CSCI-5548 Su22
Project: ZeldaLike Role Player Game
Team Members: Curtis Covington . Tim Coleman

2) Final State of the System

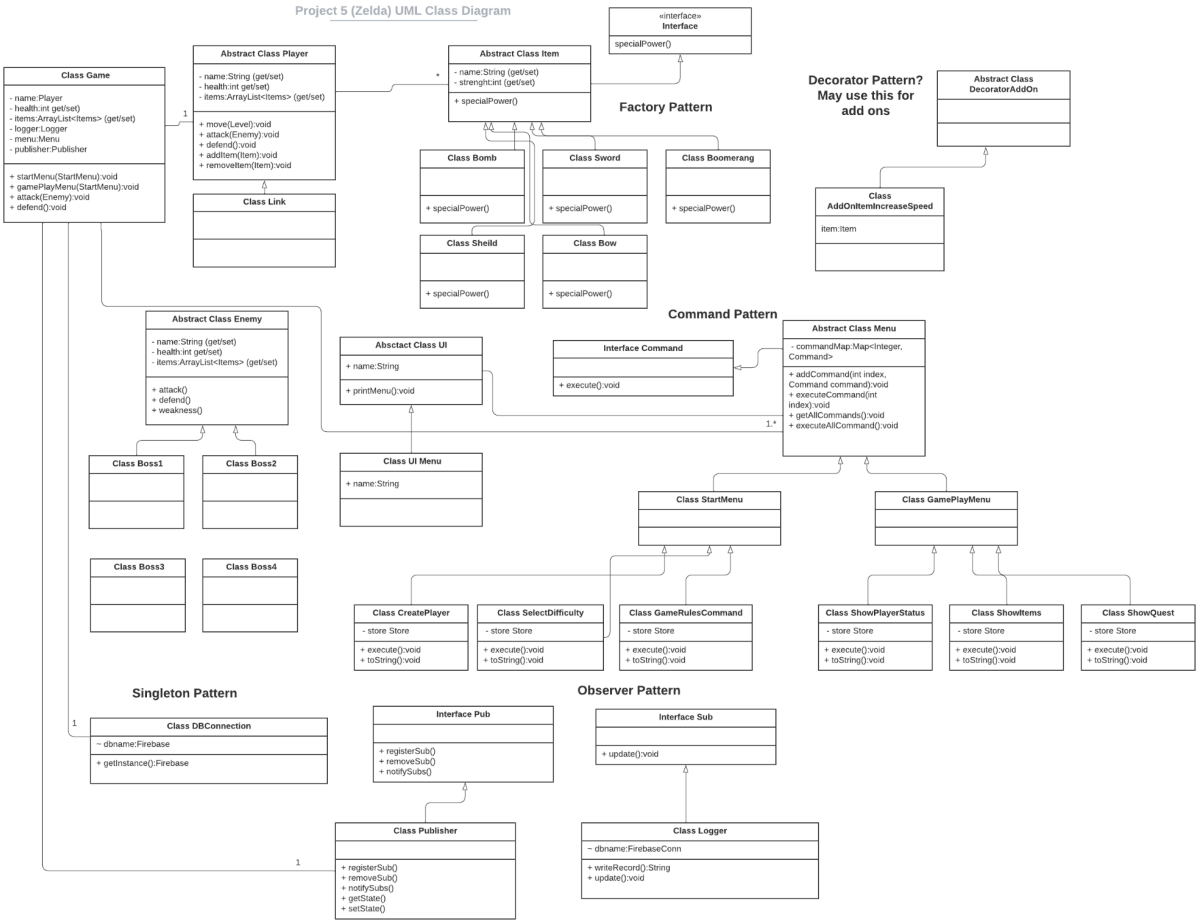
We set out to design a simple zeldalike role player game using the Flutter/Dart framework to build a mobile app to run on on Android device. The basic gameplay was accomplished in our original design in that we had the core elements such as events, shopping and boss battles, and the game player was able to navigate via a simple menu representing the levels of the game. The original UI intent was a little different from what we had developed in that it had more nodes that a user could click on to visit versus sequentially accessing the levels that we have currently. The boss battles were randomly generated and had outcomes based on the pure randomness versus the original intent to make the player solve a series of challenges to win money, but that was more of a stretch goal though and we are happy with how the boss battles work. The events, shopping and battles could be further built upon to improve the game play but this is a great and workin mobile app that we feel has good structure and design usage to be further expanded upon. The design patterns we used were observer, singleton, factory and MVC as fFutter is very widget based and we used the Flutter UI elements and separated the business logic from those to make it MVC like.
Overall, we liked how the game turned out and the introduction (Tim's intro) to Flutter/Dart was a massive bonus.

