Brian Chou (briancho), Krist Zhu (zikaiz)

Design Rational

We chose to represent a Connect4 game as a class because it makes sense to bundle all its functionality (game rules) and states into an easy to use package for clients. This class does one thing only (operating a Connect4 game in logic aspect) and does it well. Because our version of Connect4 uses a mutable board representation, it is better to hide that board behind the class and only allow clients to access a deep copy of that board. The copy also uses enum to represent the state of each cell, rather than arbitrary values such as 0 and 1. It is more understandable, and avoids leaky abstractions. Other public fields are all passed by value and not by reference, so clients are not able to change them. Many of our values which represent different aspects of the game are defined as enumerations since they are more robust and less error prone for clients.

We defined custom errors such as FullColumnError to give better indications towards the edge cases of the game. However, we tried to use built-in errors when possible to follow existing standards. To reduce the possibility of client errors, only add_token can be misused by the user. Detailed exceptions are raised when it is misused. Furthermore, we provided a is_valid_move method for the client to first check if the move is valid before calling add_token.

When designing the public methods of the Connect4 class, we focused on the core functionality needed to play the game. We identified those as getting the current player, getting the current board, getting the current game state, adding a token to the board, and starting a new game. Out of all the functionalities described above, starting a new game may not seem like a core requirement. However, we found that it was less work for clients to call a restart method rather than to instantiate a new Connect4 instance for every new game.

An important alternative we considered is whether to create a Player class to represent different players and to pass it into the Connect4 class. At first, it seemed natural for the game to be its own class and for players to be a separate class. First, the add_token method is in Connect4. Players' responsibility is to choose a valid int type column. Players can come up with the int in any way, Connect4 does not care, so there is no need to specify a method to generate the int. Third, It seems like a lot of overhead for clients to instantiate players when we can represent players internally without a class. Lastly, passing player instances into the Connect4 constructor can be error prone if the user passes in None or the same player twice. We wanted to reduce the chances of client misuse. Based on the API design principles, we removed the functionality of defining players and passing them into the Connect4 class to reduce the size and complexity of the API. Since there isn't any functionality for a Player class other than identifying which player is playing and which player won, we can do this within the Connect4 class and abstract it away from the client.

There are advantages to having a Player class, such as letting players have many fields like name, id, etc, and distinguishing players more easily. Creating bots can be as simple as subclassing the Player class. However, we do not see the value of this added complexity and chose to focus instead on the core gameplay of Connect4.