Introduction:

NPStudios (whom we inherited from for this assessment) made several changes to DicyCat's testing (whom they inherited from in assessment 3). They removed the debug functionality because it caused a memory leak, they say they used playtesting however there is no record of playtesting being updated. Additionally, they used a structural testing approach to achieve a stronger code base; this was documented and was done in the form of JUnit testing. DicyCat's traceability matrix and test tables documenting tests were also updated accordingly by NPStudios.

NPStudios testing methodology was more structured than DicyCats, which made it more suitable for us to take over. However, some of the Unit tests NPStudios added have problems. Some of the test classes are empty and do not do anything. These include; PatternTest.java, FortressTest.Java, FireTruckTest.java, FireStationTest.java and BulletTest.java. Due to this reason, a lot of requirements were not properly tested and had to be tested using other methods. Upon asking NPStudios for the missing tests that they documented, they claimed that they were in a different branch on Git, however the tests never worked with the game and attempting to salvage the broken tests was deemed ineffective as the classes that were tested by the broken tests were more suitable for blackbox play testing.

It was decided by our team that we will adopt the testing methods used by NPStudios, which was their structured testing approach and JUnit testing as a form of WhiteBox testing. However, we would also update the BlackBox Play testing that NPStudios failed to update for Assessment 3. We came to this conclusion because we deemed their testing to be decent, however the broken tests files meant it was difficult to reconstruct those tests and the team thought it was appropriate to vary our testing and since the previous team had not continued with their BlackBox Play Testing, we thought it was appropriate to continue with it and update it accordingly.

Methods for evaluation

, evaluation refers to how you determined that the product met its brief;

To ensure that our final product met its brief. We used several methods. Firstly, we made sure to implement every detail of the game's brief into the requirements. The requirements were constructed using the brief, and were constantly updated during each assessment to meet the product brief. We made sure to include even the most basic of requirements, such as UR_FORTRESS of which the description says, "The game should have fortresses". This allowed us to ensure that the game came as close as possible to the brief. Some requirements were also updated to better fit the brief, for example both UR_DIFFICULTY_LEVEL and UR_PATROL were updated. Our testing took the new updated requirements into consideration and tests that involved those requirements were updated accordingly. Black box play testing and Java Unit testing was used to ensure the requirements had been fulfilled.

NP Studios, the team who oversaw this project before us, did a respectable amount of testing for the requirements so far. To ensure their tests were sufficient, we repeated them and then made sure to add any necessary tests for any requirements NP Studios may have missed out before we added our additional requirements and tests for assessment 4. As most of their new testing was JUnit testing, these tests were easy to reproduce. The BlackBox play testing had more room for error to reproduce but we ultimately got the same results. As we felt the tests were sufficient, we left them in and added more of our own tests which reflected the new requirements we had implemented. The reason we left in NPStudios play tests was because it was believed that their tests tested a respectable amount of the requirements when we received the project. It was believed that additional tests should only be added to test the new requirements. However, due to the nature of their broken JUnit tests, we not only tested the new requirements relevant to assessment 4 but we also tested the requirements that were in the broken JUnit tests. For example, the alien patrols all had to be tested again due to the broken nature of the JUnit file.

For the Blackbox play testing, we updated them in a way which allows the product owner to see any pre-existing tests. This choice was made as we deemed the pre-existing tests to be of sufficient quality so the tests that we added were based on our additional requirements for assessment 4. We determined they were good enough after redoing the tests and getting the same results. Tests that involved modified requirements were taken into consideration when updating the testing so they would better fit the requirements and therefore the brief. We added a relevant test for each additional requirement that was added in this phase of the assessment. We tested all the new saving functionality requirements and the powerups with play testing as we deemed this to be the most effective method, as it was clear and easy to see if it met the requirements.

For our Java testing, everything was documented and put into a “Test Design and Documentation” document detailing the results and how we executed our java tests. Alien and firetruck tests had to be removed from the document as those tests were empty when we had received the project. The tests that were there were of substantial quality which is why we decided to keep them. All the Java tests had requirements they were based off which meant that if the java tests had passed, we had met the requirements. Consequently, this meant we could effectively see if the product met its brief through the passing of the tests.

We mapped our testing against our requirements so we can see which test maps to which requirement. This allowed the team to track how often each requirement is tested, which therefore allows us to see how effective we were at testing and see if we have met the brief.

Overall the team thought our method for evaluating how well the product met its brief was effective. The brief was translated to the requirements which was then tested to see if we met the requirements.

We also, as a team, viewed our finished game in a broader sense from a heuristics point of view. This was influenced by our work in our first year module, Human Aspects of Computer Science (HACS) as well as [1] “Using heuristics to evaluate the playability of games” however this provided more of a springboard to inform a more informal approach. We evaluated our game in a brainstorm structure, and concluded that we felt that it suitably met the brief set and is suitable for the target user of open day pupils. This more informal addition to our evaluation allowed us to ensure that the formality of the game’s requirements had not skewed the enjoyability of the final product. We particularly like our game’s lack of wait time between user goals and actions, largely due to the real-time strategy approach.

