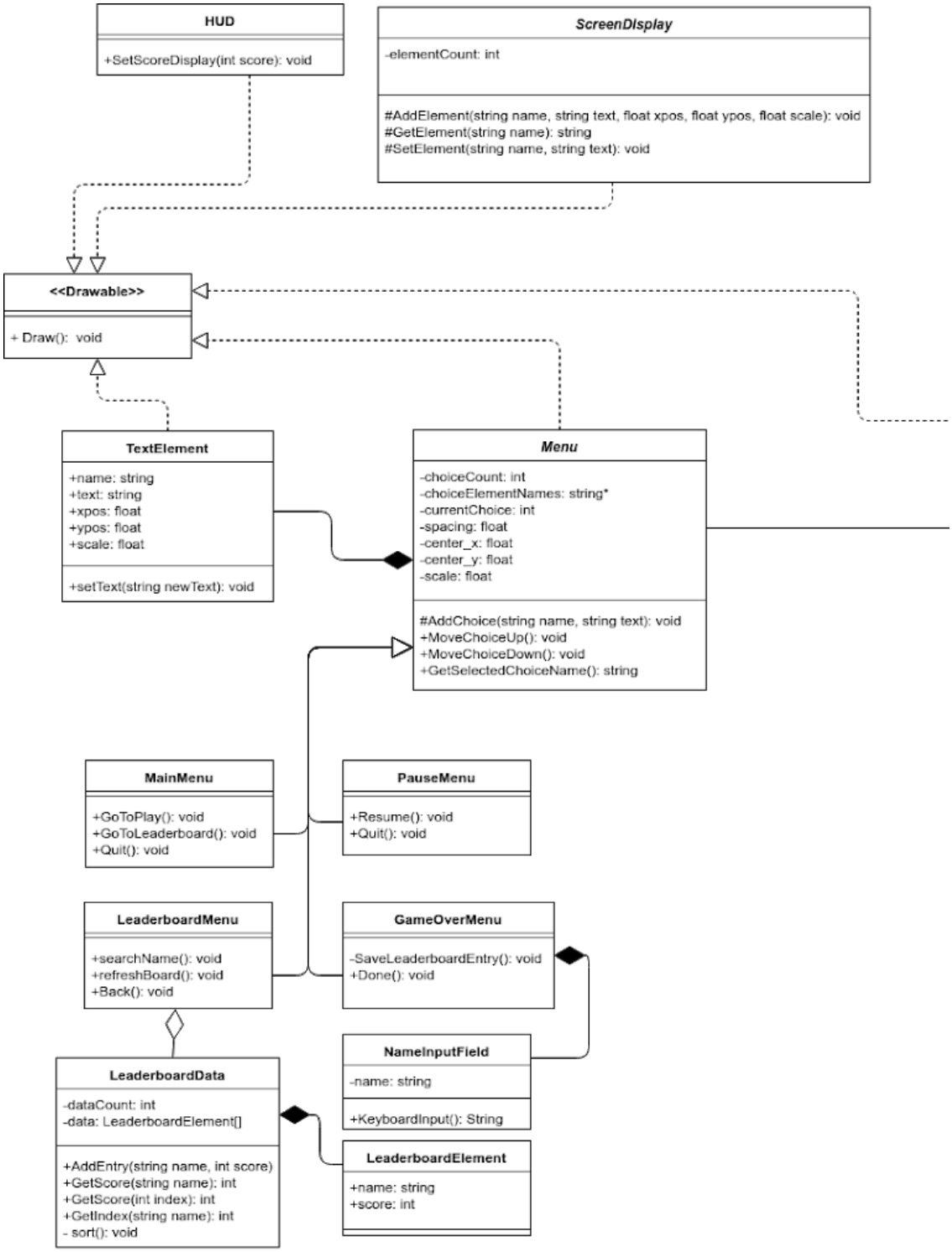
