World of Games

Live.py

Create a new python file, call it Live.py.

welcome(name)

This function gets a person name as an input and returns a string in the following layout:

```
Hello <name> and welcome to the World of Games (WoG).
Here you can find many cool games to play.
```

load_game()

This function:

1. Will print out the following text:

```
Please choose a game to play:
    1. Memory Game - a sequence of numbers will appear for 1 second and you have to guess it back
    2. Guess Game - guess a number and see if you chose like the computer
```

2. Will get an input from the user about the game he chose – 1/2.

3. After receiving the game number from the user, the function will get the level of difficulty with the following text and also save to a variable:

```
Please choose game difficulty from 1 to 5:
```

4. Will start a new function of the corresponding game with the given difficulty.

The function will check the input of the chosen game (the input supposed to be a number between 1 to 2), also will check the input of level of difficulty (input should be a number between 1 to 5).

In case of an invalid choice, return the ERROR_MESSAGE (Utils.py).

For example: If a user will choose the first option in load_game() function with difficulty 3, it will call the play() function from the module MemoryGame with difficulty of 3.

In case the user won the game, the function will call the function called add_score() (in score.py module) to add the new score the user won to the score saved in the Scores.txt function.
In case the user lost, call load_game() again.

MainGame.py

Create a new python file, call it MainGame.py.

The purpose of this file is to call the functions from Live.py, it should look as follows:

```
from Live import load_game, welcome

print(welcome("Daniel"))
load_game()
```

GuessGame.py

Create a new python file, call it GuessGame.py.

The purpose of guess game is to start a new game, cast a random number between 1 to a variable called *difficulty*.

Methods

1. generate_number(difficulty) –
   A. Will get a number variable named *difficulty*
   B. Will return a random number between 1 to *difficulty*.

2. get_guess_from_user(difficulty) –
   A. Will get a number variable named *difficulty*
   B. Will ask the user to guess a number between 1 to *difficulty* and return the number the user guessed.

3. compare_results(difficulty, secret_number) –
   A. Will get 2 variables: number variable named *difficulty* number variable named *secret_number*
   B. Will compare the secret generated number to the one prompted by the get_guess_from_user.

4. play(difficulty) –
   A. Will get a number variable named *difficulty*
   B. Will call the functions above and play the game.
   C. Will return True / False if the user lost or won.

MemoryGame.py

Create a new python file, call it MemoryGame.py.

The purpose of memory game is to display an amount of random numbers to the users for 0.7 seconds and then prompt them from the user for the numbers that he remember.

If he was right with all the numbers the user will win otherwise he will lose.

Methods

1. generate_sequence(difficulty) –
   A. Will get a number variable named *difficulty*
   B. Will generate a list of random numbers between 1 to 101. The list length will be *difficulty*.
   C. The list will be shown for 0.7 seconds (using Utils.py module).

2. get_list_from_user(difficulty) –
   A. Will get a number variable named *difficulty*
   B. Will prompt the user the following message:
   After seeing the numbers enter the numbers you saw, each one separated with Enter.
   C. Will return a list of numbers prompted from the user. The list length will be in the size of *difficulty*.

3. is_list_equal(list_a, list_b) –
   A. Will get 2 variables named *list_a* and *list_b*
   B. The function will compare the two lists (list_a & list_b).
   C. The function will return True / False if the lists equal or not.

4. play(difficulty) -
   A. Will get a number variable named *difficulty*
   B. Will call the functions above and play the game.
   C. Will return True / False if the user lost or won (based on is_list_equal()).

- Example:
  - User choose difficulty = 3
  - 3 numbers between 1-101 will be generated to the user and will be shown in console for 0.7 seconds (using generate_sequence(difficulty)).
  - User will enter the guessed numbers with an enter between each guess (using get_list_from_user(difficulty)).
  - The list of the generated numbers and the list of the guessed numbers will be compared and will return true / false respectively

Utils.py

Create a new python file, call it Utils.py.

This file will contain general information and operations we need for our game.

1. SCORES_FILE_NAME - A string representing a file name. By default “Scores.txt”
2. ERROR_MESSAGE: “Something went wrong..”
3. screen_cleaner - A function to clear the screen (useful when playing memory game or before a new game starts).
   ** The following code can be used:
   ```
   os.system('cls' if os.name == 'nt' else 'clear')
   ```

Score.py

Create a new python file, call it Score.py.

A file which will manage the scores file.
The scores file at this point will consist of only a number.
That number is the accumulation of the winnings of the user.

Amount of points for winning a game is = 1 point per difficulty level (difficulty 3 = 3 points).

Each time the user is winning a game, the points he won will be added to his current amount of point saved in a file.

Methods

add_score() –

A. The function’s input is a variable called points.
B. The function will try to read the current score in the scores file, if it fails it will create a new one and will use it to save the current score.
C. The function will print the user current score.

MainScores.py

Create a new python file, call it MainScore.py.

This file’s purpose is to retrieve the current user score from the scores.txt file over HTTP with HTML. This will be done by using python’s flask library.

