Evaluation and Testing Report

Team 12

April 2020

1 Approaches to evaluation and testing

1.1 Evaluation

Kroy 3.0 was developed with close consideration to product brief given in week 1 of the first assessment cycle and further communications from the customer. A main method that was used to evaluate the extent to which the game successfully met the brief was by considering it directly in comparison to requirements and the further functional and non-functional requirements derived from them. This method of game evaluation was considered an effective method of ensuring the game brief was met as we could formally test that all requirements were being met (using manual and automated testing), strongly inferring that the customer was getting the product that was requested. A clear link between requirements and tests were logged in the traceability matrix. By using this method of evaluation we could confidently presume the the game would check all the customers boxes as long as all the requirements were accurate and all the users desires were represented in one or more requirement. Further discussion on how these tests were carried out and to what extent the requirements were met are discussed in the following sections. We could guarantee, with a high level of confidence, that all requirements were formally written up due to the constant feedback loop between the customer and the software development team. Due to the nature of the assessment, we were able to inherit requirements from previous teams which had been reviewed twice by the customer. By considering the feedback that these teams were provided we were able to establish if there were any existing issues with the requirements as they were. Additionally, we were able to communicate potential requirement changes with the customer and clarify their needs on issues that were more ambiguous through meetings and email communications. This feedback loop also allowed us to confirm more subjective requirements such as UR_ENJOYABILITY. This requirement was evaluated as not being sufficiently met by our developers due to the simple and unchallenging game play that was inherited for this assessment cycle. Due to the teams gaming experience,it was decided that an appropriate and captivating way of adapting this game play was by turning it into a Stealth game. This was proposed to the customer, and was approved with the caveat that the game play should not diminish the look of the game, which the customer felt already met the user requirement UR_ATTRACTIVE.

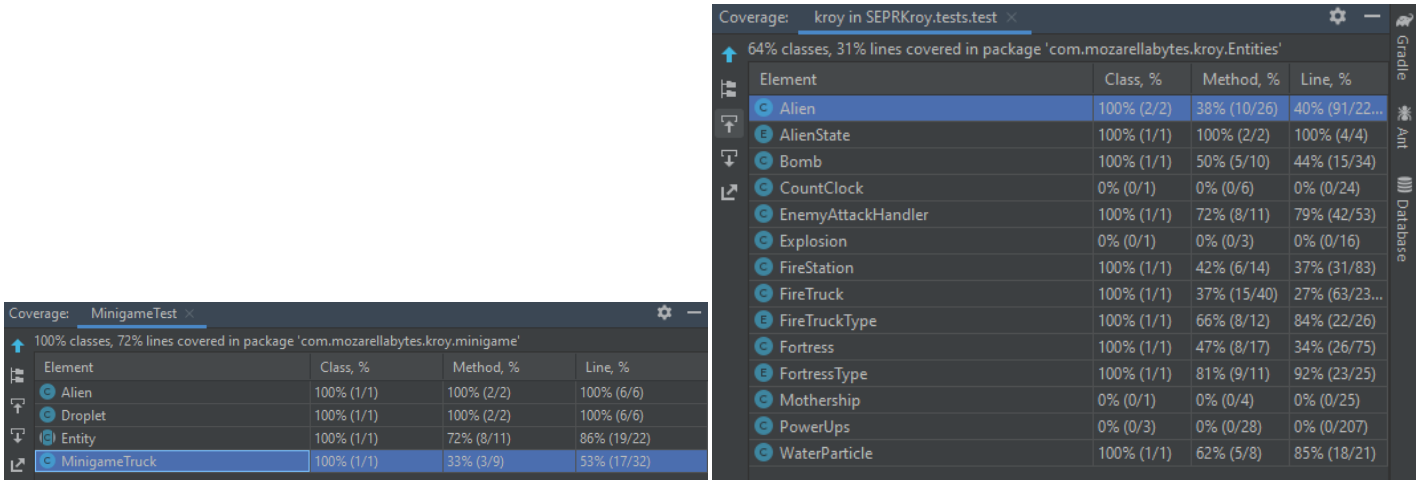
1.2 Testing

Our testing methods evolved throughout the assessments adapting to both the teams strengths and weaknesses as well as adapting to better accommodate the methods used by previous teams. To begin, we put a strong emphasis on using a combination of black box and white box testing in order to encompass a larger range of testing and hopefully covering more potential software issues by doing so. However, in this last iteration, there was little mention of the type of testing carried out by the previous team besides mentioning the use of both automated and manual testing. Due to the large amount of testing already implemented and designed successfully by this team, we decided to not restructure or rethink the testing design to more clearly separate black box and white box testing and instead adopted this teams method of considering only manual and automated tests during our own testing design.

