Module	SEPR
Year	2019/20
Assessment	2
Team	Salt N Sepr
Members	Alberto Lee, Archie Godfrey, Jacqueline Eilertsen, Jake Schoonbrood, Joshua Levett, Matteo Barberis
Deliverable	Testing (Test2)

Software Testing Report

Testing software prior to release is essential to ensuring users avoid negative experiences as a result of errors or defects when using the software, and although it is difficult to verify the software can handle *all* possible eventualities, the employment of various testing models can enable developers to minimise and mitigate these faults before they impact user experiences, and give feedback to assist in making decisions around further development around quality.

The ISO/IEC/IEEE 29119 standard[1] proposes that testing is a process; it should be preplanned, but continue throughout and beyond the development of software. For this project, this model is adapted such to make it easier to follow for the smaller number of developers involved, and simpler given the relatively minimal complexity involved with the scale of the project when compared to that which the standard was designed.

A key principle to the testing model used is that it is *agile*, and so can be used in conjunction with the SCRUM-based development cycle. One recommendation for agile testing is that as much of it should be automated as possible[2]. By using Java as the primary development language many potential faults are removed through the use of *static typing*. One particular difficulty with using the LibGDX framework and libraries is that it is not trivially compliant[3], [4] with JUnit5[5], a white-box style unit testing framework for Java projects. It instead requires significant additional configuration and a *headless* implementation due to LibGDX's dependency on OpenGL for rendering components.

It can additionally be difficult to ensure code reliability when developing in an agile way, particularly without unit tests, as an increased number of smaller updates can lead to a higher number of opportunities for the overall software code to fail (such as through concurrently modifying a feature and its dependency). Regular regression testing must therefore be performed throughout the project to catch errors or bugs arising from these code updates.

Not all errors can be caught through regression testing, and therefore it is important to have additional points of software verification and testing throughout the project. By performing a *code review* it is possible to catch significant issues in code prior to their implementation in the main program - done practically in this project by requiring a different person to approve the merging of branches than the developer committing the changes.

Furthermore, at the point of developing each feature within the project, a new series of tests should be written that exercise the feature as a replacement for unit testing which could otherwise have been performed.

Finally, black-box validation and defect testing can be performed at key points in the project, such as at submission points. Much of this can be done internally, with final smoke testing and acceptance testing performed with the client and users representative of the target audience. As part of this, there was a meeting held with the client on Tuesday 7th January.

Detailed documentation, such as Code Standards, the tests performed and documentation can be found on the Salt N' SEPR website.

Testing *Kroy*

It is not practical, or in many cases possible, to exhaust every possible scenario when testing a developed project. This is additionally true for the *Kroy* project as a result of the minimal use of unit testing because of non-trivial incompatibilities between the two frameworks which increase the importance of effective and thorough testing by visual inspection - the prominent method adopted by the project.

Testing was run frequently on the project, often by immediately implementing methods created and utilising the functionality. A series of formal tests run on the project are available at https://sepr-documentation.firebaseapp.com/testing with an outline of what tests have passed and failed, as well as their relative severity relative to the game.

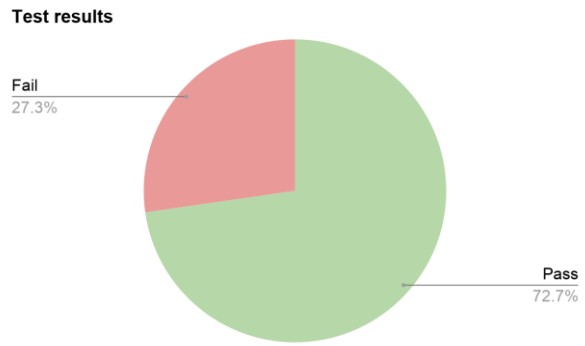

Figure 1: Passed and failed tests

Although many of the tests performed have reached the *Pass* status, there exists a significant proportion which have yet to do so. On average, the severity (scale of 1/*no concern* to 9/*severe*) of these failed tests is *4.22*, meaning prior to the final release, many of these tests must succeed.

In many cases, this is because the feature has not yet been implemented, as tests for the initial and developing project requirements have already been written such that it is possible to gauge to some extent progress through implementation of the project. These include tests around *alien patrols* - a user requirement (with accompanying functional and non-functional elements) yet to be implemented, but producing a *Fail* outcome for a collection of tests, including *T.U.008*, *T.U.011* and *T.U.012*.

Similarly, tests also fail for final game requirements not yet met by the game, such as ending the game once six *ET Fortresses* have been destroyed by the player. As per the requirements for this *Assessment 2* submission, only three have been implemented, and so *T.N.003* fails at this point. At a later point in development, when this has been implemented, this test will pass.

A test of the options available on the initial *Menu* screen displayed also fails as it should display the options “Play Game, Leaderboard, Minigame, Edit Settings” but instead has a subset of these (“Play Game, Leaderboard, Exit Game”). An additional button on this screen would, however, be relatively simple to implement - but at this point the *Minigame* has not been developed. Furthermore, at this stage in the project, a *Constants.java* file has been created containing many

of the configurable items that would have been available to the user, and so a *Settings* screen has not been added, although it could be added at a later stage.

T.N.006, a test of the cross-platform design of the game, also does not pass. The objective of the test is to ascertain whether the game can be ported to mobile platforms (a feature to consider, as requested by the client), however the test has not yet been attempted.

Testing can never be complete[6]. If a point was reached where all tests for the project pass, it would likely be a sign of insufficient testing as it is not possible to test all possible inputs, all game logic (particularly with the use of third-party libraries), all possible paths, and failures due to user interface decisions or insufficiently detailed requirements. It is therefore the case that although through further development and implementation of the project requirements the number *Fail* test results may reduce, there should be no single action that produces a *Pass* status for all tests. Should this ever be the case, it is an indicator more tests should be written.

In an effort to ensure tests for the project are effective at ensuring it meets the initial requirements, there exists a *Traceability Matrix* https://sepr-documentation.firebaseapp.com/testing to assist in determining the completeness of the testing performed. This ensures that for each *User*, *Functional* and *Non-Functional* requirement there exists at least one test. This does not, however, eliminate the possibility of insufficient testing, such as testing the way parts of the project have been implemented, but instead only that the initial and extended requirements have been met.

Bibliography

[1] 'ISO/IEC/IEEE International Standard - Software and systems engineering –Software testing –Part 1:Concepts and definitions', *ISOIECIEEE 29119-12013E*, pp. 1–64, Sep. 2013 [Online]. Available: 10.1109/IEEESTD.2013.6588537.

[2] M. Costick, (Nov. 26, 2013), 'Agile testing at the Home Office', Government Digital Service Blog. [Online]. Available https://gds.blog.gov.uk/2013/11/26/agile-testing-at-the-home-office/ [Accessed: 02 January. 2020].

[3] thisisrahuld, 'Unit Testing Libgdx applications', *BadLogicGames Forum*. Jul. 27, 2015 [Online]. Available https://www.badlogicgames.com/forum/viewtopic.php?f=11&t=20103 [Accessed: 01 January. 2020].

[4] T. Pronold, *TomGrill/gdx-testing*. 2019 [Online]. Available https://github.com/TomGrill/gdx-testing [Accessed: 01 January. 2020].

[5] *JUnit 5*, JUnit. [Online]. Available: https://junit.org/junit5/ [Accessed: 02 January. 2020].

[6] C. Kaner and R. L. Fiedler, *Foundations of Software Testing*. Context-Driven Press, 2013.