Football Manager Report

Sam Willems	Joseph Hendry
swi126	jhe135
31643284	14495072

Application Structure

The structure of our project mostly follows the use of a state holding class, the “GameManager” class. This is started from the Main class, which is used to start the program and select the UI that the user wants to use. We are using the GameManager class as a facade design pattern. The GameManager class calls various other classes like StartupMenu, MainMenu etc to allow them to serve their functionality. These functions then call an end function of the GameManager class eg onMainMenuFinished, which returns to the GameManager class which handles what to do next. This makes the classes in our application have low coupling which makes the program flow easy to understand, and easier to maintain. Additionally, because our core classes like the menus, as well as our object classes like Item, Player etc have low coupling, it allows us to focus the functionality of those classes purely on methods that relate to the class. This means we have high cohesion within the classes, which makes the classes easy to understand and keeps logical and functional separation when necessary. These ideas are shown in our UML diagram which we used additional software to develop. Throughout the time spent developing the application, we changed the structure multiple times into this current form, as we found throughout expanding and fleshing out the application that this was the best way to keep our code clean and easy to maintain. Additionally, we implement an abstract class “Window” which is used across all of GUI classes and contains common functionality such as initialising the windows, the closing operations and access to the GameManager.

We handle the different UI options - GUI and CLI - by using an interface “GameManagerUI” which enforces functionality such as the different types of menu when using each UI option. This also makes passing the data from the UI to GameManager easier as the common type means GameManager will receive data in the same way regardless of the type of UI being used.

The package structure of Football Manager separates various similar classes into their own packages. We have the main package, which contains the Main class, and main.body which contains GameManager and our other core object types like Player, Team, Item etc. We also have main.UI, which contains the GameManagerUI, and two sub-packages main.UI.CLI and main.UI.GUI, which contain their respective UI classes. This structure is useful as it separates the different types of classes, as well as allowing similar classes to be logically grouped and imported where necessary. We also have a package that contains all of our JUnit testing.

Testing

For our unit testing of the project, most of our testing is focused on the elementary classes, such as Team, Player, Item etc. For these base classes we achieved 95-100% coverage, however for the classes that tie things together such as the UI classes, the state holding ‘GameManager’ class and the UI interface ‘GameManagerUI’, the testing was done manually and is recorded in the included document ‘SENG201 Project Testing’. The overall coverage via the JUnit testing was ~40%, however due to all of the manual testing, and a significant portion of the codebase being in the aforementioned manually tested classes, we would estimate that around 90% of the codebase is tested in some form.

Thoughts and Feedback

Looking back on the project, we have identified a few key learnings about what went well, and what could have gone better. Throughout the project, we communicated well to discuss what we were working on, our priorities of when we wanted to get done next, and our general thoughts on how to do things. This made the project very easy to collaborate with each other on and share our ideas. However, we felt that we did not manage our time very well, and ended up doing a significant amount more of the project in the last few weeks than at the start. A lot of this was due to spending too much time working on the command line interface, when we should have been working on the GUI. We also had difficulty juggling working on this project with work from other classes, as well as leaving enough free time to recharge.

Due to this being our first larger scale, collaborative project, there are a few things we would do differently next time. We realised that doing more planning before starting the coding of our project would have allowed us to have a more clear vision on what we should be doing at each step of our way. This impacted our current project as we changed the way that we handled passing data around multiple times as we expanded the application, which meant wasting time on something that might have been avoided with better planning.

Overall however, we both feel that we worked well together and the collaborative approach allowed us to each focus on what we were good at, and peer review each other's code. Good communication made the project go smoothly, even when we were stressed with time pressure.

Effort and Contribution

Hours:

- Sam: 100 hours
- Joseph: 130 hours

Contribution:
Sam: All testing, Report, Use Case Diagram, 25% of Code.
Joseph: UML Diagram, All Javadoc, 75% of Code.
Contribution percentage: 40% Sam, 60% Joseph.