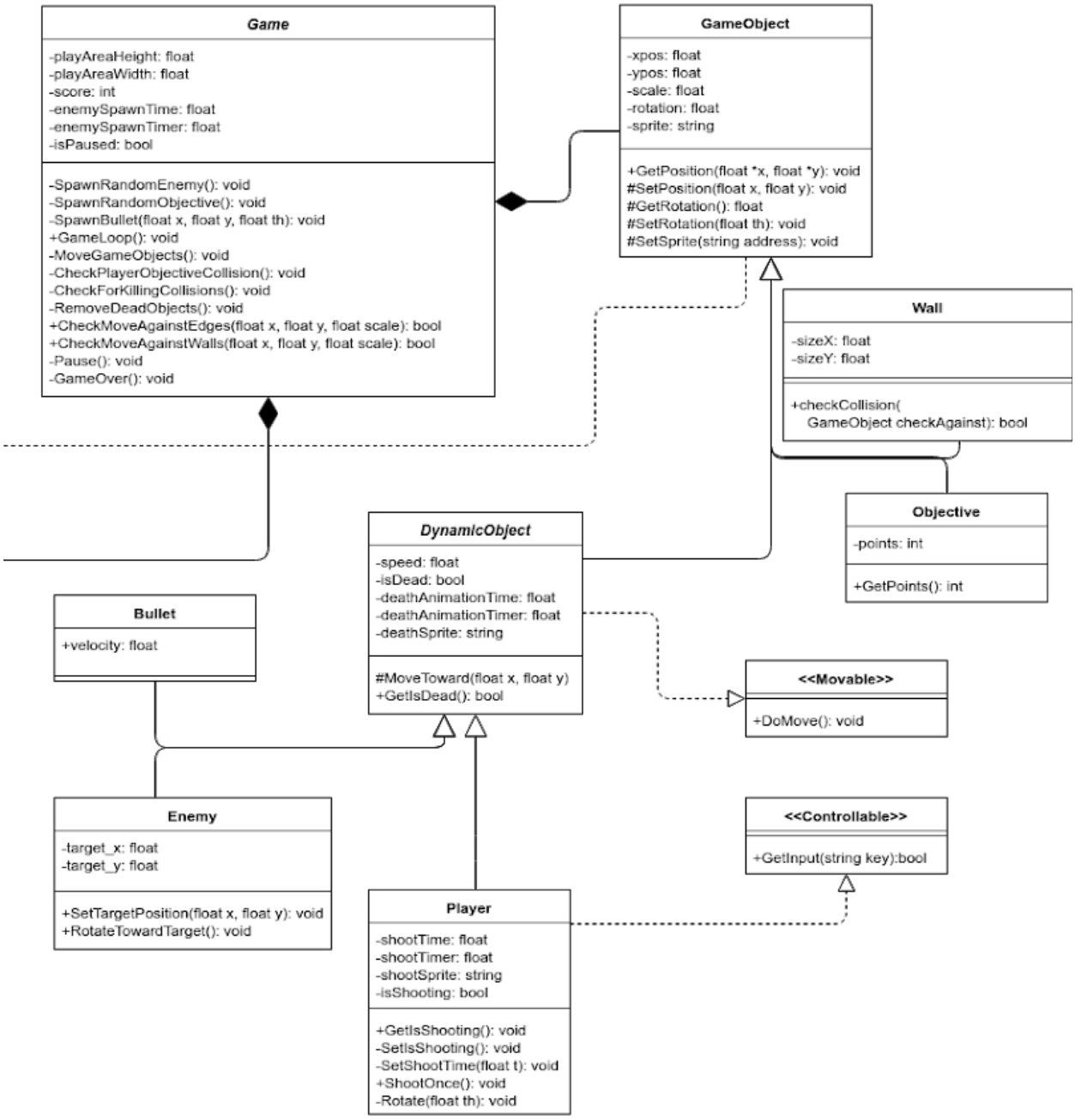
Team 15 - Java Game