1.2.1 Automated Testing

The testing suite inherited by Sepret Studios had high testing coverage and pass rates as well as a strong relationship with their user requirements as can be seen in their testing report (found here). For this reason, testing was focused on the changes that were introduced by our team in the final iteration of this project as well as the new requirements communicated by the customer.

We elected not to add any more automated tests for the new requirements we got from the updated brief, as we felt they would be better suited for manual tests. Furthermore, we did not add any new tests for the previous groups code as we felt their test coverage was sufficient and we did not make any large scale changes that would justify replacing their tests. However, we did update all their tests to ensure any changes to the code that we made were still in line with their testing.



Minigame test was modified to encompass changes made to the minigame. This included balance changes to speed of aliens and truck speed increase. Coverage remains high, though the entity and minigame truck elements look low that is because these contain a lot of getters, setters and constructor methods which do not require testing.

Entities, on the other hand, has had much more significant changes and additions. This has resulted in a significant loss of coverage from automated testing. Changes include, but are not limited to, the addition of PowerUps, re-creation of FireStation, AI of Alien, balance changes to Fortresses, as well as many more minor changes. No new automated tests were added to reflect the large changers, but the old tests were updated to match the new classes which possible. Instead of creating new JUnit tests, manual testing was used to test any new or major changes, which is not represented here.

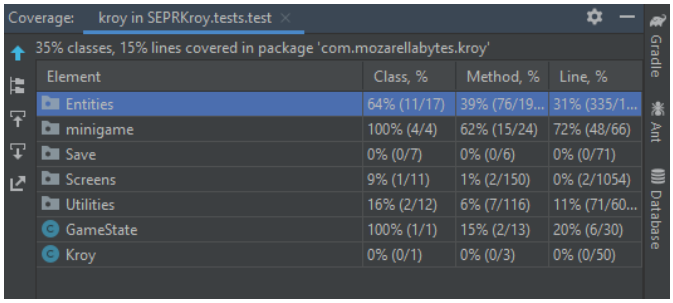
1.2.2 Manual Testing

Manual tests were used to test elements that might otherwise be difficult to test using automated tests. They were also used to test all our user requirements had been met, as this was our final iteration of the product and we wanted to ensure our program was aligned with our original requirements.

All our new manual tests are documented here. We tested each individual user requirement, excluding those too subjective for conventional testing. Manual tests were designed similarly to automated tests, however they have a specific set of steps the tester must follow to run the test. We chose not to run separate tests for UR_SAVES, UR_POWER_UPS, UR_MANY_DIFFICULTIES and their derived functional requirements as we felt they would serve no benefit and only add redundant tests. Instead, we tested the user requirement with the assumption that if the user requirement passed, the functional requirement should pass too.

1.2.3 Line Coverage

The line coverage from automated tests comes out at 25%. This value was achieved by ignoring the screens package as it is better suited for manual testing, as well as the class Kroy, which did not need testing. The value does also include getters and setters which have no test, and therefore bring the value down even though they do not require testing. However, even excluding getters and setters, the line coverage from automated testing would still be low. A low value was to be expected though, as we filled in a lot of the gaps in testing using manual tests as mentioned previously. In the figures below, you can see which packages have been tested. minigame received a large amount of automated testing as we did not make many major changes to it beyond balance changes, so the previous groups tests were still good for coverage. Entities, Utilities and GameState all had significantly lower line coverage as we made major changes to those classes without adding automated tests to match. However, we performed many manual tests to account for this so the line coverage for those classes and packages isn't as accurate as it appears.



When the number of manual tests we performed is considered, 25% seems like a much more reasonable value than it first appears. So we believe the tests carried out sufficiently cover enough of the code to ensure our software is reliable. With the combination of automated and manual testing, we have covered and tested the large majority of the customers functional and user requirements.

Though testing provided us a strong degree of certainty that the code worked, to determine that our code was of appropriate quality, we used the ISO/IEC 25010 standard[1]. This standard allows software developers to assess the quality of their code based on 8 factors: functional stability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability. Due to the nature of our software some of these factors were not/less applicable than others but sticking to this standard allowed us to view our code in a reliable and standard way that could easily be understood by other professionals. The table below shows how we rated our software in comparison to this criteria, using the results yielded by our testing as well as general knowledge on the game design and documentation.

