# Title: Rummy

Names: Lara Chunko, Maria Stull, Jake Swartwout

## Status Summary

Work Done:

- Created class files for all classes in the class diagram
- Began implementation for
    - Buttons
        - Button.java
    - CardsAndPiles
        - Card.java
        - CardPile.java
        - Hand.java
    - Display
        - BoardDisplay.java
        - DisplayUpdate.java
    - Observers
        - Published.java
    - Players
        - HumanPlayer.java
        - Player.java
        - RobotPlayer.java
    - Simulation
        - GameBoard.java
        - GameSimulator.java

  class diagram shows in more detail which classes are mostly complete vs need work
- State of the game:
    - underlying functionality is almost entirely done (only small tweaks to make)
    - i.e. creating players + cards, dealing, drawing, discarding, checking win/loss
    - A basic command-line interface is built out and functional to test the game

Changes or Issues Encountered:

- We're having issues figuring out the GUI
    - Have a few different options of ways to draw things
    - Not sure if our BoardDisplay observer setup is the right choice
    - Currently just set up a command line to be able to test if the logic works
- Template pattern may not work well with asynchronous code
    - We set up a template for taking a turn, but since one of those is the user, it has to pause in the middle somehow or otherwise be asynchronous.

    - While not necessarily breaking the pattern, this will be difficult to work around
- Checking for a win is more difficult than expected
    - Because there are so many different ways to win, it's difficult to implement every possible check
    - We are fairly certain that our implementation accurately checks, however with the size of the solution we want to do some more testing
    - We also had to build out an AI to have the robot make moves beneficial to them
    - The robot's AI seems to build towards a victory, but we aren't sure if it will make the right choices to actually make the final moves to win (ie maybe it will prioritize turning a run of 4 into one of 5, instead of getting the last card needed for its set)

Patterns:

Iterator:

- Our CardPile class (used for the deck of cards, and the discard pile) implements the Iterator pattern. This is useful for us because each cardpile behaves slightly differently (one is a stack, one is a queue), but we always want to access only the top card. By implementing Iterator, we can simply ask for the next card, and the iterator can deal with keeping track of which card is next to be returned.

MVC

- Not fully implemented (as planned) yet
- The player and gameboard objects are the Model aspect of this pattern; they keep track of the state of the game.
- View and Control are currently displayed in terminal, where the user can see the state of the game that they're permitted to see (i.e only their own hand), and give input for the actions they're allowed to take. Backend of implementation of view and control are scattered through a few classes, since we're still working on + testing the UI.

