1 Assignment

This week we are extending last weeks assignment. You must complete and fix any issues with your game engine from last week and write at least one competitive ai for the game. You may write more than one but only one will be run in the competition. For reference I have retained the assignment from last week in this document.

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

7.1 How to submit code

Push code to your mater branch. We will pull this when we run the competition.

In order for us to run your Player against our baselines strategies or in the tournament, you must have a package named "competition". This package can be anywhere in the src directory, as long as it is named "competition". The following are all fine examples.

- `package com.company.competition`
- `package competition`
- `package com.company.players.competition`

7.2 How to make sure your code compiles

Do not put any packages inside your competition package. Java files in packages which do not end in "competition" will not be executed. For example`mypackage.competition.ExamplePlayer` will be compiled, `competition.mypackage.ExamplePlayer` will not be compiled.

Do not reference any classes that you personally created which are outside the competition package.

- References to classes we provided like the Player class or CardType class are perfectly fine.
- If you have other classes you wrote that your strategy needs, they need to be in the competition package.
- This means you either need to move them to the competition package, or make a copy of the class to put in the competition package.
- Yes, repeating code is bad, but points will not be taken off for having duplicate classes in the competition package.

7.3 How to ensure your code is valid

Put only one Player in the competition package, otherwise we will not know what Player to run.

- **Do not print things in your Player:** it clogs up our logging of the games being run and we will not execute your code if your strategy prints to console
- **Don't cheat:** Don't attempt to use reflection to modify your player's score, hand or anything else. Reflection and other similar techniques are forbidden.
- **Don't write malicious code:** Things like calling `System.exit()` in your Player not only won't work but will earn you a 0 on this assignment.

7.4 Miscellaneous

Non .java files in the competition package will be copied and put next to your .class files when we compile your code This means that if you really want to build a machine learning model for a strategy, you can do exactly that. Note we do not expect anyone to actually do this.

7.5 Baseline

You are required to have an AI that preforms well against baseline strategies. Specifically you must consistently $> 90\%$ win games against only the random AI. Additionally you must also beat other AIs at a $> 70\%$ win rate. We will have several AIs in order of increasing difficulty that will compete with your AI. You will get credit based on the number that you can beat reliably.

Your performance in the competition will not affect your grade.

To test with our AIs submit to the competition. This will both run in the competition and also run against our AIs.

8 C++ Preparation

In order to prepare for using C++ in the following weeks you will need to show you can build a C++ program. Specifically the HelloWorld program. There is nothing to check in for this simply show that you can compile the program at your code review.