Methods

1. score_server() –
   This function will serve the score. It will read the score from the scores file and will return an HTML that will be as follows:

```
<html>
        <head>
            <title>Scores Game</title>
        </head>
<body>
            <h1>The score is <div id="score">{SCORE}</div></h1>
</body>
 </html>
```

   If the function will have a problem showing the result of reading the error it will return the ERROR_MESSAGE (from Utils.py) and the exception message In the below format:

```
<html>
        <head>
            <title>Scores Game</title>
        </head>
        <body>
        <body>
            <h1><div id="score" style="color:red">{ERROR_MESSAGE}</div></h1>
</body>
</html>
```

** Use the attached flask sample to start (MainScores.py)

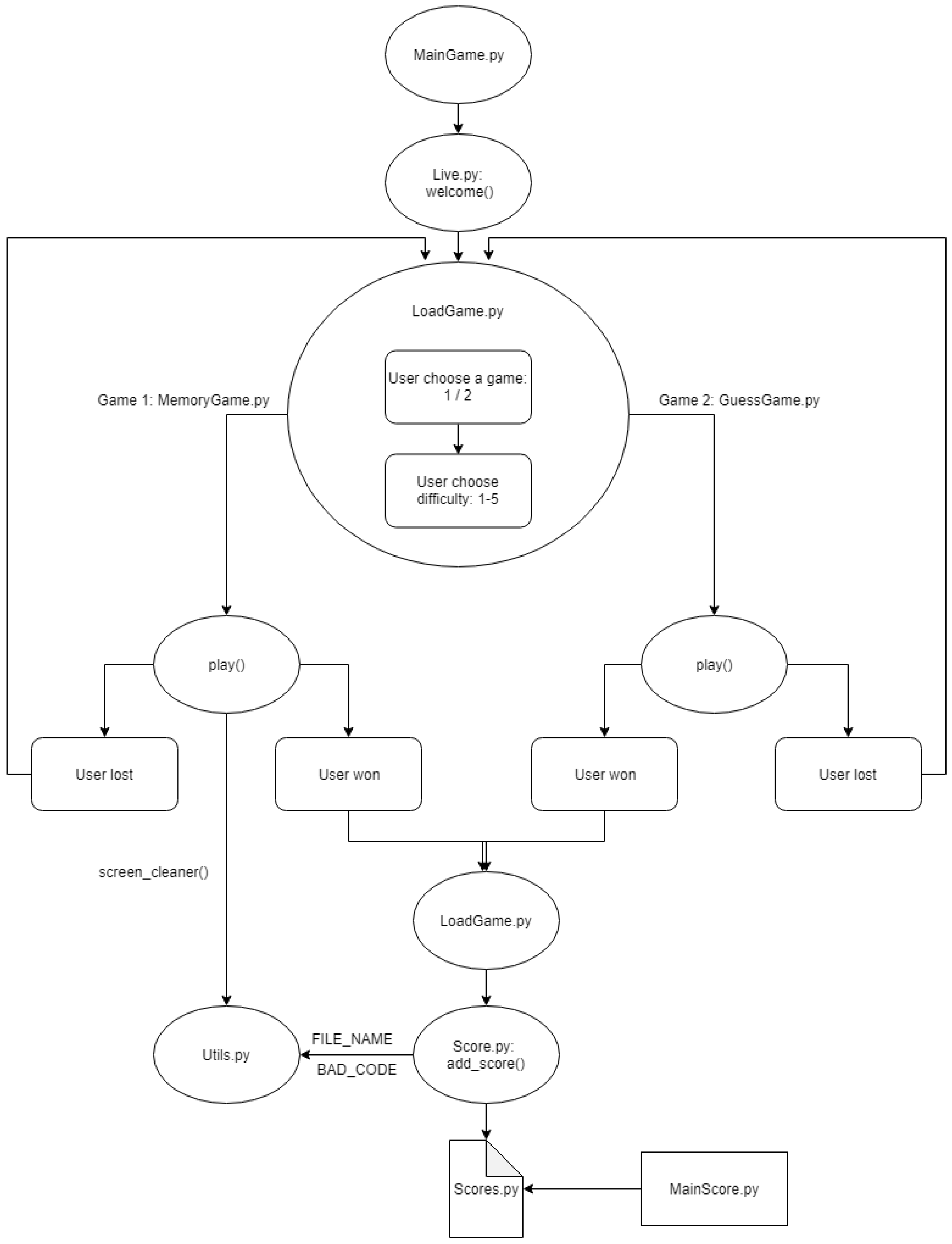
What to send me?

A compressed zip file containing the following:

1. MainGame.py
2. Live.py
3. GuessGame.py
4. MemoryGame.py
5. Utils.py
6. Score.py
7. MainScore.py
8. Scores.txt

General guidelines:

1. Each method will have documentation.
2. All variables need to have valuable names
3. Use Python naming conventions.
4. Protect code blocks where necessary.


MainGame.py
Live.py: welcome()
LoadGame.py
User choose a game: 1 / 2
User choose difficulty: 1-5
Game 1: MemoryGame.py
Game 2: GuessGame.py
play()
play()
User lost
User won
User won
User lost
screen_cleaner()
LoadGame.py
Utils.py
FILE_NAME
BAD_CODE
Score.py: add_score()
Scores.py
MainScore.py