Homework 3: Water Tank

(Deadline as per Canvas)

This homework covers concepts learned in Modules 1-3. In this homework, we will be implementing the game *Water Tank*, a card-based game.

About the Assignment

Water Tank is a competitive card game played between two players. In your case, the players will be a human player and a computer player. Each player starts with an **empty** water tank, which they need to fill. **The goal is to be the first player to fill their tank.** A tank is filled if it reaches the value of **75 to 80 units (inclusive)**. The human player's moves are decided by the user playing the game, by asking for input, and the computer's moves are decided by the program that you will write.

For the section that asks you to program a strategy for the computer, what we want you to do is come up with a reasonable enough strategy that ensures that a computer makes a logical decision. Example, if the computer's tank is missing just 1 unit of water to fill the tank, and that card happens to be in the computer's hand, then the computer should use that card in their move. So, unlike your previous assignments, this one has a creative component to it.

There are two types of cards: water cards and power cards. There will be a pile for each type of card (one pile for water and one pile for power). Each **water card** has a value that represents the amount of water that it contributes to the tank. When a water card is played, that player adds the specified amount of water to their tank. **Power cards** allow players to perform special actions:

- SOH (Steal Opponent's Half): Take half the water in your opponent's tank and add it to your own
- DOT (Drain Opponent's Tank): Empty your opponent's tank completely
- DMT (Double My Tank): Double the current value of your own tank

Players take turns either using a card or discarding a card. If a player discards a card, it goes to the bottom (last index) of its respective pile. Once the player has used or discarded a card, they draw a new card, from the top of the pile (index 0), *of the same type as the card they just used or discarded*.

If a player's water level exceeds their tank's maximum fill value, an **overflow** happens. In the case of an overflow, extra water sloshes out of the tank. The amount of water that remains in the tank is determined by a formula: **remaining water = maximum fill value - overflow**

For example, if a player’s tank level goes to 90, and its maximum fill value is 80, the overflow is 10. Deduct 10 from the maximum fill value to find the remaining water in the tank, which is 70 in this case.

The game continues in turns until one player’s tank is filled. A tank is considered filled when the tank level is between the minimum and maximum fill values (inclusive). **The first player to fill their tank wins the game.**

If a pile of cards is exhausted, the setup function for that type of card will be called to replenish the pile.

Required Functions

The following functions must be present in your code with these given names and function signatures exactly. For the autograder to run properly, **do not change the function names or the parameters**. Do not add optional parameters or change the return types. If a function returns something, make sure what it returns is consistent with what’s mentioned in the details below. Be sure to add docstrings to all of your functions and comments to your code. The *water_tank.py* should be placed in the submit folder and should not be renamed, otherwise the autograder will fail to locate your program.

The following are the required functions you have to implement:

(Note: Some functions may be used in another function)

get_user_input(question):

- Prompt the user with the given question and process the input.
- Return the post-processed user input.
    - Remove leading and trailing whitespaces.
    - If the length of the user input after removing leading and trailing whitespaces is 0, reprompt.
    - If the input is a number, cast and return an integer type.
    - If the input is a power card, return the power card as an *uppercase* string.
    - If the input is any other string, return the string as a *lowercase* string.

setup_water_cards():

- Create a *shuffled* list of water cards with the following values and quantities:

Value	Quantity of Cards
1	30
5	15
10	8

- Hint: Use the shuffle function from the random module.
- Return the water cards as a list of integers.

setup_power_cards():

- Create a *shuffled* list of power cards with the following values and quantities:

Value	Quantity of Cards	Description
SOH	10	Steal half opponent's tank value. If the opponent's tank value is an odd integer, it will truncate the decimal value (Example: ½ of 5 is 2) Hint: You may use the cast to *int*
DOT	2	Drain opponent's tank
DMT	3	Double my tank's value.

- Hint: Use the shuffle function from the random module.
- Return the power cards as a list of strings.

setup_cards():

- Set up both the water card and power card piles as described in the *setup_water_cards* and *setup_power_cards* functions.
- Return a 2-tuple containing the water cards pile and the power cards pile, respectively. (Each pile should be represented by a list.)

get_card_from_pile(pile, index):

- Removes the entry at the specified index of the given pile (water or power) and modifies the pile by reference.
- This function returns the entry at the specified index. HINT: Use the *pop* function

arrange_cards(cards_list):

- Arrange the players cards such that:
    - The first three indices are water cards, sorted in ascending order.
    - The last two indices are power cards, sorted in alphabetical order.
- This function doesn't return anything.

deal_cards(water_cards_pile, power_cards_pile):

- Deals cards to player 1 and player 2. Each player would get 3 water cards and 2 power cards. Then, call the *arrange_cards* function to arrange the cards.
- When dealing, alternately take off a card from the first entry in the pile. Example:

  Initially, the water and pile would be:

  water_pile = [5, 1, 1, 1, 1, 5, 1, 10, 1, 10, 5, 1, 1, 5, 1 , ...]

  power_pile = ['SOH', 'SOH', 'DOT', 'DMT', 'DOT', 'SOH', 'SOH', ...]

