionality and usability of the final product.

All the testing materials has been documented and shown here for the future developers to update:
Test report (assessment 2): https://team-ipatch.github.io/Assessment-2/Test2.pdf
Changed report in assessment 3: Change3.pdf
Final testing material:
https://drive.google.com/open?id=1tkHYSX8JvK_rlGDPg0540mspsZ-MpHawZwJcDVGFhLg

Requirements

The requirements document can be found at
https://team-ipatch.github.io/Assessment-4/Req4.pdf

The software meets most requirements fully, with the biggest exception being the lack of proper pause functionality [**req 5**]. There is no button to allow the player to pause the game at will, but enemies stop chasing the player if they come within range of a department shop. This doesn’t achieve true pausing (fired projectiles continue to fly) but it gives the impression that the game stops while the player is in a menu. This requirement isn’t fully met since the game does not match conventional understanding of what pausing a game does, but it serves as an anti-annoyance measure that allows the player to interact with a menu without the threat of enemies attacking them while they do so. The software fails system requirements **5.abc** because there is no way to manually pause the game and no options menu is present.
The software meets the requirement that the map contains islands [**req 9**], but with a caveat. Most of the islands in the game are physical obstacles that block the player, while the “islands” the player interacts with are game entities without collision physics which cause a menu to appear when the player gets close enough. This implementation doesn’t fully match the modular and extensible islands we assumed we’d make, but it allowed us to implement random terrain generation for more unpredictable repeat gameplay.

The fit criteria for [**req 11**] is also not met, as the game was not built around a level based structure but rather around a randomly generated open level with objectives scattered around it. This difference did not seem significant enough to fail the requirement on however.
The rest of the requirements are met to a satisfactory degree.

References

[1] University of York, “SEPR Scenario 2.” [Online]. Available: https://vle.york.ac.uk/bbcswebdav/pid-3045337-dt-content-rid-7484111_2/courses/Y2018-006404/SEPR%20Scenario%202%2C%202018-19_%20York%20Pirates%21.pdf

[2] K. Naik and P. Tripathy, *Software Testing and Quality Assurance: Theory and Practice*. John Wiley & Sons, 2011.