1 Assignment

For this assignment, you will create a game engine and a player strategy to simulate the game SushiGo. Specifically you should implement the following:

1. A game engine that can compete four players against each other and report the results of the game.
2. A naive random player implementation that will choose a random card from the hand they are given.

We will provide you with a CardType enum and an interface for player strategies called `Player` that should be used in your implementation. We will also provide you with a `TurnResult` class to use. **You should not modify the code provided.** Communication between your game engine and players should happen solely through the `Player`. This is so that your player implementations can be run on anyone's game engine and vice versa.

You should create your Github repository from this link:

`https://classroom.github.com/a/rPpwoMu8`

2 Sushi Go

The rules to the game can be found here.

`https://www.gamewright.com/gamewright/pdfs/Rules/SushiGoTM-RULES.pdf`

If you don't like reading here is a video of how to play the game.

`https://www.youtube.com/watch?v=PL2SqLo5VVk`

You should implement SushiGo with the assumption that there will always be four players. Additionally implement the Pass Both Ways of the game where the direction of passing hands switches between rounds.

3 Chopsticks

Chopsticks will behave a little differently in your game engine than the game in real life. Instead of shouting "Sushi Go!", a player may choose to return two cards instead of one in `giveCardsPlayed` when electing to use chopsticks. To do this the player must have at least one chopsticks card on the table. The game engine should then insert these two cards into the player's table hand and remove the chopsticks card. The chopsticks card should be put into the hand before it is passed downstream to the next player.

4 Player Interface

We have provided a `GUIPlayer` player that provides a GUI and allows you to play against three AI players. We do not expect you do make any changes to `GUIPlayer`. You should implement a single naive player who always plays randomly using the player interface. The player interface is outlined below:

1. `init`: called once the player object has been instantiated. This gives the player object the names of all the other players in the game.
2. `newGame`: called at the start of each game. The game engine should not create new player objects if multiple games are being played with the same players.
3. `receiveHand`: called at the start of each turn to give each player their hand for the turn.
4. `giveCardsPlayed`: called to get the card(s) that the player wishes to keep from this hand. Most times this should return only one card. If the player has chopsticks and elects to use them, this will contain two cards.
5. `endRound`: called at the end of each round. This gives the player the total points that each player has accumulated so far in the game.
6. `endGame` called at the end of each game. This gives the player the total points that each player scored in the game.
7. `receiveTurnResults` called at end of each turn. This provides information on what actions each player took.

Additionally, each player should implement a `getName` function that returns the name of the player. Your game engine should be able to handle players whose names conflict (Hint: Add a unique value to the player name before calling `init`).

You should implement a single player strategy. This strategy should choose a random card from the hand they are given and return it. Start thinking about more advanced player strategies. We will be writing them next week.

5 Turn Result

We have provided a class `TurnResult` for you to use to represent the results of a single turn for a single player. This means that each turn should generate four `TurnResult` objects. The game engine should then pass these four `TurnResult` objects to each player so that they know what actions the other players took during that turn. A `TurnResult` is composed of:

1. `playerName`: the name of the player that this `TurnResult` corresponds to.
2. `cardsPlayed`: the card(s) that this player played during the turn.
3. `playerTableHand`: the card(s) that the player has on the table in front of them. This is the previous plays of the player during this round (excluding maybe chopsticks). This includes the `giveCardsPlayed` for this turn.

6 Game Engine

Your game engine should be capable of taking in four players and competing them against each other. The game engine should be able to identify and print the points scored by each player in a game. Additionally, the game engine should be able to play multiple games and then print how many games each player won.

You should create a main function that takes your players and plays them against each other for 100s of games.

7 Competition

We will be running a competition during this assignment. You will be able to submit your smartest player to compete against other players. We will release more details on this competition in the next couple of days.