  After dealing, player 1 and 2 would have the following cards in their hand:

  player_1_cards = [5, 1, 1, 'SOH', 'DOT'] ⇒ arrange to [1, 1, 5, 'DOT', 'SOH']

  player_2_cards = [1, 1, 5, 'SOH, 'DMT'] ⇒ arrange to [1, 1, 5, 'DMT', 'SOH']

  Then, the piles would now be reduced to:

  water_pile = [1, 10, 1, 10, 5, 1, 1, 5, 1 , ...]

  power_pile = ['DOT', 'SOH', 'SOH', ...]

- Return a 2-tuple containing the player 1's hand and player 2's hand, respectively. (Each hand should be represented by a list.)

apply_overflow(tank_level)

- If necessary, apply the overflow rule discussed in the “About the Assignment” section of this assignment.

  remaining water = maximum fill value - overflow

- Return the tank level. If no overflow occurred, this is just the starting tank level.

use_card(player_tank, card_to_use, player_cards, opponent_tank):

- Get that card from the player’s hand, and update the tank level based on the card that was used. This does not include drawing a replacement card, so after using the card, the *player_cards* size will only be 4 cards.
- Apply overflow if necessary.
- Return a 2-tuple containing the player’s tank and the opponent’s tank, respectively.

discard_card(card_to_discard, player_cards, water_cards_pile, power_cards_pile):

- Discard the given card from the player’s hand and return it to the bottom of the appropriate pile. (Water cards should go in the water card pile and power cards should go in the power card pile.) The bottom of the pile is the last index in the list.
- Same as *use_card()*, this function does not include drawing a replacement card, so after calling this function, the *player_cards* size will only be 4 cards.
- This function does not return anything.

filled_tank(tank):

- Determine if the tank level is between the maximum and minimum fill values (inclusive).
- Return a boolean representing whether the tank is filled.
    - This will be True if the tank is filled.

check_pile(pile, pile_type):

- Checks if the given pile is empty. If so, call the pile's setup function to replenish the pile.
- *pile_type* is a string to determine what type of pile you are checking ("water" or "power")
- This function does not return anything.

human_play(human_tank, human_cards, water_cards_pile, power_cards_pile, opponent_tank):

- Show the human player's water level and then the computer player's water level.
- Show the human player their hand and ask them if they want to use or discard a card. If the human player enters an invalid answer, reprompt until a valid answer is entered.
- Carry out the human's turn based on the action they have chosen (based on user input). Be sure to use the *get_user_input* function.
- Print the card the human player uses or discards. If the human player enters a card to use or discard which is not in their hand, reprompt until a valid card is entered.
- Remember to handle potential overflows.
- Once the human has used or discarded a card, draw a new card *of the same type they just used/discarded*.
- Make sure that the human's hand is still properly arranged after adding the new card.
- Return a 2-tuple containing the human's tank level and the computer's tank level, respectively.

