1 Assignment

Implement a system that is designed to play the "Gin Rummy" card game. Specifically your system should implement the following:

1. A game engine that can compete two players against each other and report the result of the game.
2. Three or more different player implementations.

We will provide you with an implementation of a Card class and an interface for player strategies, called `PlayerStrategy` that should be used in your implementation. We will also provide you with Meld classes to use. You should not modify the code provided, so that your player implementations can be run on anyone's game engine and your game engine can run anyone's player implementations.

You should create your Github repository from this link:

`https://classroom.github.com/a/a_GEkx49`

2 Gin Rummy

`https://en.wikipedia.org/wiki/Gin_rummy`

Note that our implementation described below might differ from the wikipedia page or variations of Gin Rummy. You should implement your game engine following our rules.

Gin Rummy is a card game that is played between two players. The game uses a full 52 card deck and the first player to reach 50 points wins the match. The starting player is selected at random and then afterwards players should alternate. The goal of the game is to form `Melds` and to have as few `deadwood` cards as possible. There are two types of `Melds`:

1. A set (cards of the same rank/number) that contains three or four cards.
2. A run (cards of sequential ranks) that contains at least three cards. A run can only be made with cards from the same suit. Aces are only counted as a low value. `A,2,3` is a valid run but `Q,K,A` is not.

Players will always have 10 cards in their hand. Any card in a player's hand that is not part of a `Meld` is considered a `deadwood` card. `Deadwood` cards have point values that sum to be the `Deadwood count`. Deadwood values are as follows:

1. An ace is worth one point
2. Numerical cards are worth their rank value
3. Face cards are worth ten points

The game is played as follows:

1. The deck game engine shuffles the deck and then deals 10 cards to each player.

2. The game engine then places the remaining cards face down to form the `Deck`. The first card from the `Deck` is flipped over to form the start of the `Discard Pile`.
3. The starting player then may choose whether to take the card in the `Discard Pile`. If they choose not to, the other player then has the chance to take the card in the `Discard Pile`. If they choose not to, then the starting player may take their turn by drawing from the `Deck`. Whoever goes first and takes their turn must then discard a card from their hand onto the top of the `Discard Pile`.
4. The other player then can choose to take a card from either the top of the `Deck` or the top of the `Discard Pile`. They then have to discard a card from their hand onto the top of the `Discard Pile`.
5. Step 4 is repeated (alternating players) until a player decides to `knock` or `call gin`. Knocking can only be done once a player's `deadwood count` is ten or less. Calling gin is simply knocking with a `deadwood count` of zero.
6. If the `deck` runs out of cards, the round is considered a tie and neither player receives points.

Once a player has decided to `knock` or `call gin` both players should reveal their cards split into `Melds` and `deadwood cards`. After the cards have been revealed, any `deadwood cards` in the hand of the player who didn't `knock` can be appended to the other player's `Melds` where appropriate.

1. If the knocking player decided to `knock`, that is they have a `deadwood count` that is greater than zero (and less than ten), then the knocking player should be awarded the difference between the `deadwood count` for each player in points.
2. If the knocking player decided to `call gin`, that is they have a `deadwood count` of zero, then the knocking player should receive 25 points plus the `deadwood count` of the opponent's hand in points. When gin is called, the opponent's deadwood cannot be appended to the `Melds` of the player who called gin.
3. If the knocking player decided to `knock`, but actually has a higher `deadwood count` than the opponent, the opponent should receive 25 points plus the difference in `deadwood count`.

The game ends when a player reaches 50 points or more after a round. This player is then declared the winner of the game.

3 Player Strategy

You should implement three or more different player strategies. These should follow the interface outlined below:

1. Receives and processes the initial hand dealt by the game engine at the beginning of each round.
2. Allows the game engine to prompt the player on whether they want to take the top card from the discard pile or from the deck.
3. Allows the game engine to prompt the player to take their turn given a dealt card (and returning their card they've chosen to discard).
4. Allows the game engine to ask the player whether they would like to `knock`.

5. Provides the player strategy information based off of their opponent's turn.
6. Provides the player strategy information based off of their opponent's hand and list of `Melds` at the end of the round.
7. Implements a method that can be called on the strategy to reset its internal state before competing it against a new opponent.
8. Allows the game engine to access the player's current list of `Melds`.

You should implement at least **three different player strategies** and compete them against each other. Your strategies should be different enough that there is a significant difference when competing against each other.

4 Game Engine

Your game engine should be capable of taking in two player strategies and competing them against each other. The game engine should play as many rounds as necessary until one of the player strategies win. This is considered a `game`. Your game engine should be able to play a specified number of `games` and then display the number of games each player won.

You should create a main function that takes all of your player strategies and competes them against your other player strategies. You should run each competition $1,000$ times. For each pairing, your main function should print the final score and who won/lost.

5 Competition

We will run a light hearted competition during this assignment. You will be able to submit your smartest `PlayerStrategy` implementation to compete against other `PlayerStrategy` implementations. Implementations will be competed against each other at multiple points throughout the week. We will release more details on this competition in the next couple of days.