SENG 201 Report

Matthew Smit – mjs351

Jacob McKenzie -

We iterated over several versions of Food/Toy classes. The first attempt was an enum, but since the toy needed to remember durability this caused issues. Then we had Food/Toy/ToyType classes, which worked better until we needed to have a GUI store. Since the store had one list for all items, we ended up with Food/Toy/Item. Due to this, we think it would have been better to design the UI earlier on, rather then right before implementing it.

We designed our classes so there was no circular references. We had a game class which was the main interface that the outside world used to interact, which in turn had players and events, and the players in turn owned pets. This allowed us to have pets which only interacted with themselves, players which interacted with themselves and their pets, and the game which interacted with everything.

The only reason the UI interacted with non game classes is for status related events, such as displaying the pet name, and so on.

We do have inheritance for the pets, although since pets are so similar to each other, we could have gotten rid of them and just used Pet/PetType exclusively. We used an abstract Event class, since the 3 types of events had the same interface. The other way different types of things was handle was through Enums/Enum style classes, since the functionality didn’t change, only the data.

Except when creating pets, arrays were used for player lists and pet lists. The reason is that those two lists aren’t changed, so having an ArrayList would give us features that weren’t desired. We do use an ArrayList when creating objects and the size isn’t known immediately, such as creating pets through the UI.

We designed the UI into separate windows, implemented as JFrames. This allowed us to separate unrelated parts from each other. We also separated duplicate sets of components into custom JPanel components, to reduce on duplicated components.

Unit test coverage is how much of your code is run via unit tests. Our test coverage was 93.3%, the reasons for not covering everything is to reduce useless unit tests. An example would be Game.play. All this method does is call getCurrentPlayer and Pet.play, both of which are tested. Therefore testing this function would be pointless. We also don’t test all the getXYZ methods, due to the simplicity of the method. The remaining missing test coverage is due to not all asserts being tested.

GUI implementation can be very tedious, getting everything laid out in the correct places wasn’t challenging, it was just boring. Using eclipse was irritating, changing over to intelliJ made it easier since it is a better designed IDE.

It did showcase a fairly large project, so the project in the context of the course seemed to succeed.

Attempting to use GroupLayout cost a lot of time, since WindowBuilder didn’t generate it very well, and doing manually was time consuming. We eventually switched over to AbsoluteLayout with a non-resizeable border, and that was much faster.

Our Game/Player/Pet classes didn’t change from the original design much, mostly it was adding more functionality. We did lack a lot of the later features we needed in our initial UML design, generally from overlooking elements of the spec or not taking into account UI functionality.

We contributed around 50% each.

Matthew Smit:

I worked on the events, a lot of check-style fixes, the game class, a lot of the UI functionality.

Jacob McKenzie:

Designed the UI, lots of javadoc, ...