Documentation

LIBRARIES

It is required for this game to use **time**, **csv** and **random** libraries.

Before we go into the game logic and game mechanics, we will need to set up the logistical functions.

CLASS

First we start off by using classes, we create a **Player** class, we use instantiation to create instances of the class.
Players have

1. A name
2. initialize stage 1 and 2 points to 0
3. stage 3 points to 0
4. stage 3 lives to 6

Class Methods

1. **add_points(self, amt, stage)** method that will add points to the stages respectively.
2. **loselife(self)** method, which upon calling will deduct the number of lives in the player's instance.

Next, we create a function **create_players(num)** to help us set the number of players and this will create for us the instances of the players using a for loop.

Overall Game Logic

Here we have 3 main variables,
lis_of_players which is an empty list
player_current_index = 0
stage_level = 1

Next, we prompt the user for how many players using the input() and then we use our **create_players(num)** function to set the amount of instances respectively. We then append the result of this function to the **lis_of_players**.

Next, we have 3 main game functions
start_stage1(lis_of_players)
start_stage2(lis_of_players)
start_stage3(lis_of_players).
They each take in the **lis_of_players** and they will start each stage of the game respectively.

Lastly, we have the **determine_winning_player(lis_of_players)** that determines the one winning player from the lis_of_players who are left.

STAGE 1 LOGIC

In this next section, we will solely focus on stage 1 functions only.
Stage 1 has 3 rounds of questions each with a different topic - Geography, History and HASS.

Our first function is **start_stage1(list_of_players)**. It takes in the list of players from the global variable.

1. It uses a while loop that is true at the start
2. It prints 4 lines of introduction to the player
3. It then enables each round of the game to begin through the function **Topic_game(topic)** which occurs 3 times.
    a. Topic_game('Geography')
    b. Topic_game('History')
    c. Topic_game('HASS')
4. We set the start_loop variable to False
5. Continue enables us to break the loop

Our main function in stage 1 is the **Topic_game(topic)** function. It takes in a string as its input.

1. We have 3 if statements to check if the topic is either Geography, History or HASS.

2. Inside the if statement, we have 2 variables followed by 2 while loops.
    a. Variable **questions** = 6. This is the number of questions available per round.
    b. Variable **stage_1_question_list** is the result of a function **convert_to_list(topic)**

3. Next, we have 2 while loops.
    a. The first while loop has a conditional statement that checks that the variable questions is greater than 0.
        i. If greater than 0, it contains one variable **players_who_havent_answered**.

            1. This variable players_who_havent_answered is the result of using a shallow copy of the **lis_of_players** (global variable). We will need the new variable in our second while loop and also to enable the rotation of players within our questions.

    b. The second while loop checks for the length of the list **players_who_havent_answered** is greater than 0.

        i. If its greater than 0, we have a tuple variable named qn, dic.
            1. This tuple is the result of a new function **turn_stage_1_wheel_to_pick_question(topic, stage_1_question_list)**
        ii. Next we have a string variable, we print it to display the question and the options available.

        iii. Next we use the **random.choice(players_who_havent_answered)** to choose a random player and assign the player to the variable player.
        iv. We then use a input to get the players answer, which we assign to the variable answer.

            1. Here we use 1 if else statement which has nested if else statements within each statement.

a. In the 1st if statement, we check if the answer is equal to the ‘Correct Option key-value pair in our dictionary’.
b. If correct, we remove the player from the players_who_havent_answered so that this player is unable to be chosen for the next question.

    i. He will only be able to answer once all the players have been given a chance to answer
    ii. The next player gets the wrong answer, the question will be opened up to him again.

c. Our next line, we try to add points to the player for getting the correct answer, we use a for loop

    i. We loop through the global variable **lis_of_players** and we look for a match with the local variable player using an if statement.
    ii. When we get a match, we call the **add_points** method for the **Player** class and we print the new amount of points.

        1. We use another if statement within this if statement to check if the player has more than 2000 points.
        2. If they have more than 2000 points, we set the local variable question to 0 and we clear all the players from **players_who_havent_answered** list. We use a continue statement to return to the while loop and check the condition for players_who_havent_answered and it breaks the entire loop.

d. After this for loop is run, we remove the same question from the stage_1_question_list with the remove method. We use a continue method to start the next question.