Template Pattern

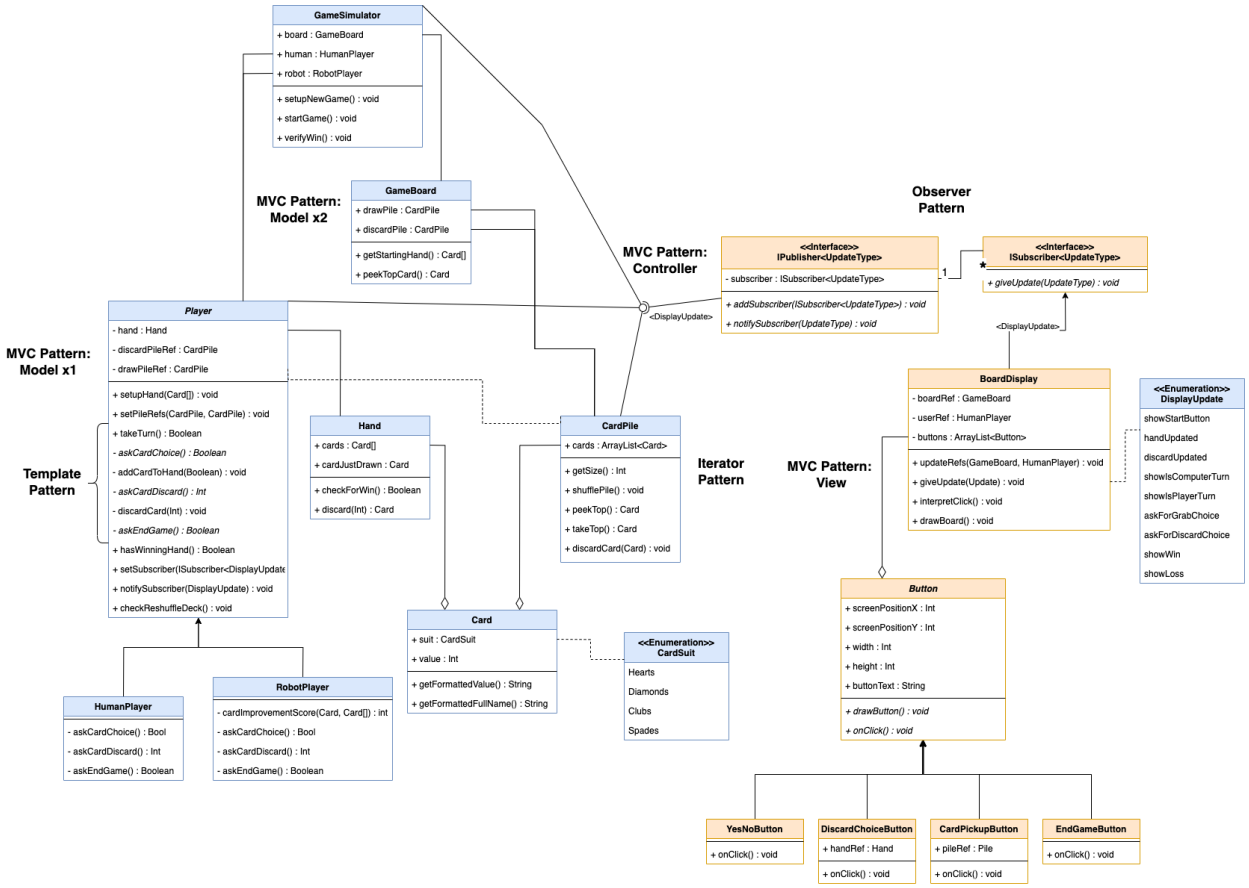
- We used the template pattern to set out the steps in a turn for each player
- Each player does the same steps in the process, so we can set out the functions we are going to call for each, but the user and the robot will implement some of them differently
- Our current steps include:
    - abstract askCardChoice - to ask whether they want the top of the discard or not
    - final addCardToHand - give them that card
    - abstract askCardDiscard - see which from their hand they want to get rid of
    - final discardCard - remove the card they chose from their hand
    - abstract askEndGame - see if they want to end the game or not
- The user's will include prompting for user input, while the robots are built out with logic/AI to make choices for them

Observer Pattern

- We are trying to use an observer pattern for our display
- Each item which is displayed implements the IPublisher interface, while the game board implements the ISubscriber interface.

- When the item detects some major change, it sends an update to the subscriber that it has changed
- Then, the subscriber can interpret the update, go read whatever data it needs, and update the display for the user

# Class Diagram




Class diagram:
blue label = mostly or entirely implemented;
orange label = still needs significant work

# Plan for Next Iteration

Estimate of how much more work needs to be done:

- Program
    - Buttons
    - Observers
- Create graphics / GUI

Plan for the final iteration to get the work done:

- Week of Nov. 29th
    - Buttons
    - Observers
    - Graphics
- Week of Dec. 6th
    - Final touches and documentation

Deliverables ready by 12/8:

- User will be able to run the Rummy Game
- The Rummy Game will open a game window that displays the game
- The user will engage with the game by clicking within the game window
- The game will start a game, shuffle the cards, and distribute 7 cards each to the player and computer
- The user will take turns with the computer to draw and discard cards until either the player or computer has a winning hand
- User and computer will be able to call victory and the game will score the hand to verify the win
- The game will end.
- OOAD patterns implemented: MVC Pattern, Template Pattern, Observer Pattern, Iterator Pattern