Plan

Project Timeline

- Dec. 1st
  - Set up project space
  - Create `main` function and implement command parsing for human players
  - Complete the `Card` class, followed by the `View` and `TextView` classes
- Dec 6th
  - Create `Player`, `HumanPlayer`, `Deck`, and `StraightsModel` classes
- Dec 7th
  - Create `StraightsController` class and implement game initialization
  - Implement rules logic
  - Refactor `main` input into the controller
- Dec 8th
  - Test with only human players
  - Bug fix
- Dec 10th
  - Create `ComputerPlayer` and its strategy classes
  - Test with both human and computer players
  - Bug fix
- Dec 13th
  - Start Design Document
  - Submit early
  - Test submission (for compilation)
- Dec 15th
  - Submit
  - Test submission (for compilation)

Questions

What sort of class design or design pattern should you use to structure your game classes to that changing the user interface from text-based to graphical, or changing the game rules, would have as little impact on the code as possible? Explain how your classes fit this framework:

Using the MVC design pattern, I am able to delegate all UI responsibilies to the `View` class. In this class, I have seperated all possible UI duties into seperate virtual functions. Then, to make a graphical interface all I would need to do is create a new child of the `View` class which re-implements all the virtual functions.

As well as this, the MVC architecture restricts all the rules logic to the controller class. Therefore, any changes in rules will require I only change this one class. However, there are some exceptions to this to keep cohesion high and to preserve encapsulation— I placed an `isLegalPlay` function in the model class. This way, the controller doesn't have to access any internal pile information in the model.

Consider thet different types of computer players might also have different 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 class structures:

I employed a strategy design pattern in the `ComputerPlayer` class. This way, at any moment I can polymorphically switch out one turn strategy for another without affecting how any other classes work. To keep cohesion low, I could encapsulate these functions within the `ComputerPlayer` class.

How would your design change, if at all, if the two Jokers in a deck were added to the game as *wildcards* i.e., the player in possesion of a Joker would choose it to take the place of any card in the game except the 7S:

In the constructor of the `Deck` class, I could add two jokers, which could be `Card`s with special suit/rank values (specified with a global constant). Functions like the `isValidCard` would have to change. Then, I would have to change the function `isLegalPlay` function in the Model class. This should be it, since the design has low coupling.