David Nyberg, Charlie Davies, Ryan Davis, William (Ryan) Cooper

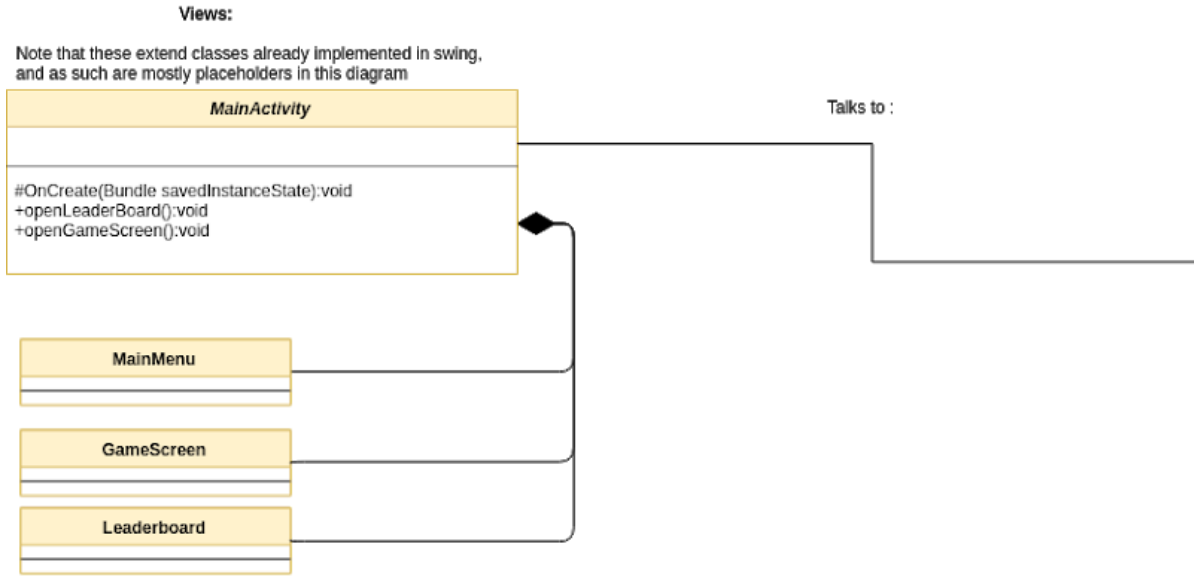
Initial Class Diagram:





***single image diagram can be found at**
https://drive.google.com/file/d/1CLHY_TZK2KPSFxiV0nlrfOoSAh3tEiW-/view?usp=sharing

Progress Class Diagram:



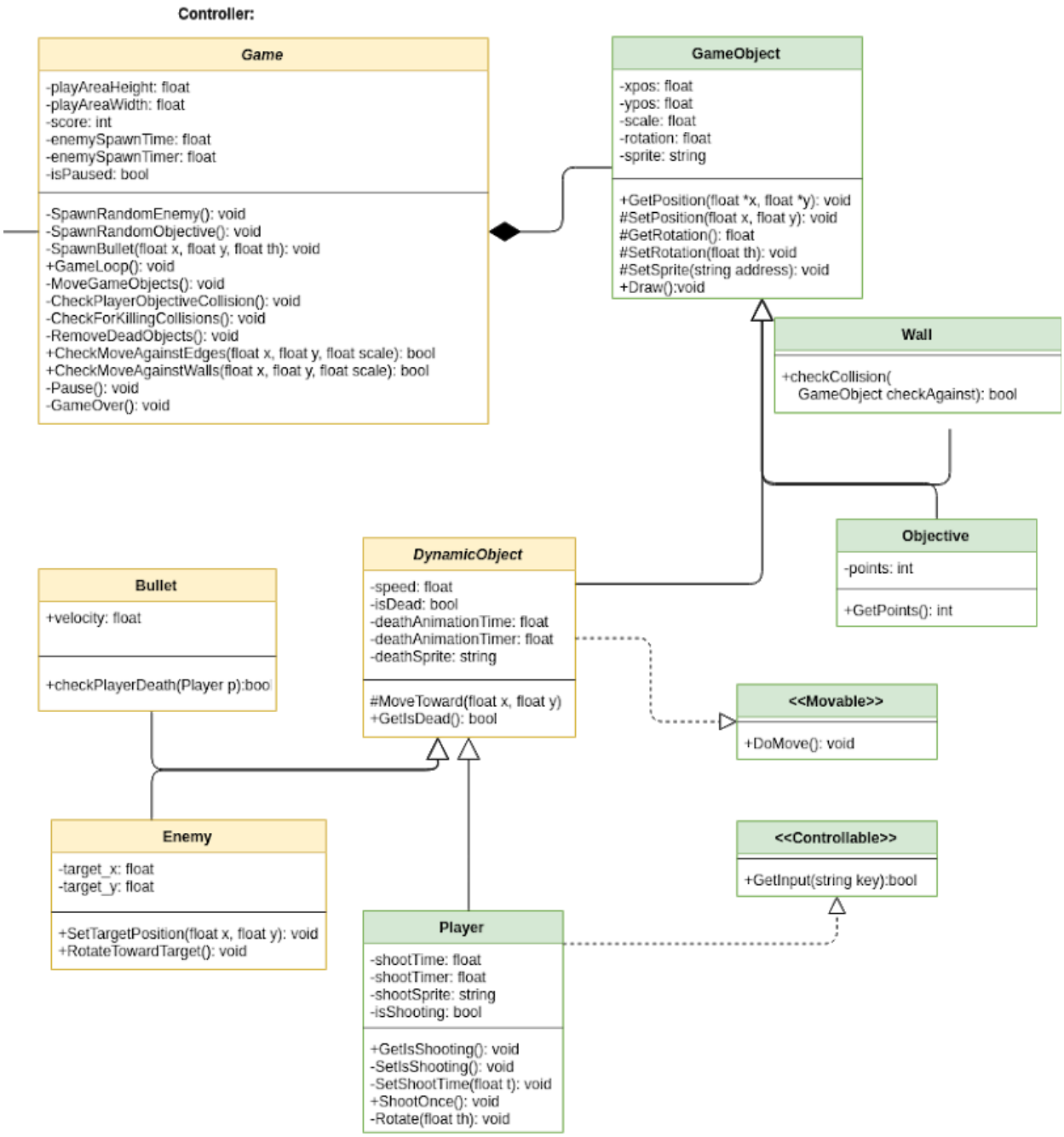
Goal In Progress
- framework in place
- function implementation not finished
- still needs graphical integration