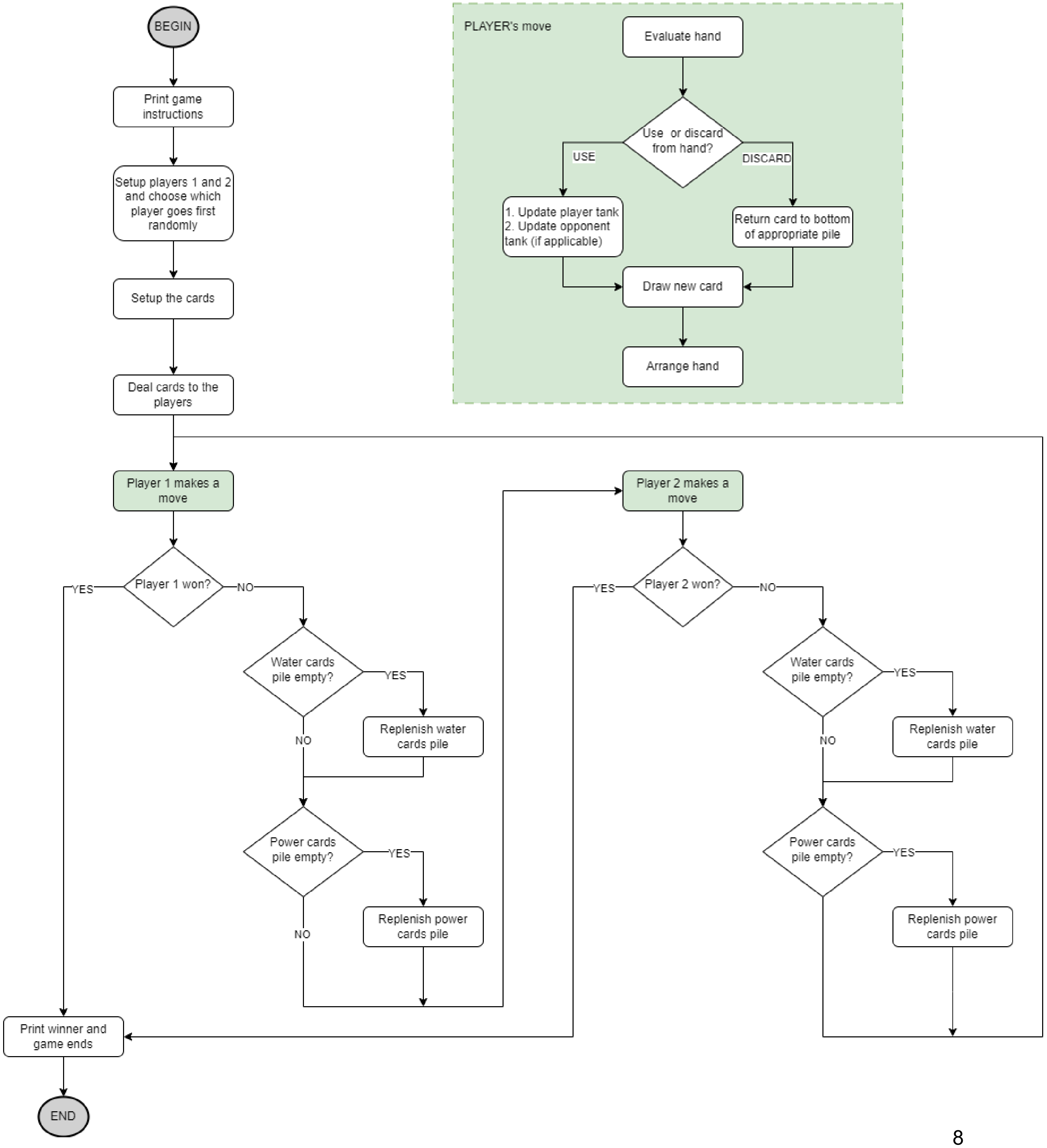
computer_play(computer_tank, computer_cards, water_cards_pile, power_cards_pile, opponent_tank):

- This is the function where you can write the computer's strategy.
- You are supposed to be somewhat creative here, but make sure your code is deterministic (not random).
- The computer's strategy should consider all of its cards when playing. For example, you should not have a strategy where the computer always plays the first water card or the first power card.
- The computer should not "cheat." They should not be able to see any cards from the human's hand, the water card pile, or power card pile. When they draw a card, they should only see that card and no cards from the rest of the water or power card pile.
- This function should carry out the computer's turn based on the action that the computer chooses.

- Remember to handle potential overflows.
- Print the card the computer player uses or discards.
- Once the computer has used or discarded a card, give them a new card *of the same type they just used/discarded*.
- Make sure that the computer's hand is still properly arranged after adding the new card.
- Return a 2-tuple containing the computer's tank level and the human's tank level, respectively.

main():

- The main function would be where you would structure the flow of your game, calling the functions defined above, as needed.
- As noted in the "User Interface" section of this assignment, remember to choose a random player to go first.
- For each turn, a player can use a card (then draw a new card) or discard a card (then draw a new card).
- To help you with the structure, a flowchart of the game is shown:


BEGIN
Print game instructions
Setup players 1 and 2 and choose which player goes first randomly
Setup the cards
Deal cards to the players
Player 1 makes a move
Player 1 won?
YES
NO
Water cards pile empty?
YES
Replenish water cards pile
NO
Power cards pile empty?
YES
Replenish power cards pile
NO
Player 2 makes a move
Player 2 won?
YES
NO
Water cards pile empty?
YES
Replenish water cards pile
NO
Power cards pile empty?
YES
Replenish power cards pile
Print winner and game ends
END
PLAYER's move
Evaluate hand
Use or discard from hand?
USE
DISCARD
1. Update player tank
2. Update opponent tank (if applicable)
Return card to bottom of appropriate pile
Draw new card
Arrange hand

Program Output

We have provided *template_behavior_*.txt* files which show some sample runs of the program -- yours should provide similar information. You can test this using the **"Run Your Program".**

User Interface

You're free to create your own user interface for the game, as long as it makes sense for the user playing. You should include instructions for how to play at the beginning of the game.

