GROUP PROJECT

COMP202, Fall 2020

Due: Wednesday, November 18^{th}, 11:59pm

Please read the entire PDF before starting.

Question 1:	100 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

To get full marks, you must follow all instructions, including the following. Large penalties will be applied if any of these are not followed.

- Make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, you will not receive a grade for those files/functions.
- Make sure that your code runs without errors. Code with errors will receive a very low mark.
- Put the names & student IDs of all your group members in a comment at the top of all your `.py` files.
- Name your variables appropriately. The purpose of each variable should be obvious from the name.
- Comment your work. A comment every line is not needed, but there should be enough comments to fully understand your program.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing. Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.
- **You are not allowed to use anything seen after the Week 10.4 video or not seen in class at all. You will be heavily penalized if you do so.**
- Make sure to add an appropriate docstring to **all** the functions you write. The docstring **must** contain the type contract, description of what the function does, and at least 3 examples of calls to the function. You are allowed to use *at most* one example per function from this PDF.
- **Do not include any function calls in the main body of any of your files.**
- Submit early and submit often to codePost. Make sure that your code passes all tests. If it does not, then there is something wrong with your code. Some of these tests are drawn from the examples given in this PDF. Your code should output the same as those examples. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests that may use inputs not seen in the examples.

In this project we ask you to write an automated player for the card game Three Thirteen. Write your code in a file called `ai_player.py`.

Three Thirteen is a rummy variant for two or more players (we will play with four or more). It uses two decks of 52 playing cards (104 in total). (The following rules have been adapted from `https://www.pagat.com/rummy/3-13.html`.)

The game has eleven rounds. In the first round, three cards are dealt to each player; in the second round, four cards are dealt; this continues until the last round where thirteen cards are dealt to each player. The remainder of the cards in each round are placed face down in a pile called the **stock**. The top card of the stock is flipped face up and put in a separate pile called the **discard pile**.

The object of the game is to arrange all the cards in your hand into combinations. There are two types of combinations.

- a **group** of three or more cards of the same rank, such as Five of Hearts, Five of Clubs, Five of Spades. It is possible for a set to contain identical cards as we are playing with more than one deck (such as Six of Clubs, Six of Clubs, Six of Hearts).
- a **sequence** of three or more cards of the same suit with consecutive rank, such as Five of Hearts, Six of Hearts, Seven of Hearts.

Combinations can contain more than three cards (such as four sevens, or the 8-9-10-Jack-Queen of Clubs). However, you cannot count the same card as part of more than one combination. If the Five of Spades is included in a group, it cannot be included in a sequence (unless there are two Five of Spades in your hand).

Aces rank high in this game, so Q-K-A of a suit is a valid sequence, but A-2-3 is not.

In each round there is a **wildcard rank**. It is the rank equal to the number of cards dealt that round. For example, in round 1, where three cards are dealt to each player, all Three's are wild cards. In round 8, Ten's are wild cards; in round 9, 10 and 11, Jacks, Queens, and Kings are wild, respectively.

A wildcard can be included into any group or sequence. For example, if the wildcard rank is King, and the hand currently contains a Two of Hearts, Two of Clubs and King of Hearts, then these three cards can be put into a group, because the King can stand in for another Two. Or, if the wildcard rank is Seven, and the hand contains a Two of Hearts, Four of Hearts and Seven of Spades, then these three cards can be put into a group, because the Seven can stand in for the Three of Hearts. Note that we are providing you with updated `is_valid_group` and `is_valid_sequence` functions that recognize groups/sequences that contain wildcards.

The starting player of the first round is chosen randomly. Players take turns clockwise around the table. A turn consists of **drawing** one card (either the top card of the face down stock pile, or the top card of the discard pile), and then **discarding** one card to the top of the discard pile. In each successive round, the starting player will be the player to the right of the previous starting player.

A player can **finish** the round (also known as 'going out') if, after drawing, they are able to arrange all the cards in their hand except one into separate groups/sequences, and then discard the final card. Each of the other players is then allowed one more turn. When they finish and it is the winning player's turn again, the round ends and the round scores are calculated.

At the end of a round, each player arranges as much of their hand as possible into groups and sequences. Any cards that they cannot include in a group or sequence are counted as **penalty points** against them. An Ace is 1 penalty point; Twos through Tens are their numeric value; and Jacks, Queens and Kings are 10 penalty points each.

These scores are accumulated from round to round. Whoever has the lowest score (the lowest amount of penalty points) at the end of the eleventh round is the winner.

We have already written certain functions for this card game in Assignment 1 and 2. You will be provided the solutions to those questions for this project, in addition to other code to enable the game to be played. Please verify that you have downloaded the zip file along with this PDF, which contains the following files. You may import one or more of these modules if you need to use some of the functions and/or global variables in your code. **Note: Do not modify these files as you will not be able to submit them.**

- `card.py` has the functions and global variables from `card1.py` and `card2.py`
- `arrangement.py` contains the `is_valid_group` and `is_valid_sequence` functions from Assignment 2, as well as some other functions dealing with arrangements, including `get_arrangement(hand, wildcard_rank)`. This latter function takes in a hand of cards and returns a nested list containing the arrangement (groups and/or sequences) that minimizes the number of un-arranged cards while also minimizing the total penalty points of the un-arranged cards.
- `human_player.py` and `random_player.py` contain the solutions to the Assignment 2 questions for those modules.
- `game.py` contains the code necessary to play the game (dealing cards, calculating scores, playing each round and printing out the current game state). You can edit this file to add or remove print statements if you prefer (for example, on line 139, you can comment out the input call so that the game runs without pausing after each turn).
- `play.py` is the file you will run in order to play the game. You will need to modify line 6 of this file in order to decide which players will play the game. Line 6 contains a list of module names (strings) corresponding to players. Right now, it has the `random_player` twice. If we were to run the file, then there would be two players in the game and both would play randomly (according to the random player we described in Assignment 2). If you want your automated player to be in the game, you will have to add a new element `ai_player` to the list (or replace one of the existing elements). You can also add `human_player` to the list if you want to be able to control one of the players yourself (as described in Assignment 2). Note that you will not be able to submit this file, as we will be using our own list of players to test your code, but it is OK to modify line 6 so that you can test your player.

