SENG201 Project Report

Lachie Angus: 82545516 Luke Armstrong: 46823276

Our tower defense game is structured into several packages, ensuring that the code is modular to promote maintainability and testability. ‘seng201.team0.gui’ contains all the fxml controllers as well as the FXWrapper and MainWindow classes. These classes use JavaFX to handle all components related to the GUI of our application. These classes handle the screens and the transitions between them smoothly, as well as the core functionality of the GUI, allowing for optimal user interactions. A downside to this structure of the controllers is the project's testability, which is a topic covered below. ‘seng201.team0.models’ contains all the core game models such as towers, items, carts, round, and random events. Within the towers package, we have 2 abstract tower classes, Tower and SupportTower (which extends tower). These two implementations have 9 and 3 concrete subclasses respectively which collectively make up the tower representations the game is played with. There are also the abstract classes Item and Cart, which have 6 and 3 concrete subclasses respectively of distinct types. Tower and Item implement the interface Purchasable because they are both items that can be bought and sold through the game’s shop. This allows for a flexible and extensive design and promotes polymorphism, code reusability, and consistency. We elected to implement RandomEvent as an enum with the logic for the random events located within the GameEnvironment class.

The final relevant design feature is having AlertHandler as an interface within the ‘gui’ package. This allows for much improved JUnit testing within GameEnvironment because it enables the alerts to be handled separately. By adding a ‘Mockito’ dependency we could then fully test GameEnvironment which was important for our project's testability because that is where the greatest portion of the game’s overarching logic is. By adding ‘Random’ as a parameter in GameEnvironment we could set the seed of the random generator. This allowed us to have consistent results when testing features that implemented the random generator, such as random events, and circumvent the non-deterministic behavior of the random generator.

According to the final ‘JaCoCo’ test coverage report our project had unit test coverage of 31% on instructions and 36% on branches. According to IntelliJ's inbuilt test coverage report our project had unit test coverage of 76% on classes, 44% on methods, and 37% on lines. At first look, this may seem low but considering how GUI intensive our project is, this is a fair coverage score. For non-fxml classes, our unit test coverage is about 80% which is spot on considering trivial getters and setters which do not need to be thoroughly tested. The inability to unit test the controller classes is because of classes’ extensive use of JavaFX. This does not mean that this entire portion of the project is untested, however. User testing played a vital role in ensuring that the

controller classes work as they are intended. Because of this, we feel the calculated test coverage is misrepresentative of the project's actual portion which is comprehensively tested.

We feel that overall, the project went successfully but it did not come without its struggles and mistakes. We found it hard to interpret some of the requirements because our grasp of the specifications clashed with what we should have been implementing. This led to code and features that we had to backtrack on or totally rethink. This could have been avoided with more thorough planning and making use of our UML diagrams coupled with detailed reading of the project specifications. If we were to approach a project like this again, we will be sure to try to think out the shape we want the project to take before fully implementing features. We also had a fair share of bugs that we had to fix. One bug which was particularly tricky occurred after some random events. The tower labels in the game screen would become incorrect but the game's state stayed consistent. This did not seem to be logical at first but after some thorough debugging we eventually found the issue to lie within the methods for random events. It turned out to be a local list that was assigned to the same place in memory as the main towers list.

On the other hand, there are aspects which we feel went well. Scenebuilder integrated seamlessly with JavaFX to create our fxml files and design our scenes. On top of this we did not have trouble where we most expected to. This was with the logic behind setting up the application window and switching between scenes. This is because the previous labs and tutorials had been helpful in giving us an understanding of how this works as well as solid example code. We are happy with how the gameplay turned out and the way that we were able to apply it within the project requirements. Using animations with the progress bars and having to manually fill the carts by clicking the buttons is what we feel completed the gameplay aspect of the project. Lastly, using 'checkstyle' to iron out style errors proved incredibly beneficial to the consistency and readability of the code.

Lachie worked about 100 hours and Luke worked about 75 hours in total. This gives an estimated project contribution of about 60% and 40% for Lachie and Luke, respectively.