2. The else statement here also has a for loop and nested if statement. The difference here is the scenario where the player answers wrongly, the question opens up to the rest of the players

    a. Here we have a new variable called **lis_no_wrong** which copies the **lis_of_players** (global variable).
    b. We use the remove method to remove the player who answered wrongly from the **lis_no_wrong** variable and we also remove him from the **players_who_havent_answered** variable
    c. We then use a for loop to loop through the **lis_no_wrong** players.
        i. Here we have an if else statement
            1. First scenario the person gets correct, we

follow the same logic as the above
2. Second scenario the person gets wrong, we add a continue statement to continue the for loop.

d. Outside the loop, we subtract 1 from the local variable questions and we remove the question from the variable **stage_1_question_list**. We then use continue to continue the loop of questions.

4. Once the round ends, we print a statement to indicate the round has ended.

We have 2 other functions involved in the **Topic_game()** function.

1. **turn_stage_1_wheel_to_pick_question(GK, lis_of_questions)** function
    a. The first function has 3 if else statements, it checks the topic, then it selects a random question using the **random.choice(lis_of_questions)** and it access the value of the key ‘Question’ and returns a tuple of the question and the dictionary

2. **convert_to_lis(topic)** function
    a. The second function takes in a topic parameter which is a string. It initializes a questions_list variable to an empty list.
        i. It opens the file **‘stage1.csv’** and iterates through it using a for loop.
            1. Each iterator variable is checked, if its the same as the topic, we append this iterator variable (which is a list containing a dictionary) into the variable questions_list which is an empty list.
        ii. After the loop, the questions_list is returned containing a list of dictionaries, each dictionary containing the question, options and correct answer.

STAGE 2 FUNCTIONS

Starting Stage 2, it starts with **start_stage2()** function.
start_stage2(lis_of_players) function takes in a list.

1. It has 5 variables.
    a. Variable **letters** that contains all the letters from a-z
    b. Variable **vowels** that contains vowels
    c. Variable **vowel_cost** which is 150
    d. Variable **Letter_price** which is 100
    e. Variable **reward_price_per_char** which is 200

2. Next, we call a new function **csvFiletoListDic(‘stage2.csv’)** and the end result is a list is assigned to the variable **list_of_words_to_guess**.

3. Next, **descending_player_index_sort(lis_of_players)** will sort the players in descending order starting with the player with the highest number of points. This will be the order for this stage of the game, highest points starts first.

4. Next, a local variable **attempt_Num** is assigned 0 which is used for cycling through the players

5. Next, a for loop is used to loop through each question in our **list_of_words_to_guess**

list.

a. Within the loop, list(letters) creates a list of letters from a-z and is assigned to variable **letters_remain**.

b. Variable **letters_guessed_correctly** is assigned to an empty list.

c. The iterator variable question is a dictionary, we access **question['Question']** and lowercase the letters as our **letters_remain** are all in lowercase and we assign this to the variable **question_word**

d. Next, we pass **question_word** as a parameter into the **obscurePhrase()** function and its result which is a string (eg '_ _ _ _ _ _') is assigned to the variable **covered_word**.

e. Next, we use a while statement to check if **covered_word** not equal to **question_word** this is to repeat until the word is solved.

    i. Inside the while loop, **obscurePhrase(question_word, letters_guessed_correctly, "_ ")** function is called and its result is assigned to **display_covered_word**. Thi difference between the displayed version is that for each blank, there an additional space after each '_' for better readability.

    ii. Next, we print the word to guess for the user to see.
    iii. Next, we have a system to select players by a fixed order.
        1. We use a **player_index** variable which is **attempt_Num** % **len(lis_of_players)** which will help us select the player index. Example for the first player, 0 % 5 = 0.
        2. We use list indexing to select the player.

    iv. Print a statement to remind the player of the number of lives and points they have.
    v. Set a variable **letter** which contains an empty string
    vi. Here a while loop is used to get the players guessed letter.