Your task in this question is to create your own player, in the file `ai_player.py`. You must implement the following two functions:

- `draw(hand, top_discard, last_turn, picked_up_discard_cards, player_index, wildcard_rank, num_turns_this_round)`: This function decides from where to pick up a card: from the discard pile or from the stock. It must return either the string `stock` or `discard`. Note that it can only draw from the discard pile if the `top_discard` card is not `None`. To aid in your decision, the function has the following parameters:
  - `hand`: a list of the cards currently in your hand
  - `top_discard`: the card currently on the top of the discard pile (if the pile is empty, `None` instead).
  - `last_turn`: a boolean variable that indicates if the round is about to end (i.e., if a player has declared that they have finished for the round, then `last_turn` will be `True`, meaning that this is your last turn for the round).
  - `picked_up_discard_cards`: a nested list containing the cards that other players have picked up from the discard pile so far in the current round. (As the top card of the discard pile can be seen by everyone, then everyone will know if someone picks up a card from there.) In particular, the sublist at index `i` is the list of cards from the discard pile picked up by player `i` so far during this round.
  - `player_position`: an integer corresponding to your position at the table. You can use this in conjunction with the above parameter to see what cards the person to your immediate left and right picked up from the discard pile during this round.

  - `wildcard_rank`: an integer corresponding to the current round's wildcard rank (i.e., one of `THREE`, `FOUR`, ..., up to `ACE`. `num_turns_this_round`: an integer for the number of turns in total that all players have taken during this round so far.

- `discard(hand, last_turn, picked_up_discard_cards, player_index, wildcard_rank, num_turns_this_round)`: This function decides which card to discard from your hand. It must return one of the cards in the `hand` list. It has the same parameters as `draw`.

The provided `game.py` code will call your (and other player's) `draw` and `discard` functions as necessary (each turn of a round). After you discard, your hand will automatically be checked to see if all cards in the hand can be arranged. If so, your player will automatically finish the round and the other players will be given one more turn until the round ends and round scores are calculated.

If, after a certain number of turns in a round, no one has gone out, then the round will end due to timeout, and each player will be penalized for all the cards in their hand that they cannot currently arrange. The maximum number of turns per round is currently set to 3000 times the number of players in the game, but we may alter this value during testing if it becomes necessary. (If you are often getting timeouts when testing your player against the random player, then your player may not be performing well.)

Hints

These are just some hints which you may want to implement, but there are many other ideas you can think up!

- If you already have a group/sequence in your hand, then you should not discard any of those cards. Similarly, if you have something that is almost a group/sequence, you should probably not discard those cards.
- If the round is nearing the end, you would probably want to discard the cards of the highest penalty in your hand.
- If you can find out that another player is picking up a lot of a certain rank or suit, then if you have a chance to draw a related card, you may want to draw it to put a wrench in their strategy.
- If you have a chance to draw a wildcard, it is probably best to take it. Similarly, it may not be a good idea to discard a wild card, if you have one.

Grading

When we grade your submission, we will first check if your two functions return appropriate values in all cases (e.g., that `draw` always returns either the string `stock` or `discard`, and that `discard` always returns a valid card from the hand). If not, you will receive 0 points.

If they do always return appropriate values, we will then test your player against three random players in a series of 1000 games. If your player has more wins on average than any other random player, then you will receive 85 points out of 100 for this project (minus any penalties applied for not following the instructions on page 1).

If your player does get more wins on average than the random player as described above, then the remaining 15 points will be dependent on your player's performance in a class-wide tournament. The tournament will be a single-elimination format and proceed as follows. Submissions (which beat the random player) will be randomly assigned into groups of 4. A large number of games will be played amongst the players of each group. The bottom two players of each group (in terms of wins) will be eliminated, and new groups of 4 will be formed from the top 2 players of each group. The tournament will continue in this fashion until 4 players remain. We will announce these top 4 players as our winners. If your player places in the top 30% of the entered submissions, then the amount of points you get out of 15 will depend on your ranking (the higher the ranking, the more points out of 15 you will get).

Finally, note that this is a group project. We expect everyone in your group to participate equally in this project. When you make your final submission, you must send a direct message over Slack to your group's TA. In this direct message you will rate each of your group members contribution to the project on a scale from 1 to 3 (3: worked very hard, 1: did little to no work). If a group member receives a large number of 1's from their teammates, they may be asked by the instructors to take an oral exam on the project to determine their mark.

What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. As this is a group project, one member of the group should create a group submission and send the others the link to join the submission. All other group members must then join the submission. Any deviation from these requirements may lead to lost marks.

`ai_player.py`

`README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.

This assignment is a group assignment. It must represent the effort of only your group members. You are permitted to discuss ideas with other groups, as long as no written notes are taken. If you do discuss it with anyone from another group, please make note of those people in this `README.txt` file. If you didn't talk to anybody outside your group nor have anything you want to tell the TA, just say "nothing to report" in the file.