Goal Complete for checkpoint
- important functions done
- still needs graphical integration

One Big Change is we are no longer using
android studio, but implementing our own MVC system.

The model is not shown here, and is the database.
When the game is over, the controller will access
the database to add a new high score

Note that not much has been acomplished on the side of the class diagram, we have spent the majority of the time since the last checkpoint looking into converting from android studio to an MVC design and swing.



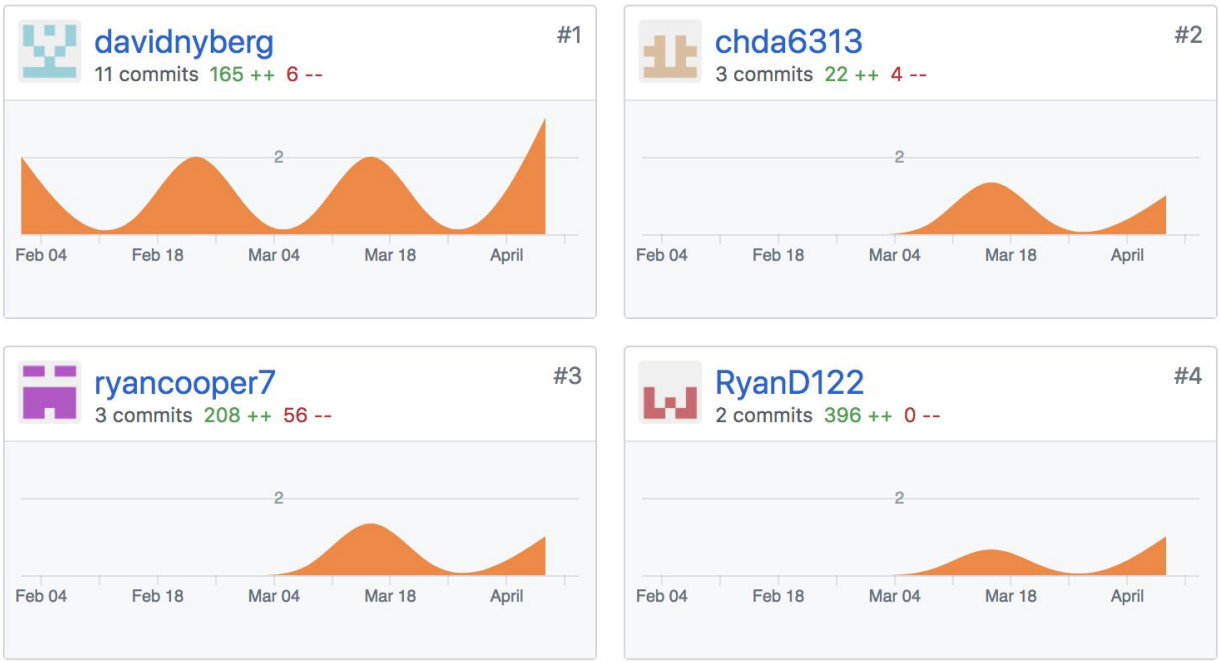
***single image diagram can be found at**
https://drive.google.com/file/d/1IS13SOg3Ddz4RLYNM3vEfRaGe75FI5_b/view?usp=sharing

