**Seng 201 Project: Sky Buckaneer**

Names and student IDs:

Oliver Garrett - oga20 Matthew Garrett – mga114

**Project Architecture:**

We decided to use one base class in our project, “GameObject”, to host an x position, y position, and a JFrame Image object for rendering. Setters and getters were given to the class and a default rendering method, GameObject.draw(), was implemented. In one case, (In the Path class,) GameObject’s draw method was overridden to allow for a better custom draw mechanism, where the Path object rendered a long, continuous set of points as opposed to individual nodes. The GameLogic class serves as a state machine to keep track of all the current class instances, mainly Ship instances and Path instances. The GameData class served mainly as a namespace for hosting the Game’s images, and had no setters, only getters. Alike class GameLogic, the class EventHandler was another state machine that hosted functions to enact the event scenarios, e.g., pirate events, abandoned ship events, and weather events.

The Ship class, the Path class, and the PriceSprite class all inherited from the base class, GameObject. The PriceSprite class was used for displaying prices of goods, The Path class was used to represent the paths between the islands, and the Ship class represented the player’s Ship and held the inventory. **(See class diagram, Figure 1.)**

**Test Coverage:**

Our JUnit test coverage percentage was not the best. However, this was partially because many trivial methods were not tested. We did not test any of our one-liner getters or one-liner setters, because we were comfortable enough knowing that they worked just by looking at them, and we believed that adding tests for those methods would enlarge the code base unnecessarily and add technical debt. The main reason test cases are done is to check if a future change in the codebase has broken a method that was defined previously. For getters/setters, this would never realistically happen due to compile time errors.

The areas that we did test with JUnit consisted of small portions of the Ship class, and the GameData class. One reason much of our code remained untested was due to the inability to examine JFrame’s internal state easily. (For example, testing UI is borderline impossible.) However, the main reason a lot of our code was untested was due to the complexities surrounding the internal state of the GameLogic class. Due to much of the program’s functionality relying on the exact state of the GameLogic class, we thought that it would be more efficient to test the aspects of the program manually.

**What went well:**

Agreed contribution:

50 – 50 contribution.

We think we utilized inheritance quite well to distribute base GameObject behaviour to the Path class, the Ship class, and the PriceSprite class. Keyboard and mouse input was done relatively seamlessly through inheriting from the JPanel base class, which was nice to work with. The images kept in GameData was a good design decision, as it ensured that Image objects would never instantiate the same image file twice. We also think our use of setters, getters, and private class members worked well in making interactions between classes transparent and clean. Another great design decision was creating the State enum to represent the current game state. Assigning GameLogic a singular state at all times made a lot of code way more clear and less error-prone.

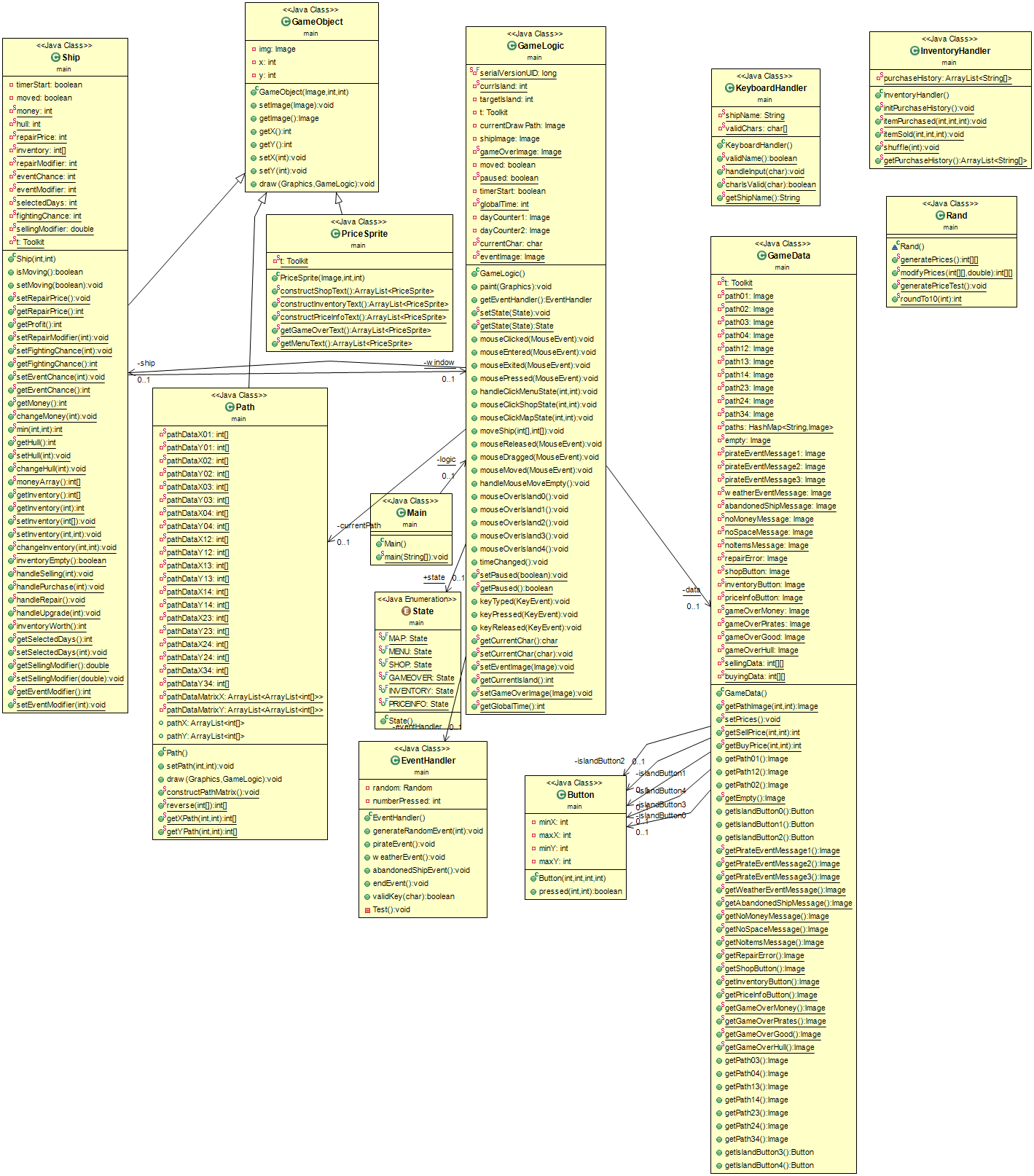
**What did not go well / improvements:**

As said before, unit testing was not done very well for us. Another thing that we should have done in hindsight is created an Island class, as opposed to assigning them to integer IDs. We think this would have simplified the program by a lot in the long run. Also, the GameLogic.moveShip() method should have belonged in the Ship class as opposed to the GameLogic class, however by the time we thought of this, too much of the timing state belonged in GameLogic. Moving it over would have been very expensive timewise.

In the Path class, there are giant integer arrays that were generated in python to interpolate the ship’s pathing across the map. A much cleaner implementation would have involved generating the paths in Java.

**Time spent on project:**

We estimate that the time spent on the project was roughly 40-50 hours each, including learning aspects of JFrame and artwork.

**Figure 1: The generated class diagram for our project**