CANDIDATE'S FULL NAME: Daniel Mula Tarancón

PROJECT STATEMENT

The hangman game is a word guessing game where the player is given a word and has to guess the letters make up the work.

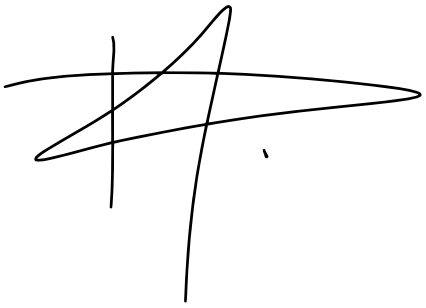
The player is given a certain number of tries (no more then 6 wrong guesses are allowed) to guess the correct letters before the game is over.

 # Output

You have 6 tries left.
Used letters:
Word: _ _ _ _
Guess a letter: a

You have 6 tries left.
Used letters: a
Word: _ a _ a
Guess a letter: j

You have 6 tries left.
Used letters:
Word: j a _ a
Guess a letter: v
You guessed the word java!



INDEX

1. How did I approach this project? [3]

2. Python libraries used along this project [4]

3. Getting into details [5]

    3.1. Files Structure

    3.1.1. app.py

    3.1.2. __init__.py

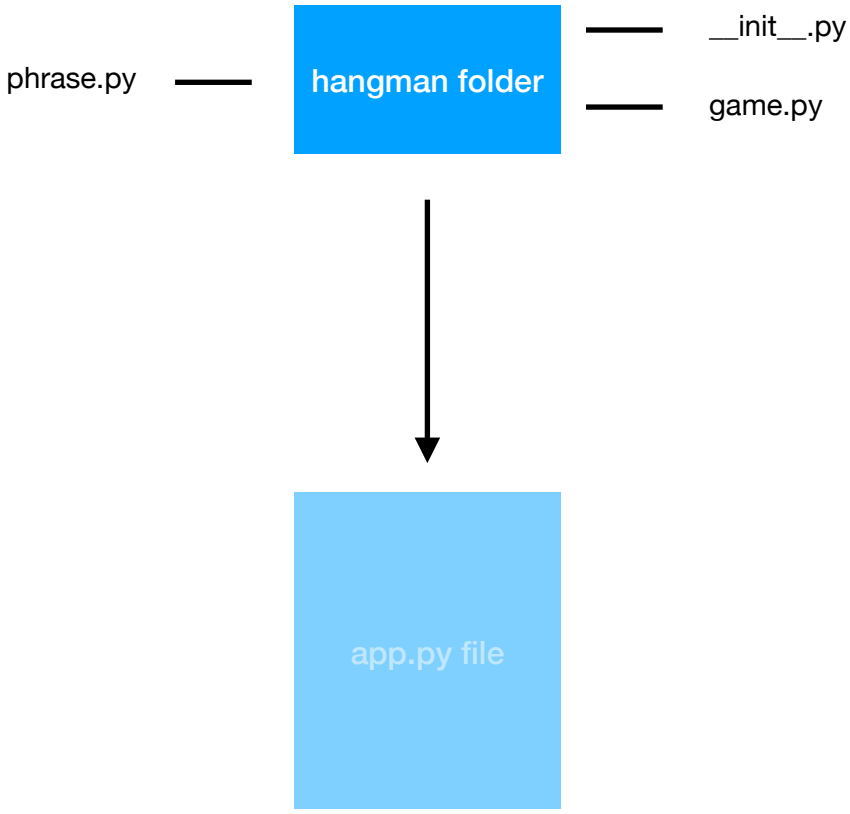
    3.1.3. game.py

    3.1.4. phrase.py

4. Conclusions [16]

1. How did I approach this project?

In order to having a neat project I decided to create several files.



And as you will see I decided to work with Classes so my code is more compact.

2. Python libraries used along this project

So the only Python library I needed was the ‘string’ library. So I could use the ascii_lowercase method.

The string.ascii_lowercase is a constant string that all the lowercase letters of the English alphabet.

So this is absolutely necessary in order to verifying that the input is correct.

On the other hand I created as I anticipated before a python package called hangman.