Methods for Testing

Our main method for testing the integrity of the game files was using WhiteBox testing. The WhiteBox testing approach we used was JUnit testing for Java. We used JUnit testing in our previous assessments and can confirm from our previous assessments that using JUnit testing guarantees good code quality and functionality. JUnit tests will not work as intended if the code quality is poor and must be refactored, we made sure the Unit tests tested classes that were important to the core functionality of the game.

Our strategy for WhiteBox JUnit testing was to continue to use the JUnit tests that were already there. The previous team had missing Unit testing and it was decided by our team to test those specific requirements that were missing using our Blackbox playtesting. Generally, our JUnit testing involved just adding tests relevant to assessment 4 as the previous team did a robust job with the tests prior and tested enough code. The tests we added to the existing JUnit testing were tests to test the functionality of the powerups. As testing powerups involved modifying fire truck specifications, we deemed it appropriate to test this using JUnit Whitebox testing as we can see if the firetruck values are modified. This allowed us to properly see if the powerup mechanism in the game was functional while simultaneously allowing us to have reproducible results. All the JUnit documentation was added to the Test Design and Documentation file.

Our WhiteBox testing can be found here:

Our Test Design and Documentation for our WhiteBox testing can be found here(additions are highlighted in purple):

For our BlackBox play testing, we decided to continue with the tests that were already there. Our approach was to test the requirements that were not covered by our JUnit testing within the blackbox testing. Some requirements were better suited to being tested in play testing as they could not be shown effectively in JUnit testing. For example, testing whether the game can pause is much more effective to do in play testing as opposed to JUnit because it's easier to see visually that the game has been paused rather than testing the code. Initially, our testing was only going to involve testing the new requirements for assessment 4, however, the team before us failed to provide testing files for some of the previous assessments and it was decided those tests were best suited to be tested in Blackbox playtesting. For example, Patrol_Attack was better suited to play testing as we can physically see the patrols attacking the fire truck and the fire trucks taking damage. Generally, our playtesting focussed on what couldn't be tested effectively in the Java tests.

Our BlackBox testing can be found here:

For the Traceability matrix, it was decided by the team to combine the functional requirements traceability matrix and user requirements traceability matrix into one large traceability matrix. This allowed us to better track how many times a requirement was met giving an indication to how effective our testing was.

Our Traceability matrix can be found here:

Testing Results

We decided to put the Blackbox play testing results and the Whitebox JUnit testing results in separate files as they followed a different structure which was unsuitable for one table. We made sure to give each test a unique identification so it could be easily traceable in the traceability matrix, a short description so the game manager knows what the test does, a requirement ID to show which requirements the test relates to and a Pass/Fail test result.

From our whitebox Unit results, 33/33 tests passed, this was done using Java's JUnit testing. We decided to continue with NPStudios JUnit testing as JUnit test results are easily reproducible, they can easily test if we have met specific requirements which helps us know our game has met the brief, they ensure that the code of is of quality and factored properly

From our blackbox play testing results 30/30 tests passed. Our play testing focussed on testing that was deemed not suitable for JUnit testing. We made sure to document our tests and detail them with a short description describing what we did to make them as reproducible as possible so anyone who wants to follow our tests gets the same results we did.

The total number that an individual requirement was identified in a test was 346. From this figure we know that there was enough testing to ensure the game met the brief and was of outstanding quality.

Requirements Evaluation

Comment on how your product meets, and does not meet, the requirements. Include precise URLs for the web pages where the final requirements can be found. (5 marks,≤ 1 page)

We updated the requirements document to include requirements for assessment 4. We added user requirements to fit the new features we are supposed to add into the game such as requirements for the save game features and requirements for the power ups. We also added some basic requirements that were missing from the project when we received it to help give us a more accurate brief when we did our testing. Some requirements were also modified to help better fit the brief.

From all our requirements, each requirement corresponded to at least 1 test. The only requirement that was not within a test was UR_MOBILE as throughout our testing there was no effective way to test “The game may be cross platform transferable” in our play testing or our java unit testing. In defence of this, the game is cross platform transferable as it is built on LibGdx therefore we have also met this requirement adequate for this assessment, as it was not ever directly addressed by the customer as a final requirement. We didn’t think it was responsible to develop a new testing method so we could meet just 1 requirement that has a low priority.

Our final product generally does a good job of meeting the requirements. Upon conducting our Blackbox and Whitebox testing, we made sure to only reference those requirements that could be met as a result of the test. This way our Blackbox and Whitebox testing could automatically test if the requirements had been met. Upon looking at the traceability matrix we can see that each requirement has been tested at least once (apart from UR_MOBILE). The total number of times an individual requirement was identified was 346 according to the traceability matrix.

The final requirements can be found here:

Bibliography

[1]

M. C. J. A. T. Heather Desurvire, "Using heuristics to evaluate the playability of games," in *CHI EA '04*, 2004.