Hydra: Plan of Attack

Overview of Classes:

Board Class: The Board class is responsible for keeping track of all the cards in play, the heads in play, and each of the players that are playing. It stores all the heads and players in vectors for easy access, and calls on each head or player when necessary with easy access. It is also responsible for generating a pool of cards and randomly distributing them to each of the players. For example, if there are n players, then it will generate and distribute 54 * n cards. This class also keeps track of which player's turn it is and determines if any given play is valid given the condition of the game.

Play_Game Class: The Play_Game class is responsible for the operation of the game. It determines which player's turn it is, and updates the heads and the players cards as necessary. It is also responsible for the initialization of the game and keeps each component updated as necessary.

TextDisplay Class: The TextDisplay class is responsible for outputting the correct output in the correct format to the console. Depending on the action, we can call each separate function to output as desired. In detail, it will print the players, the heads in play, if a player is in winning state then the winning play, and the corresponding actions made by the player that requires input.

Player Class: The Player class keeps track of each player and the cards that they have. It keeps the draw pile and the discard piles separate in a vector, as well as the current and reserve card in strings. It is also responsible for identifying the player with the player number. Additionally, the corresponding methods in this class allow for modification of their deck as required. It also records the joker value that the player has chosen to use.

Head Class: The Head class keeps track of the set of cards that are in play. It is indexed with an integer to identify the age of the head. So, whenever a player has to draw, the program will look for the head with the lowest index. It also determines the top card in the head and updates it as necessary.

Key Dates:

Sunday, August 1: Finish Head class and Player class

Saturday, August 7: Finish Board and Play_Game class

Sunday, August 8: Finish TextDisplay class

Friday, August 13: Finish testing and add extra features if time available, and design document

Answer to Questions:

Question: What sort of class design or design pattern should you use to structure your game classes so that changing the interface or changing the game rules would have as little impact on the code as possible? Explain how your classes fit this framework.

- We could use an observer pattern. In this case, the Board that keeps track of all the cards and implements all the rules would be the concrete subject, and all of its dependencies (observers) would be the player, heads, play_game, and text display. Since these are all classes that are used and dependent upon the board class, they would be concrete observers. As such, any rules or interface change that is implemented has little impact and the program works just as intended.

Question: Jokers have a different behaviour from any other card. How should you structure your card type to support this without special-casing jokers everywhere they are used

- If a player gets a joker in their hand, then we store it as a "joker" string. Since they are not required to decide the value of that card until it is played, we keep it as that string. However, when this card is played, the user must decide on the value. As such, we do not store it as a joker, yet as a string of whatever the value chosen is, "A,2,3,4…,J,Q,K". Since we only need this value when the joker is at the top of a head, this is only temporary. As such, when a head containing a joker is drawn, it is reset to be the joker string.

Question: If you want to allow computer players, in addition to human players, how might you structure your classes? Consider that different types of computer players might also have differing play strategies, and that

strategies might change as the game progresses i.e. dynamically during the play of the game. How would that affect your structure?

- Adding varying computer players with varying play strategies would probably require defining them explicitly before the game has begun. I would make a separate computer class that incorporates different computer players with differing strategies. It would not change much of the structure in our already existing classes except for a few lines of code that calls on the computer class to make decisions instead of requiring input from the user. Depending on the game stage and the current condition of the heads, we would call the corresponding methods.

Question: If a human player wanted to stop playing, but the other players wished to continue, it would be reasonable to replace them with a computer player. How might you structure your classes to allow an easy transfer of the information associated with the human player to the computer player?

- We could add a field to the player class that determines if a human or a computer is playing that player. Then, when the play game class calls on a player for a turn, instead of reading from input, we would utilize various methods and play strategies to make the correct play. Ideally, we would use the computer class and call it within the player class instead of requiring input from the console. In this respect, we do not have to transfer any of the information -- but rather, the computer class utilizes the player class to make the correct play.