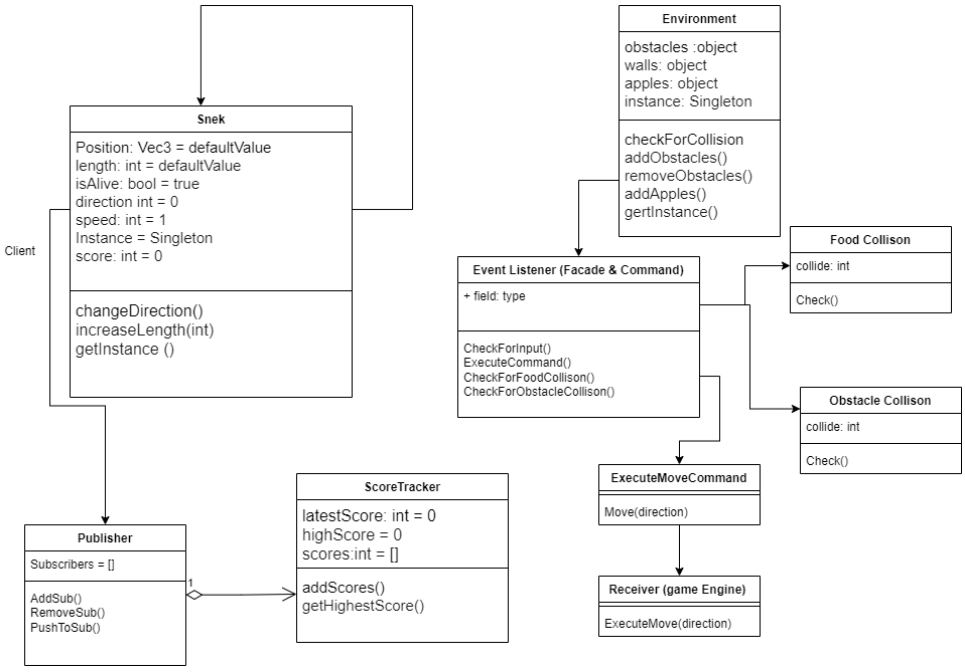
Amir Safavi
Colton Klable
Daniel Migdale

