Team iPatch

Assessment 2: Software Testing Report

Christian Pardillo Laursen
Filip Makosza
Joseph Leigh
Mingxuan Weng
Oliver Relph

Testing methods and approaches

Throughout the implementation of the testing project, we choose to test the code by using mostly the black box testing and some of the elements of white box testing since white box testing is not very helpful in game development, where creating test cases is very laborious as events are triggered in all sorts of ways and in many cases handled by the engine, outside of our control. We ran the test on each feature and the majority of class outcomes.

Black box testing

Black box testing is a test that examines the functionality of the code without considering the internal structure. Black box testing can be done from the user's point, and we can simulate the action the user may execute in the game. We can test the function by simply playing the game and check whether the function is implemented or not. After the feature is implemented, a series of black box testing is needed to check if the feature is work as we designed. And the test can imitate the environment observed by the player.

Black box testing is also very useful when the person who is running the test is not a person who wrote the code. Since each team member is playing different roles in the team, not all of us contribute to the internal code, so the other members can run the black box test effectively. But the disadvantage of black box test is obvious which is that we can not accurately imitate every single step of the player so there may be some problems that we can't discover. To deal with this issue, we define the test precisely but may still miss some obscure scenarios.

White box testing

We use the white box testing to test the internal routine, code structure of the game. It can be taken in the earlier stage without waiting for the GUI to be available. The tester of white box testing, usually the developer, produce test cases based on the analysis of internal structure. In this stage, white box testing is essential to make sure there isn't internal error in the code or design. It can make sure the function that we have already implemented will not be impact by the new features. White box testing sometimes will be taken after the black box testing failed to find what is the error inside the code. It will improve the quality and efficiency of the software and ensure the consistency of each procedure.

Testing Analysis

Black box testing

We plan to run the black box testing on all the functions and features of the project. The main approach is doing functional testing based on the requirements we made to test that specific functions are working well. And the project will follows the Test Driven Development, so we will run the game periodly to check whether the feature will be presented as we designed.

Result

Since the project is based on the Test Driven Development, in the final black box testing, only one of the ten tests failed. Although most of the functions are working as we designed our code base still has a few problems. We have imitated the action that a player may have and without considering the actions of a player during run time. For example, in test 9, we assumed that the player would restore their health until their health points were max. But they may still press the restore button (which will still cause a loss of gold) if they are not aware that their healthpool is already full since there is no notification to tell them about that. So the tests still have a lot of limitations. Later a peering test or a actual player test may take place to solve this problem.

White box testing

We planned to use Junit test before we started to implement the function but it did not work well. So eventually decided doing the white box testing manually by looking at the code periodically to check if there is any structural errors that we did not anticipate and when the function was not implemented well.

This approach cost quite a lot of time, in the future a alternative way should be used in order to check the functions are still functional when the other features being implemented.

Result

As we did not see the merit in using JUnit in an action diven game the majority of our internal testing was done during bug fixing and stress testing. On many occasions the groups programming team came together to discuss their code and edit it to increase code integrity and the quality of life for the player during run time. Due to the lac of unit testing there may be some specific and obscure cases in which our code may fail or struggle due to oversights of time complexity and hardware limitations, however we tested our code as extensively as we believed necessary for the purpose of this task.

Linking

The test material will be displayed in the test evidence.

Link:

https://drive.google.com/open?id=1A-2Ji7d7BoDALjGVbG37mUdDoEgX_X82

When the test is failed, it will be labelled Fail and be highlighted by red color; if the test pass, it will be labelled Pass and by green color.

The related requirement is based on the updated requirement file.

Link:

https://theshutupman.github.io/Team-iPatch/Assessment-2/Req2.pdf

The testing evidence will be presented as pictures (screenshot while testing).

Link:

https://drive.google.com/drive/folders/1PYDl0uw1bynkSeSN5uiLeJMb0U44Kadh