3) UML class diagram

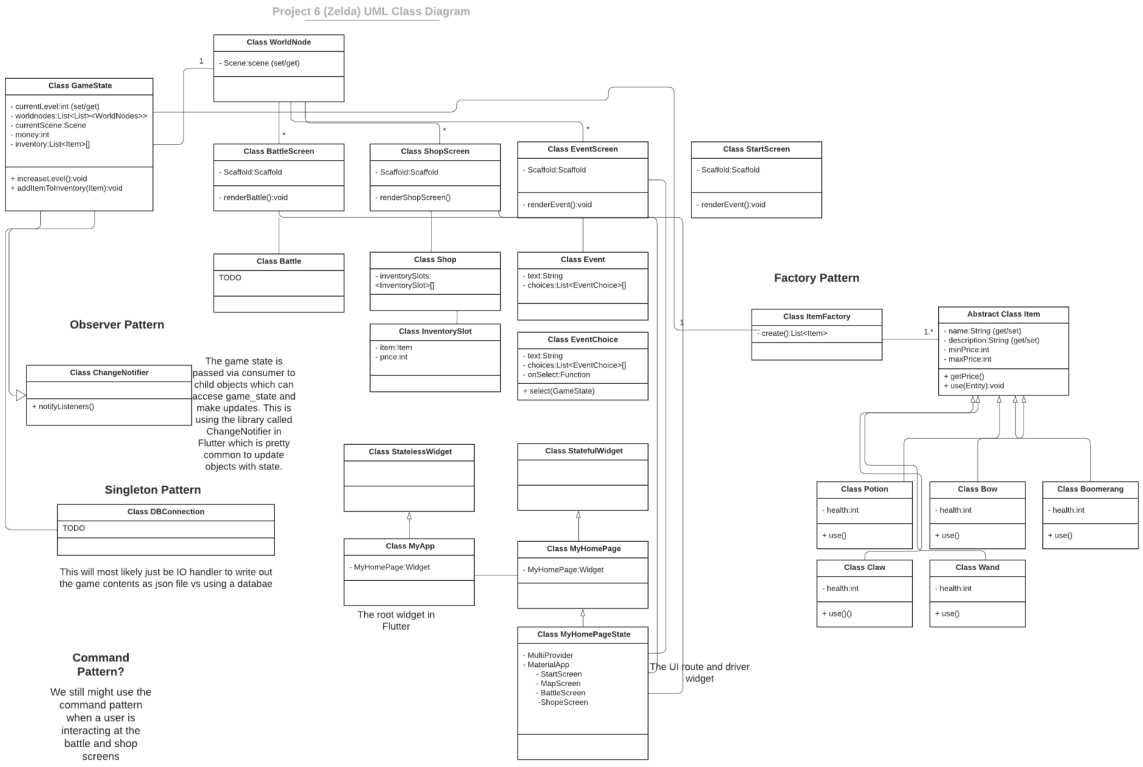
The UML diagrams changed from project 5 to project 6. Below are the three UML class diagrams representing Project 5, 6 and 7. Project 5 was really about using design patterns first and then building out the game based around that. Project 6, when we actually started coding, things started making more sense about what design patterns made sense to use as Flutter does things a little differently vs something like Python, Java. Flutter is very MVC in that it has UI widgets that are drawn to screens, and these UI widgets make use of plain old classes but there is separation between them, which is very MVC like. We also adopted the use of singleton so that only one game state object could be initiated, and a simple factory to generate the items, monsters and messages. We also used the observer pattern using Flutter's built-in state manager called Provider and NotifyListeners. The pattern we listed in Project 5 but not in Project 6 design was command pattern. We may have gotten this to work if we built a more advanced menu. For the final, we added more business logic classes for monsters, player and the boss

battle screens. We also expanded the use of factories by creating factory classes for monsters and messages. Below are the diagrams for project 5-7.

