SENG201 Project Report “Volleyball Mania”

Stephen Hockey

Student ID: 77313588

Lachlan Stewart

Student ID: 62378351

Date: 23/05/2023

Introduction

Volleyball Mania is a Java based game made using the Java Swing GUI toolkit, written and developed by Stephen Hockey and Lachlan Stewart for the SENG201 Project.

Application Design and Decisions

The program is run from the main method of the ‘GameManager’ class, which serves the purpose of overseeing the launching and closing of all GUI windows. These windows are the user’s only way of interacting with the program, as shown in the UML Use Case Diagram. The ‘GameEnvironment’ class stores the game’s important data such as the inventory and current week. These variables needed to have a wide scope and there would only ever be one instance of these variables across the program. We chose to define them statically, so that every class can get and set the same data without needing their own instance of ‘GameEnvironment’. The ‘Athlete’, ‘Item’, ‘Team’, and ‘Match’ classes make up the rest of the game logic, with ‘Athlete’ and ‘Item’ being subclasses of the ‘Purchasable’ class. ‘Purchasable’ was created because the ‘Athlete’ and ‘Item’ classes share the properties name, price, sellPrice, description and stats. An example of association between classes is the ‘Match’ class having the property opposingTeam, which is a ‘Team’ object, and the ‘Team’ class has the property players, which is a list of ‘Athlete’ objects. ‘FileHandler’ is a special class with a singular method that is used to read text files. ‘FileHandler’ gets used by ‘Athlete’ and ‘Team’ in their respective random generation methods.

Through the process of developing a command line based version of Volleyball Mania first, to then develop a GUI based version plenty of changes were made which resulted in a far more streamlined and cohesive system. In the command line version, as the player had more freedom to enter unwanted characters into the interface, exceptions were needed to handle these user errors. In the GUI based version these user errors could be handled by the front end resulting in far fewer exceptions needing to be thrown. In developing the GUI based version multiple classes became redundant as their functionality was replaced by the windows. Other classes were pulled apart and refactored into other locations, such as the generation of new ‘Athlete’ and ‘Item’ objects. These were static methods of the ‘Market’ class but after its removal, they were moved to their respective classes while remaining static methods. The classes written for the command line based version were well modularised, meaning that methods dealing with the users input were separated from methods dealing with game logic. This modularised approach allowed for an easier transition into the GUI based version as the game logic classes were able to be reused with minimal change and only the user input classes were rewritten.

To test various classes functionalities a test suite was developed to check their correctness. A measure of quality and comprehensiveness of these tests was the unit test coverage, which returned the percentage of total code run during the running of the test suite. The coverage of the test suite developed for Volleyball Mania had a coverage of 25.6%. To declare the application well tested a unit test coverage of 85% to 90% was preferred. A reason for the low unit test coverage of Volleyball Mania was the large volume of code which generated and ran the screen classes such as ‘HomeScreen’ and ‘StadiumScreen’. Junit was not equipped to test outcomes from events such as Swing button presses so these lines were not covered in the unit tests.

Thoughts and Feedback

Overall, the project was enjoyable and a good view into how a team of engineers might interact in the workplace. The concept of a game in which you create and manage a sports team was good for people new to coding games, as lots of logical knowledge came from knowing how the game works in real life. It was a huge project to take on, which made it very rewarding to get to the end and be proud of something we made. The sheer size of the project also made it challenging to make decisions on what classes were even needed, what relationships these classes had with each other, when to use an interface or abstract class or inheritance et cetera – it felt like you could make the program in a million different ways. To also do it all in a programming language that we had only started learning in the past couple months was an exciting endeavour.

To improve the experience a more comprehensive covering of the Java, Swing and Git system content would have been useful. The lectures of week 9 centred around architectural and design patterns would have been good knowledge coming into this project rather than in the final weeks.

In this project time management could have been handled better. Beginning work earlier and keeping working on the project consistently would have been less stressful overall. Next time setting up a good Git framework and learning how to merge branches effectively would have sped up the coding process. Good team communication was an essential part of finishing this project on time and to a good standard, and we were lucky that we were always on the same page about what needed to get done and we were both open minded to each other's ideas and solutions.

Contributions

Both members of the development team contributed roughly 80 hours to this project. With both partners contributing 50% each to the project. No one person did more work than the other. The way the workload naturally fell into place ended up being Stephen doing majority frontend, and Lachlan doing majority backend, which was a very beneficial arrangement.