Project 7 - No Step on Snek

1. **Final State of System Statement**

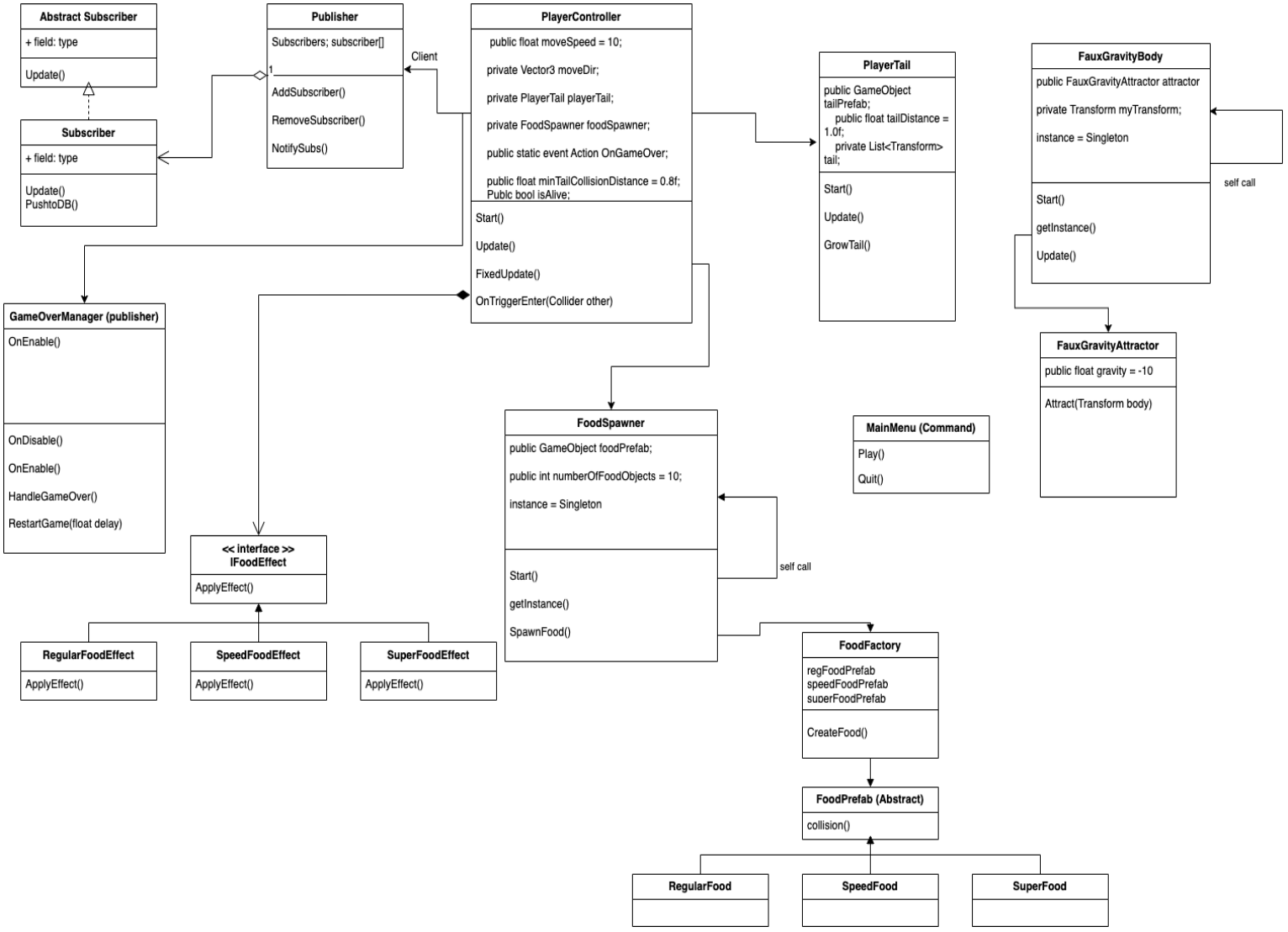
We have implemented the classic snake game on a three dimensional surface. It has a menu and game over screen with a navigable UI. Food spawns randomly across the map and there are two types of power ups. There is a bug we found and were unable to fix in time, where if you get two speed food objects in a row you automatically die.

2. **Class Diagram and Comparison Statement**



Above is our original class diagram. We created this before knowing what The unity engine provided and what needed to be implemented. Firstly our Snek class ended up being multiple classes. This helped better adhere to the single responsibility principle. Furthermore, We implemented our foodSpawner and FauxGravityBody as singletons. This helped us prevent errors as all of our other game objects would refer to the same instances of those classes. We did create an observer pattern as intended. We used observe to notify our subscriber of the game being over and push the game score.If the new score is higher than score stored in the database then the new score is stored in the database. We implemented strategy pattern to determine the score added and player speed based on the type of food the player ate. Factory pattern was used to instantiate the different types of food in the game. The main menu implements the command pattern by initializing the next scene object or by calling on the application object's quit function.

Final UML Diagram:



3. **Third-Party Code Statement**

We wrote and conceptualized the game logic ourselves, however the Unity editor creates a lot of object files which were used in the project. Additionally we took inspiration for some of the logic and UI from the Youtube videos listed below.

FauxGravityBody.cs and FauxGravityAttractor.cs scripts from Sebastian Lague's YoutubeChannel: https://www.youtube.com/watch?v=gHeQ8Hr92P4
Menu tutorial: https://youtu.be/zc8ac_qUXQY

4. **Statement on the OOAD process**
   a. Spherical game map - Early in the design process we decided to make the game on a spherical map. We felt that the classic flat map was too simple and

warranted an update for the new technology we were using. This meant adding new game objects such as a spherical gravity body.

b. Loose coupling - We were able to successfully implement the game by working on separate parts that relied on each other to create the full application. For example the main menu is its own game scene object which was built separately from the rest of the game UI, however the main menu object calls either the next scene object or the application object's quit function.
c. Good communication - We achieved success in our project by communicating early and often via discord. This allowed us to adapt to new challenges and ideas during the development process.