Factor	**Meaning**	**Comment**	**Rating**
Functional Suitability	Degree with which our code provides functions to fulfil stated and implied needs of the user.	Development occured with close communication with the customer as well as strictly attempting to fulfil all formal user requirements. The degree with which this factor was met is directly related with the success of our testing suite. All user wishes fulfilled, little room for improvement	8/10
Performance efficiency	Performance of the software relative to amount of resources used	Due to the nature of the software the major potential performance pitfalls was graphical rendering to display the constantly moving entities and storage to store saves. Software was tested on basic computer with normal computational power and did not lag nor eat all the CPU time. Additionally, a limitation of only allowing 3 save slots in the game ensured that storage also remained minimal	7/10
Compatibility	Degree with which our software can perform its function while sharing the same hardware/software and share information with other systems/processes	This criteria is less applicable to our software as it does not require any communication between other processes, however the software is able to share hardware software somewhat as it does not eat up processing time. Room for improvement by considering this criteria better curing further development.	6/10
Usability	Degree with which our software can be used by specific user to achieve specific goals effectively	Our software is branded for a young audience and makes that clear by using bright colours and fun music. The accompanying product website has clear manuals making it easy for the user to learn and achieve the goals. Clear, large buttons are used to guide users to the game and through the controls. The range of difficulties allows users of all types to enjoy the software. Room for improvement in preventing users from making errors by adding pop ups to check if user really meant to press quit for e.g	8/10

Factor	Meaning	Comment	Rating
Security	Degree with which our software protects information and data depending on levels of authorization	Software contains no sensitive data and therefore requires little security. Room for improvement in requiring authentication for users to continue game from a specific save (a different user may try to play from someone elses save), however, out of scope for this type of arcade-style game play.	8/10
Maintainability	Degree of effectiveness and efficiency with which our software can be modified	The amount of documentation kept along side this software and the clear code with appropriate comments to explain functionality allow it to be easily modified. Room for improvement by introducing a Term Glossary which would prevent confusion about use of terms and words.	8/10
Portability	Degree with which our software can be transferred from one software/hardware to another	Having been written in Java, our software can easily be run on any machine. Gameplay was also designed with the possibility of migration from desktop to a mobile device.	10/10
Reliability	Degree with which our software performs specific functions under specific conditions for a specific period of time.	Software can perform its intended function for the duration of the game specified by user requirements (<15 mins). Room for improvement investigating occasional faults.	8/10

2 Requirements

The inherited documentation from the previous team included user requirements which were considered coherent with the brief. This was established by comparing the requirements with the brief and additional communications with the customer. 2 new user requirements were added to reflect more subjective requirements that our software development team deemed core game requirements, these were:

- UR_PG13: emphasising the clear communication from the customer that the game should be appropriate for younger audiences, a key requirement to keep in mind during development.
- UR_ATTRACTIVE: emphasising the desire for the game to be 'attractive to the eye'.

These new requirements, were testing in the same manner as UR_ENJOYABILITY and UR_SCALABILITY which, due to their subjective natures had no formal tests written for them. Instead, feedback by the customer during previous development cycles as well as our own experiences and research was used to determine if the game met these themes overall. For example, we ensure the game was family friendly by playing through and insuring no graphics or words were explicit as well as using general knowledge about younger audiences preferences in game play.

In addition further requirements were added to reflect the requirement change provided by the customer. These included:

- UR_POWER_UPS: game contains 5 varying types of drops.
- UR_SAVES: game must have a way of saving game state and be picked up again at other time.
- UR_MANY_DIFFICULTIES: game must have a way of choosing game level difficulty.

The full new list of requirements can be found here. The testing methods inherited by Sepret Studios were similar to that used by our team in the previous iteration allowing us to easily inherit and understand the procedure. Their testing encapsulated all major functional requirements and resulted in high test coverage and pass rates. This testing report (which can be found in full here) allowed us to be confident enough in the testing of those aspects of the game. Therefore additional testing was performed primarily on missing user requirements and additional functionality introduced during this iteration. Again, these tests provided a good level of coverage and pass rates, the reports of the new tests are also included in the testing report previously linked and specifics discussed in the previous section. Overall the testing showed that requirements were being strictly met, the only exception to this was for UR_SAVES. This user requirement was added by us in the last iteration of development based on the requirements change. Though the major part of this requirement was met, users are able to save their game and select their save game to return to when restarting the game, our team made a design decision to not allow the user to save the game during the mini-game. The reason for this change was due to the nature of the mini-game. As the mini-game was a Space Invaders-like game, the constant stream of aliens descending the screen and their speed provided an unrealistic challenge to a user picking up the save game in the middle of this type of attack, likely resulting in an immediate death for the user.

References

[1] ISO/IEC. Iso/iec 25010 system and software quality models. Technical report, 2010.