Your program should randomly choose a player to go first. A turn should proceed as follows:

- Print whose turn it is.
- Print the player's tank level and their opponent's tank level.
- On the human player's turn, print their hand and ask if they want to use or discard a card. Print the new hand after drawing a new card.
- On the computer player's turn, do not print their hand and the new card they draw.
- Print what card the player uses or discards.
- Print the new tank values for the player and their opponent after a player uses or discards a card.

Evaluation

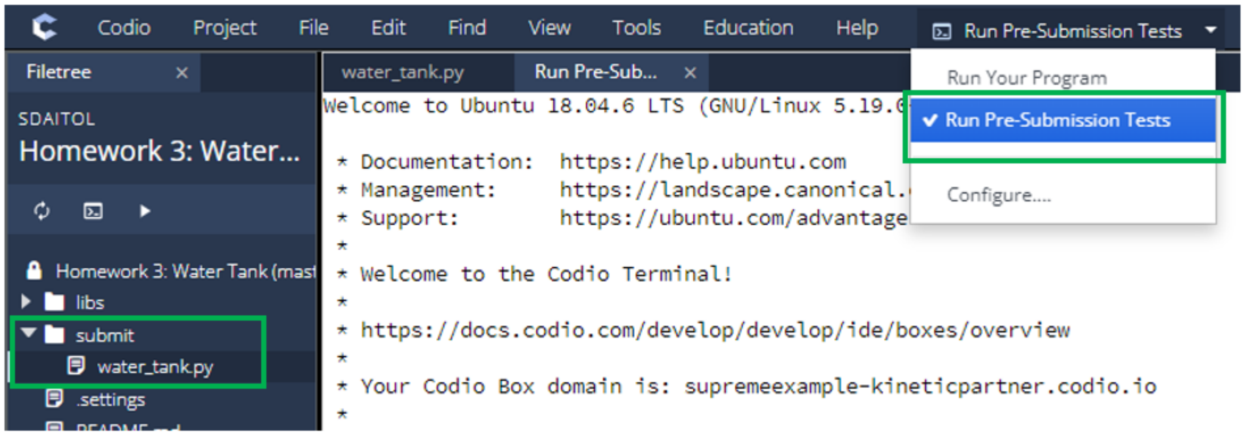
The primary goal of this assignment is to get familiar with lists and tuples while having fun creating a game.

While we want you to spend some time coming up with a strategy for the computer, that is NOT the primary part of the assignment. Any simple but reasonable strategy will have you doing some fun things with lists. If the human player does absolutely nothing at all, that is, if they choose to simply discard a card every turn, then we want the computer to be able to win. Your computer should be smart enough to beat a player who does nothing logically.

You will be evaluated on the following:

1. Program correctness - 80 points (Autograded)
   This section will be fully autograded. All tests are available to you by choosing "Run Pre-Submission Tests." If there are indentation errors, syntax errors, or incorrect file names and signatures, the autograder will fail to run. Please read the error messages

and resolve them, otherwise you will get a zero for the autograder portion.



```
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.19.

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.
 * Support:        https://ubuntu.com/advantage
 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/develop/develop/ide/boxes/overview
 *
 * Your Codio Box domain is: supremeexample-kineticpartner.codio.io
 *
```

2. Correctness of the *main* function - 10 points (Manually Graded)
   This section is where we will evaluate your game flow as described in the instructions. This also includes calling in the appropriate required functions as needed (Example: When getting user input, invoke *get_user_input()* and not repeating those lines of code again.)

3. Code setup and style - 4 points (Manually Graded)
   Correct code setup means that your code runs properly in Codio. This means that there are no indentation errors or syntax errors and that your file is properly named and placed in the submit folder. There should be no errors that prevent your program from successfully running.

   Proper style means that your variable and function names are descriptive and related to the information they store. You should add in comments for all non-trivial lines of code. There will be no penalty for over-commenting, so if a particular line of code needs to be explained–especially if it is doing some calculations or important flow control changes in your program–then put in comments on what that line of code is supposed to do. If you use helper functions, those are also named descriptively and include comments and docstrings. Docstrings should include a description of what the function does, a description of each parameter (if applicable) and a description of the return value/s (if applicable).

4. User Interface - 6 points (Manually Graded)
   As a general rule, a good user interface is easily understandable to users of this program who are not familiar with the assignment. This includes readability when printing information such as welcome and status messages (See the "User Interface Section"). The program should clearly indicate what input is allowed from users when asking for user input. Since we are running the program only in the console, make sure the user can easily see the difference between different turns / steps in the game. Using sufficient whitespace will help with overall clarity.

Submission

- Submit a single file called *water_tank.py*
- Check your submission by running the Pre-Submission Tests
- Fill out the statement of work header
- Remember to put the following lines of code at the end of your file, as the entry point of your program:

```
if __name__ == '__main__':
        main()
```