

Seng 201 Report

Application Structure

Our application uses several Java structures to implement the game. Such as abstraction, enumerated types, overloading and more. Abstraction of the StatLevel class allows the implementation of pet stats to be more modular and have most of the behaviour encapsulated to methods of a higher class, and be overridden where necessary. This Inheritance means the objects can be accessed similarly. The Enumerated Type Specie stores the information of all the types of pets available in the game, By Extracting all this information to this class specific information for pets can be accessed through the Specie Class. This was useful for keeping all the base stats and image locations away from each individual pet. This allows the Pet class to focus on the pet as it is at that moment and then use calls to Specie when it needs other resources that are specific to Specie. This also allows the species to all be defined in one place so they can be compared and balanced easily.

The Game class is the workhorse of the application and the object created is passed through other classes (primarily GUI classes). This gives access to important properties and behaviour that is needed in many methods. By storing Array Lists of players, toys and food within the game object it allows them to be accessed anywhere the game object is passed in. There is only ever one game object used per play of the program.

Array Lists were chosen for the collections as they have many useful built in methods and allow Arrays of objects we created, also we were more familiar with them at the beginning of the project. Methods like; .add(), .get(), .clear(), and .contains() were useful for manipulating the Array Lists throughout the game to add and remove objects like players and pets. The Array List also allowed simple selection of an object in a JList that had been populated from such an Array List.

As Java is an Object-Oriented-Programming language objects were heavily used throughout the project. By making an object with specific starting properties and never changing it, instead only cloning it. This allowed independent objects with the same properties such that they could change by themselves and be given to different players without affecting each other.

Unit Test Coverage

JUnit Testing was used to test individual classes method by method, whilst basic methods that simply return or set values were not normally explicitly tested on their own however many of them were used to implement other tests thus covering them. Detailed methods that included behaviour we implemented were tested to make sure they functioned properly.

Whilst some private methods were not tested directly, however we tested them indirectly by seeing their side effects. An issue that was noted was having to be smart in how we measured some tests. For example, calling a public method to change something so we could then test another method on it with parameters that were feasible in the game.

The GUI classes were not tested using JUnit testing as JUnit does not have the functionality to test graphics. Instead to test this we ran the program and played through it with different settings to confirm that it worked as intended. Because of this the overall coverage was not incredibly high however for non-GUI classes which hold most of the important behaviour the coverage was much higher.

Thoughts and Feedback

The assignment was rather interesting to complete however a lack of GUI and thread experience led to some difficulties. However, throughout the assignment these were overcome allowing us to learn some more intricate functions of these paradigms. And get a functional knowledge through practice.

The Freedom given by the assignment was welcomed as it allowed us to put our own spin on the program and made it much more fun to code compared to having to code something very bland and uninspiring. This allowed us to experiment on things our own and problem solve when we ran into a problem instead of being spoon-fed what we are expected to produce. Thus, allowing us to learn by engaging and doing.

Retrospective

The Graphical User Interface was not how we envisaged it to be. During the making of the GUI difficulty was had getting the GUI to change panels allowing for one window to handle everything. But the eclipse window builder add-on made the creation of GUI's much simpler and not such a daunting task to complete, the auto-generated code allowed us to get a feeling of what was going on under the hood of the GUI. This gave a level of understanding to many GUI components, such as being able to see their attributes and behaviour and understand what certain calls and parameters do to different swing objects.

Getting the program to wait for a GUI window to finish was a difficulty until learning of threads and the ability to wait, using an Object "waiter" allowed the use of .wait and .notify, to provide thread based waiting while the GUI window was running so the program could continue without sitting in CPU intensive loops or holding up other actions from happening.

Improvements for next project

This project has shown the importance of time management in software engineering. Being the first big project either of us have undertaken, we did not realise how fast small bugs and misunderstandings can hold up the completion of the project. For future projects, better planning and allowing more time for unforeseen coding issues is paramount.

Now at the end of the project it is noticeable that at the beginning some structures and ideas were implemented as that was all we were familiar with. However, looking back it seems that spending time thinking about what must be implemented in more detail and looking at multiple different ways of implementing something, then comparing them to see which is the best for the situation. One example is using Array Lists, whilst for this assignment it feels like this was a good choice. Not much research was done into other similar structures that may have allowed for different implementation.


Key Contributions

Morgan: 85%



My Key contributions were making the GUI, implementing the game behaviour, making art for the game, bug fixing (GUI testing) and balance testing, writing Junit tests and keeping track of UML Diagrams, Javadoc and the Report.

Adrian: 15%



In the Production of the program My tasks involved writing some Junit test classes, some behaviour for the command line application and basic testing of the GUI.