SENG201 - Software Engineering - Island Trader Game
Project Report
18/05/2021

Ryan Boyd - 47284216
John Elliott - 98040483

Project overview
For this project we have created a java application. Our application is a game where the user plays as a trader who travels by ship between Islands, trading goods at different stores to try and earn a profit whilst avoiding danger like storms and pirates. Our application was built using javascript code primarily in the Eclipse IDE, making use of such tools as Windowbuilder to create a graphical interface for our application to be played with.

Project structure
The game has lots of consistent generation that is made every time it is played, such as Islands and Items. To not overwhelm the GameLogic class which worked to determine key game events. We decided to create a static final class SetUp which allowed for the initialization of all these components. The GameEnvironment class worked as our game GUI manager and is responsible for opening and closing screens during the game and passing the current GameLogic class to each new screen. This allowed for effective transitions between screens.

As both the Upgrade and Item classes were able to be brought from the Store class we implemented a parent Purchasables which these classes used to inherit key methods and properties. More inheritance was used during the creation of the GUI as many custom listeners and the table generation inherited or implemented the Swing classes and interfaces. This allowed for unique systems which helped to make the game more simple and less error prone.

Unit Test Coverage
Unit test coverage is a measure used to describe the amount of code of a project that is run properly during its execution. Measured in percentage - an application that has a higher percent of its total source code run in a test suggests that there is a lower chance of undiscovered bugs.

Our Project in the end had a Junit coverage of 49.6% on the main code, this could be considered lower coverage in a professional coding environment where the goal is usually to reach as close to 100% coverage as possible. The main reasons for lower test coverage involve having rushed testing as it was left towards the end of the projects timeline, without much room to do more detailed testing covering a wider range of the projects main functions. This also led to less detailed tests that do not fully test the functionality of the methods they are using.

Project Thoughts and Feedback

Ryan - The project was interesting and definitely had more depth to it than I first thought especially when it came to implementing the GUI. GUI definitely had a bit of a learning curve to it when it came to making it function and not being entirely clunky codewise. I believe we did a fairly good job on the project. Happy with my partner I believe we had good communication and was always happy with everything he did on the project. I would have liked to be more helpful when it came to functional code as implementing that was my biggest weakness when it came to the project.

John - The management of the project did not prioritize initial key aspects like the GUI. This meant that during the project little about the GUI was decided and when we went to implement it we found that we were unsure how to get it working with the current code. Also, the time management of the project was poor and led to a large amount of the project being completed near the end of the project due date. I would have also liked to have added more testing that fully tested the functionality of the methods and added more error prevention.

Retrospective

Overall our concept and our ability to implement features we both agreed we could do worked well. We never ran into an issue where we wanted to implement a feature into our project and it was something we decided we could not do. Functionally our program meets all the specified requirements (add more about projects efficacy after its done)

Our project would have greatly benefitted from better time management and better initial planning of goals. Certain aspects of our project needed to be reiterated and changed as over the course of development they were made redundant, such as the GameLogic class which initially had the interface. If we had a full grasp on scope from the beginning it would have saved time and made things work more effectively during development. While planning tools such as Trello were used in the beginning they were not consistently utilized which could have helped with managing set work.

Finally, while basic testing was implemented we were not able to fully realize the leave of testing and bug catching we had originally wanted to have due to the time limits we encountered. If we had more time we would look to fully implement more unique tests and add more bug prevention code to our game.

Effort and Contribution

Time spent on the project (in hours per week) varied greatly depending on other commitments both of us had at certain times. John on average worked greater hours than Ryan per week with a much harder focus on testing and constructing code as well as Java documentation, whereas Ryan dedicated overall less hours than John with more of a focus on design of the GUI and adding the testing. Both of us maintained steady communication of ideas and were almost always able to agree on key design choices.

Agreed contribution is between 65% John 35% Ryan. John definitely averaged more time on the project and was taking on tasks that could be considered generally harder.