**Milestone 3 Design Document**

Wyatt Stagg, Nathan Kimball, Ben Hallihan

**Overall Design**

The application we’ve created is split into two distinct parts: Frontend and Backend. The Frontend exists as a ReactJS app, which will be run in a web browser and will mainly fulfill the role of the “view”. The Backend is the main logic of the application, which fulfills the role of “Controller” & “Model”.

**Backend Design**

**Backend Purpose**

The backend serves to host all the data for the game and act as a source of “truth” for the frontend views that are connected to it. This means that anything that would change the state of the game will have to be sent to the backend, which will then choose what to do with the incoming request.

The backend stores the information of the game in the form of a GameStateModel, this model also includes references to other Models (such as CatModel, PlanetModel, and CardModel). This model represents the current state of the game, and will be modified throughout the course of the game by the GameStateController. The GameStateController acts as the channel of communication between the Frontend view, and the Backend model, it facilitates this communication through a WebSocket which allows it to send requests to the frontend, as well as receive information from the view.

**Design Choices**

Firstly, many of our methods have changed from the initial UML diagram. First of all due to time constraints, only one Controller was used, this was purely because we needed to finish the implementation and didn’t have enough time to split the necessary parts into their own controllers. We decided it would be best if the GameStateModel acted as a hub to contain all of the information about the game, as it allowed for an easy way to access the game and modify it with the correct checks to ensure that no rules of the game are being violated.

We also chose to maintain a connection between the frontend and backend using a websocket. We thought this was the best choice for facilitating this connection, as if one client (view/frontend) updates the backend, the backend should notify every connected client that the gamestate has changed, which websockets allow us to do! Without websockets, the frontend would have to constantly check for changes by making queries regularly, which would impact performance.

The Game controller also contains a "lobby" class, this is mainly used before the game starts to store the cats that are trying to join the game. The main reason that this needs to exist is because the gameStateModel is not created until all players have joined the game, and in the meantime the cats that are waiting to start need to exist.