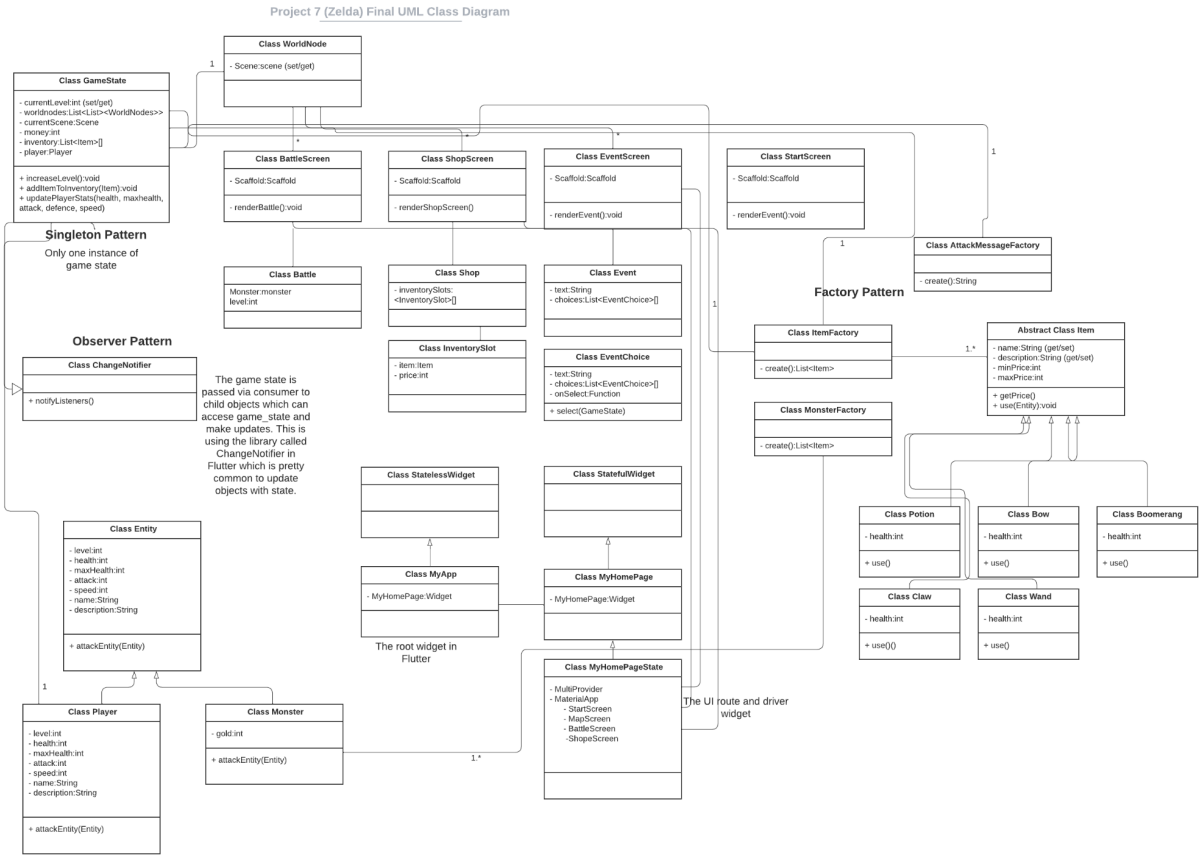
Project 5 UML Class Diagram



Project 6 UML Class Diagram



Project 7 Final UML Class Diagram



4) Third-Party code vs. Original code Statement

The game was written using the Dart/Flutter framework. We used the core dart libraries https://dart.dev/guides/libraries (dart:core) and the Flutter library. We also used Android Studio (https://developer.android.com/studio) for the IDE to build and use Android images to run virtualized Android devices for testing the app. We also used Flutter Getting started code to provide some basic boilerplate code found in examples on https://github.com/flutter/flutter. The rest of the code was original classes and/or made use of the dart libraries such as using the observer pattern and using the Flutter Provider/ChangeNotifier classes.

5) Statement on the OOAD process for your overall Semester Project

There were many great design patterns that were covered in this course. I am thankful to have learned so many of these. Some of the ones that felt very natural and easy to use right away (at least to me) were Factory, Singleton, Observer and Command. We used these in the course Pet Simulator and adopted Factory, Singleton, Observer and MVC in our game that we developed. The other patterns that really intrigued me were chain of command and builder. In my previous job where I did lots of data parsing, the chain of command would have been pretty great to use. If I encounter another project that has that type of workflow this pattern will be high on the list to

try out. I also would like to use Builder. Sometimes objects can be really hard and complicated to build and having a massive constructor with tons of various arguments just gets messy. Using builder seems like an intuitive way to construct an object and have cleaner code..