SENG201 SpaceExplorer Report

Jack van Heugten Breurkes - 23859472 Connor Macdonald - 41647584

Application Structure

The game uses a simplified **Model View Controller** style architecture. The view and controller are both represented in the form of a graphical user interface. The model is comprised of classes for objects in the game such as planets, crew members and items. The state of the player's crew, ship and inventory are handled by the CrewState class. The random events are handled by the GameEventManager class in the eventmanager package.

We identified and **separated the central systems** of the game during the design stage to reduce coupling. In the source code this appears as the packages main, crew, items, eventmanager, commandline, gui and test. Reducing coupling also made collaboration easier.

Inheritance is used heavily in the project, for classes such as CrewMember, GameEvent and Item. CrewMember and GameEvent each have a number of subclasses that offer variations on their core behaviours. Item subclasses form a multi-tiered hierarchy and includes abstract classes and interfaces to allow for different implementations of similar functionality.

The **Observer pattern** was used as a simple way of notifying the GameEnvironment of status effect changes to crew members without coupling the implementation of GameEnvironment to CrewMember. This allowed for easy creation of popups to notify users of changes.

Use of Collections

Collections were used extensively in keeping track of groups of objects in the game. We used the **List<T> interface** for keeping track of inventories, our crew, and the planets in the game. This was implemented on the object level using ArrayLists, but could be swapped out for any suitable implementation of the List interface.

We used an **EnumSet** to track the status effects active on each crew member. Each effect is unique and can only be included one or zero times per crew member, so a set was a suitable collection type.

During development we could not find a suitable collection for storing data with an integer weight associated that we could use for randomly generating loot. To solve this **we created the WeightedArrayList class** to handle this association.

Test Coverage

Our overall unit test coverage was **40.3%**. This is mostly comprised of the game's base classes such as features relating to the crew and ship, our item classes and our shop and planet classes.

To test classes such as GameEnvironment and the event system that have multiple lower level dependencies, **we developed a system testing framework** that links in with the command line. This allowed us to quickly run through test scenarios based on our UML State Chart file and verify that they play out as expected, dramatically increasing our test coverage on a system level and making it clear when a change caused issues elsewhere in the code. Details on usage can be found in our readme.

Thoughts and Feedback

Overall we thought the project was a valuable test of the skills taught over the course. The soft skills required to work with others on a software project became clear very quickly, and the practical aspects of designing and building a system with interlinking parts was a beneficial exercise in problem solving. The project strongly complemented the course and it was interesting seeing how the content of lectures related to the real world.

Retrospective

If we continued with or redid the project one thing we would do would be to **consolidate the GUI popup messages**, so instead of having multiple popups for events happening at the same time we could display them all in a single window. In general a more generic notification system would be a good next step.

Another thing we would do would be to **improve our command line test system**. Currently it requires manually checking that the game finished as expected; it would be good to be able to set up a system to verify the result of the test so that we could run them automatically when updating our repositories.

A lesson we learned was the **importance of unit testing**. We made sure to write unit tests at the same time as we wrote the base classes, and although it made the implementation take slightly longer it meant that we thought about the purpose of our classes more and noticed a number of potential issues before they caused any errors.

Overall we were happy with how we went about designing the game. Our underlying system is split into sensible packages and the functional side of the project linked up well with the GUI. We were especially pleased with our approach to testing, creating unit tests at the same time as we built the base classes and using the command line to do system testing as the project came together.

Effort (50/50)

Each partner committed an estimated 70 hours of work. After discussing our views of each other's effort we agreed that it was roughly a 50/50 split. We were both involved in the development of the core system and the unit tests. Some of the higher level work was distributed asymmetrically since some systems were harder to collaborate on. For example, Jack worked on the GUI while Connor worked on the event system.