Summary:
After doing more research with Android Studio, we decided to move in a different direction and make our own MVC framework. This allows us to better adhere to our original class diagram instead of having to adapt all of our work to Android Studio. After deciding to make the change, we set up an example MVC structure to adapt our classes to. We also created a very basic menu GUI using Java Swing. A MySQL database will be used with jdbc to create a high score in the model component of our app.

Breakdown:

William: Worked on learning MVC structure and Swing. Made simple Menu GUI.
Ryan: Started a basic MVC program in which a button renders a customizable rectangle.
David: Created MySQL database for leaderboard, worked on 'model' part of MVC using jdbc which connects to the database and will be used to store and retrieve data for leaderboard.
Charlie: Integrated MVC components into class diagram, continued to develop UI for implementation later once we are more comfortable with the tools we are using.

GitHub Graph:

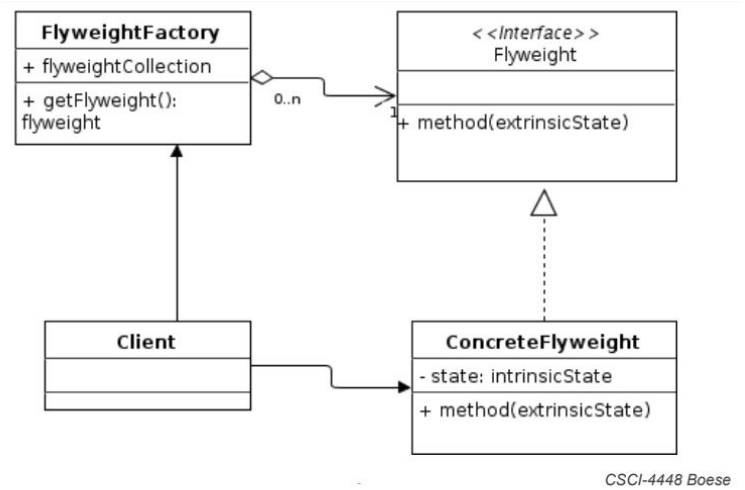


Remaining Effort:
-Test the model implementation to both import data into the database and send data to the high scores where it will be viewed.
-Develop graphics to be drawn with swing tools already in place.
-Link swing elements up properly

Design Pattern: We are implementing a MVC for our Java game. This works very well as each part can be easily split into a category. Currently we are planning to have the view as the menus, and leaderboards. The controller will be the main game and user interactions with it. The model will store scores in a database and will be used for the high scores. The design patterns used in MVC are Observer, Strategy, and Composite. The design pattern that will be most used is unfortunately not shown in our class diagram, as it is encapsulated in javax.swing. Composite is used in how the view is set up. This includes all buttons, text boxes, panels etc. which come implemented in the swing package. The whole class diagram is essentially part of the MVC as we can work separately on each part of the MVC but will be used together in the final product.

Apart from the MVC, one design pattern we have in our class diagram is the flyweight. Our Game class can act as a factory, and spawn entities. All of these entities share a lot of information, as can be seen through the inheritance tree. For example, the Enemy class shares a ton of common methods and data with everything else involved in the game, but they have a couple of tiny attributes that are unique to each instance of the Enemy class. This is also true for Bullet, Objective, and Wall.

Design Pattern in class Diagram: As our design pattern is very large, we ask you look at the class diagram shown previously. The template for flyweight is:



Following this template: our client is the MainActivity class, the factory is the Game class, and the flyweight is the Enemy class. The parent class (GameObject) that all flyweight classes inherit from is acting as the interface.

Final Iteration Goals:
-Manipulate basic mvc program into a game and fit in previous classes from diagram
-Have a working program that a user can interact with.