Date: Sep. 28, 2020
To: CS4500 Staff
From: Jake Hansen and Nick Thompson
Subject: Fish Systems

High Level Overview
We want to approach the Fish game system with two model-view-controller (MVC) stacks, representing the two major aspects of this system: the tournament and the game. The tournament MCV handles the tournament players and any information that persists from game to game. The game MVC is responsible for all information and logic that are specific to a single Fish game. The one point of overlap is the player model, which will exist within the data layer for both the tournament and the game.

Tournament MVC Stack
The tournament view will provide a GUI for users to register players for the tournament and view the bracket, including game rules and results. If necessary, this will be a visual interface for users to upload their AI players to the tournament.
The tournament controller will be responsible for communication between the tournament view, tournament model, and game controller. It will provide endpoints for users to register players and retrieve game results, for tournaments to be started, and for games to be started with specific rules (i.e. whether all games across a tournament use the same rules).
The tournament model includes the data and logic managing the bracket and all registered players. It will maintain a list of registered players, including their meta information (e.g. submitter and record), and the player's algorithm. It will also manage a bracket model, which will determine the players playing in each game and the file containing rules applied to each game.

Game MVC Stack
The Fish view is responsible for displaying the Fish game, including the game state and the moves each player makes. It may need to be interactive, depending on whether human players are permitted.
The game controller manages all interaction with a Fish game. The tournament controller will communicate with the game controller to create games with given players and get the results of the game. To the players of the game, it will provide endpoints to perform specific moves and to get the board state. It will enforce the communication protocols, but game logic will be left to the game model. The game controller's job is to make sure every message gets where it needs to go and employs the correct format. It will also perform error resolution, such as informing the player that it has requested an illegal move without the system crashing.
The game model manages the state of the Fish game. Through the controller, it will communicate with the players in the game to determine what moves they would like to make. It will act as the referee, implementing rule logic and preventing cheating.

Conclusion
The layout we have proposed here is designed to encompass all aspects of the Fish game and tournament system, but we have left avenues for scaling up or down if the requirements change. Decoupling the game and tournament MVC stacks comes with some overhead time cost, but allows for game specifications to change without impacting the tournament code and vice versa. The only tight couplings will be in the player model, and the interface through which the game and tournament controllers interact. This will also allow for more iterative development and demoing, as game progress can be demoed without tournament infrastructure in place.