SENG201 Project

By

Finn McCartney (fwm16)

Michael Freeman (mjf145)

Structure:

Functionally, the game is all structured around the ‘GameEnvironment’ class where the source code for all players actions within the game resides. Smaller actions such as the initialisation of the game are confined to their respective classes.
We designed it in this way because most of the game windows classes have potentially multiple instances of implementation as they are just simple objects for input and output operations, so storing game data and methods had to be done outside of them. All classes besides the ‘GameEnvironment’ and the super classes are hence treated as supporting classes in order to ensure information is not being tossed around like a hot potato, risking its loss.

The game has five super classes (‘ConsumableItems’, ‘CrewMembers’, ‘Planet’, ‘Shop’, and ‘SpaceShip’) that most of the game’s objects use.
Our objective with creating the super classes was so that they could match any design changes we may wish to make further along the lines as we originally had multiple ideas of ways in which to incorporate many of the objects, such as the items and shop. As a result, they are broad enough to make the individual objects flexible in their implementation and use, but constrained enough to meet the requirements in every instance of their potential implementation.

In terms of collections, we only used array lists, as they are easy to pop to and from, easy to search, and easy to implement with the GUIs we used for the games interface. These were used in cases like drop-down menu’s, current crew lists, and inventories of both the ship and the shop.

Unit Tests:

We wrote test cases for all the super classes first of all, because if the super classes are in working order then we can expect the objects within the game under these super classes should operate without issue. Further unit tests were used but were not deemed necessary everywhere because many of the variables in the game (such as items) were hard coded into arrays (such as the shop inventory) and as such were subject to little risk of error.

Review:

Beginning the project, we had a lot of ideas for how we could make the game and what would result in the most fun to both build and play. We drew up a design plan (super classes, flow charts for gameplay, points of interest in order to stay within the requirements) and got straight into building the program.
Once started we didn’t really stop and had a working version of the game within a week, however we underestimated the wait of the work that would be attributed to writing the unit tests and documentation. This mistake lead to more work than we would have had to do had we done the tests and documentation as we were building it.
Regardless of this slip, the project came out as we had both hoped it would functionality

wise. What we did miss out on doing was creating all the graphics we had originally wanted for the game in order to give it a unique personality, but with the end of semester coming down on us we ran out of time to spare on the aesthetics of the game.

Both having an interest in developing the game for entertainment purposes meant that there was no lack of motivation to get the project up and running. I (Finn) predominantly worked from the outside-in (design, structure of gameplay, and GUI designs and structure in order to ensure flow of gameplay), while Michael worked from the inside-out (classes and methods, structure of code to ensure efficiency and flow, and data storage and communication). Working in this way allowed us to each become experts in portions of the construction process, rather than both semi-knowing how to do a majority of it. This is not to say that we did not both have a hand in every step of the process, but rather that throughout each step we were alternating leadership and input based off who had started with that part of the project.
As a result we have concluded and agreed that we each contributed an even 50% to the project, and have estimated that we each spent approximately 15hrs on it.

Going into future projects, although we were happy with the finished product, we would change our approach to the entire process by better planning our time in terms of supporting context for the project (tests, documentation, readability, etc.). This would predominantly mean actually working on all these things as we write each portion of the source code.