        1. Input gets the letter the player guesses and assigns it to a variable **letter**.

        2. We use an if statement to check if the letter typed is more than 1 letter or the letter has already been used by checking the list **letters_remain**.

        3. Otherwise, **letters_remain list** will remove the letter guessed.

        4. Next, we have an if else statement, if will run if the letter is a vowel, we get the **vowel_cost** variable and deduct 150 points else it will deduct 100 points from the players **stage_1_2_points**

        5. Next, we have another if else statement to check if the letter is inside the **question_word**.

a. if it is not, we deduct a life by calling the class method **loselife()**.

b. Else, we add 200 points to the player per character guessed correctly. If there are 3 of the letters the player guessed inside, we will reward the player for each of that same letter guessed.

   i. Next, we use the append method to add the guessed letter to our list **letters_guessed_correctly**.

   ii. We update our **covered_word** by calling **obscurePhrase** function and passing in the updated **letters_guess_correctly** parameter.

c. Outside the else statement, we call the **eliminate_check(player, lis_of_players)** function to remove the player from this round until all players have been given a chance to guess.

d. Next, we add 1 to the **attempt_Num** variable to cycle through the players.

While running the code in **start_stage2()**, we will call 4 other functions.

1. **csvFiletoListDic(filename)**
   a. An empty list is assigned to our variable list_of_words_to_guess
   b. This function takes in a file as its input
   c. Using with open, we open the file and use a csv reader to create a csv.DictReader object.
   d. Using a for loop, iterating over the object, we append each line (which is a dictionary) into our list_of_words_to_guess list.
   e. Outside the loop, we return the list_of_words_to_guess (which is a list of dictionaries, each dictionary corresponds to one row on the csv file)

2. **descending_player_index_sort(list_of_players)**
   a. This function is used to open sort the players in descending order based on their points
   b. We take in the global variable lis_of_players which is a list as its input.
   c. By checking each player's points in **list_of_players**, an insertion sort is run to sort the **list_of_players** objects in descending order of their points.

3. **eliminate_check(player,lis_of_players)**
   a. This function is
      i. Used to show the players lives and points
      ii. used to remove a player if they have less than or equal to 0 points or if their lives equal to 0.
   b. Does this by using an if statement
   c. Uses the remove() method to remove the player from the global variable lis_of_players

4. **filter_top_players(lis_of_players,number_of_top_players)**
   a. The aim is to get 2 top players that move onto stage 3.
   b. By calling, descending_player_index_sort() we get the list of players with highest

number of points as the first.

   c. Using a while loop, if the length of the global variable lis_of_players is greater than 2, we will use the method pop to remove the players. This will leave us with only 2 players left in our lis_of_players.

STAGE 3 FUNCTIONS

At the start of stage 3,our global variable lis_of_players only has 2 players inside. Here we will focus on the functions used in stage 3.

Starting us off is the function **start_stage3()**

1. We set the initial variable start_level to 3 for stage 3.

2. Calling the function **csvFiletoListDic()** which will return us a list of dictionaries and assign it to variable **ls_questions**.

3. Using 2 for loops
   a. 1st for loop uses an enumerate function which adds an index to every question in the ls_questions list. Inside the for loop,

      i. 2 variables **best_player** and **best_time** is set to None datatype.
      ii. 2nd for loop just loops through the lis_of_players to rotate the players.
      iii. Using a variable correct, we initialize it to a boolean value False.
      iv. After prompting the question, **.time()** function immediately starts to record how long it takes for the player to select the correct option.
      v. Variable answer is an empty string.

         1. While loop follows the same logic as stage 2 while loop to get the answer from the player. When player answers, we end the time() function and save the time.

         2. Usage of 3 if else statements are used.
            a. 1st is to check if input is valid (ABCD)
            b. 2nd is to check if input is correct answer by cross checking with the **ls_questions['ans'**]. It also checks if its better than the previous players time and if it is, we set the player to best player

         3. Outside this while loop, we check if a **best_player** exists, if it does, we add points.
         4. We then display the updated scores

Lastly, we call **determine_winning_player()** which takes in a list of players as its input and determines the player with the highest score via the max() function.