Assignment: Player class

Introduction to Software Engineering CSCI 2360

Objectives

- Learn how to run Python programs in a shell and how to capture output from the program into a file.
- Learn how to understand the output of the Player class.
- Learn how to create a YAML configuration file to configure players.
- Learn how to build an Excel file with bid strengths for cards.
- Begin devising your bid strategy for the end of semester Pitch tournament.

Overview

You will not write new code in this assignment. Instead, you will gain experience using a command shell. This is a fundamental skill for a software engineer in industry.

You will also learn how to understand the output of the Player class so that you can analyze the effectiveness of various card strengths in calculating a bid (suit and bid value) from a dealt hand.

Based upon your analysis of the output of the sample bid configurations, you will create your own bid weight definition as an Excel file and a configuration file to demonstrate your bid weights in use.

The process of understanding and devising your bid strategy is an example of acquiring domain specific knowledge about the problem you are trying to solve and both configuring your system to use that knowledge and assessing the effectiveness of your implementation. This type of activity is increasingly important to software engineers as the mechanics of programming are increasingly performed by tools (e.g., AI) and the software engineer is increasingly focused on effectively solving problems (e.g., business problems) rather than simply constructing code.

Background

Executing `player.py` in a shell

In the examples below, `>` represents the shell command prompt. Depending upon your operating system and configuration, your command prompt may look different from this.

After opening a shell, navigate to the directory holding your code. This will likely require using the `cd` (change directory) command to navigate the file system. For example, if your code is in the `player_student` subdirectory within your `source` directory, you might use the following command to change to that directory.

Here is an example on a Mac.

```
> cd source/player_student
```

Windows uses a backslash (`\`) rather than a forward slash, so the following example is for Windows.

```
> cd source\player_student
```

To run `player.py` using the following command. By default, this produces an output for 4 players for a single hand. Look at the output to find the cards dealt to each player and what each player considered their best hand and their default bid.

```
> python player.py
```

The `player.py` file has several command line options. Use the following command to see the help information for the options.

```
> python player.py --help
```

Try the following command to show the detailed information about each player's bid. Investigate all the detail on each bid. Can you determine how the bid strength is calculated for each suit?

```
> python player.py -d
```

You can evaluate more than one hand by specifying the number of hands to process. In the example below, we specified 2 hands to be dealt.

```
> python player.py -d --hands 2
```

You can have the program pause after each hand by adding the `-i` option to the command line.

```
> python player.py -d --hands 2 -i
```

You can capture all the output from the program by redirecting the output of the program using `>`. When doing this, leave the `-i` option off because you won't be able to see the prompts for pressing enter when the output is redirected. In the example below, the output is redirected into a file called `test.out`.

```
> python player.py -d --hands 2 > test.out
```

Executing `player.py` with a configuration file

The file `bidders_config.yaml` is a sample configuration file for `player.py`. This file uses YAML (Yet Another Markup Language) as the syntax to specify attributes about players. Use the VS Code editor to see the

contents of the configuration.

There are 4 players configured. Each player has a `name` and three or four `strategies` attributes.

The bid strategy in `player` uses strength weights for each trump card. The values of the weights are specified in an Excel file (`bid_strength_file`) with a particular sheet name (`bid_strength_sheet`). The weight for each card is summed for each possible suit to compute a base bid value for each suit. The sum of the weights represents how many points the player believes they can make in the hand using only the cards in their bidding hand. Look at the file `bidders.xlsx` for a sample Excel bid strength file.

The `aggressiveness` strategy is a numeric value added to the highest bid computed by the card strengths. The `aggressivness` value in essence represents the bidding player's opinions about how many additional points beyond those from their base bidding hand that they are likely to get from either (1) cards in the deck after bidding, and/or (2) cards held by their partner.

The `restraint` strategy is a numeric value that is subtracted from the card strength plus the aggressivness value. This value is only subtracted if the bidding player's partner currently has the highest bid. While this can happen in the full Pitch implementation, it does not happen in the `player.py` code, so this value will never affect the calculations. However, keep this value in mind for future strategy in the full Pitch game.

Here is an example of how to run `player.py` file specifying the `bidders_config.yaml` configuration file.

```
> python player.py -d -c bidders_config.yaml
```

As before, you can use redirection to capture the output of the run.

```
> python player.py -d -c bidders_config.yaml > test.out
```

Task Description

Analyze the default bid configurations

Run several instances of `pitch.py` using the sample configuration and analyze the bids produced for different dealt hands. Consider the impact of bid strengths for each card and for the `aggressiveness` strategy.

Devise your own bid strength and aggressivness strategy

Copy the `bidders.xlsx` file to a new file called `my_bidders.xlsx`.

Edit `my_bidders.xlsx` to implement your desired bid strengths. To do this, rename the sheet called `Mario` to your first name, then adjust the weights as you think appropriate.

Copy the `bidders_config.yaml` file to a new file called `my_bidders_config.yaml`.

Edit `my_bidders_config.yaml` to change the name of the `Mario` player to your name. Change the `bid_strength_file` for all players to your new bid strength file. Change the `bid_strength_sheet` for your

player to reference the sheet with your name in the Excel file.

Change the `aggressiveness` and `restraint` values for your player as you deem appropriate.

Assess your new strategy

Run `player.py` several times to analyze the impact of your new strategy and adjust your strategy as you see fit. You may wish to run several hands to check your strategy

```
> python player.py -d -c my_bidders_config.yaml --hands 10
```

Capture your final assessment file for submission

When you are satisfied with your strategy (at least for now), capture the output from `player.py` using your strategy into a file named `my_tests.out`. Run 5 hands for your final assessment file.

```
> python player.py -d -c my_bidders_config.yaml --hands 5 > my_tests.out
```

Submit your work by committing all your work to your GitHub project.

Submission Requirements and Grading

- `my_bidders_config.yaml` properly created, updated, and committed with all changes to names, strategies, etc. as described above. (5 points)
- `my_bidders.xlsx` properly created, updated, and committed. The weights submitted must be modified from the sample values. (10 points)
- `my_tests.out` properly created and committed. The output must demonstrate successful execution with your new configuration files. (10 points)
- Submit a link to your GitHub repository on Blackboard.

Due Dates

The due date is specified on Blackboard.

Good luck! Remember to check the requirements and expectations carefully and ask questions if you need clarification.