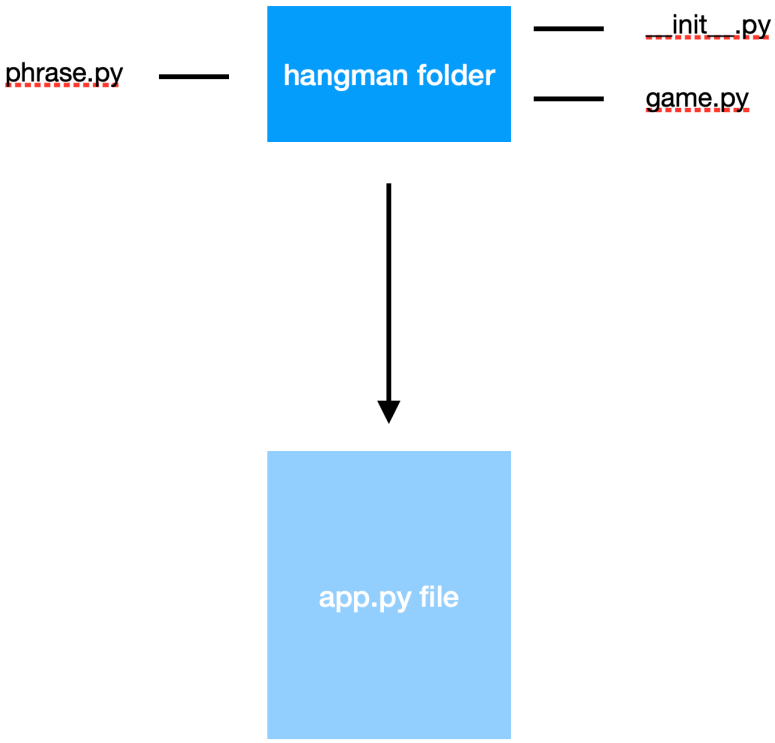
And I did that by creating a folder with the __init__.py file inside. The __init__.py file is a special file that is used to mark a directory as a Python package. When you import a package or a module, Python looks for the __init__.py file to determine if it is a package or a module.

3. Getting into details

Along the chapter we will get into details explaining the main features of each part of the program.

3.1. Files Structure

Let's refresh our memories, this is the files structure that I've used:



3.1.1. app.py

This is the file that runs the program.

It contains only a few lines of code. And it execution is controlled by a ‘dunder main’.

```
from hangman.game import Game

if __name__ == "__main__":

    game = Game()

    game.create_phrases()

    game.get_random_phrase()

    game.start()
```

Firstly I import the class Game which is inside the package hangman and inside the module game.

This class (the Game) class defines how my game will behave.

Inside the ‘dunder main’ I create an instance of that class called game.

Then I create the phrases of the game so I can play. I do so, by calling the method ‘create_phrases( )’.

After that, I need a phrase so calling the method ‘get_random_phrase()’ I get one random phrase so I can start playing.

Now I have everything I need to start playing the game so I just call the method ‘start()’.

3.1.2. __init__.py

This file is empty. It's only purpose is to create a python package.

3.1.3. game.py

This file consists of only a class, the Game class. I will explain the methods and attributes that compone that class in this section.

This class is only composed by methods. Those methods are the following:

A) __init__(self)

```
def __init__(self):

    self.missed = 0
    self.phrases = []
    self.active_phrase = None
    self.guesses = []
```

This is the class constructor. The method called when the class is executed so it can build the class structure.

So I've defined the self.missed (monitors the number of wrong attempts), self.phrases (a list that will contain the sentences I've created for the game), self.active_phrase (this instance attribute will store the chosen sentence. The one the user must guess) and eventually self.guesses (it will store a list that will contain the letters used).

B) create_phrases(self)

```
def create_phrases(self):

    a = phrase.Phrase('May the force be with you')
    b = phrase.Phrase('Darth is back')
    c = phrase.Phrase('It is the Matrix')
    d = phrase.Phrase('Hacking Time')
    e = phrase.Phrase('If you read this you are pretty cool')
    self.phrases = [a,b,c,d,e]
```

Inside this method I create the phrases that will be available for the game.

C) get_random_phrase(self)

```
def get_random_phrase(self):

    self.active_phrase = random.choice(self.phrases)

    return self.active_phrase
```

The purpose of this method is to choose the phrase that the user must guess.

D) welcome(self)

```
def welcome(self):
    print("""   *** ---------------------------------------------- ***
WELCOME LADIES AND GENTLEMEN TO HANGMAN!!!
--------------------------------------------------""")
```

This method is just used to welcome the user.

E) start(self)

```
    def start(self):

        self.welcome()
        while self.missed < 6 and self.active_phrase.check_complete(self.guesses) == False:
            print("\nYou have {} tries left\n".format(6-self.missed))
            self.active_phrase.display(self.guesses)
            user_guess = self.get_guess()
            if user_guess not in self.guesses:
                self.guesses.append(user_guess)
            else:
                print('''\nYou have already entered that letter, try another one.
Easy, I will not consider it a mistake. Remember the used letters are {}'''.format(self.guesses))
                continue
            self.active_phrase.check_guess(user_guess)
            if self.active_phrase.check_guess(user_guess):
                print("\nYAY!!!")
            if not self.active_phrase.check_guess(user_guess):
                self.missed += 1
            print('\nUsed letters: {}'.format(self.guesses))
        self.game_over()
```

This is the method that starts the game. It starts welcoming the user by calling the self.welcome method. Then we display the tries left, the current status of the active phrase, I catch the guess, I consider the user trying to guess the same letter twice. The I verify if the letter guessed is correct. Eventually as we can see the self.game_over() function is executed when the ‘while loop’ is over (either because the user misses 6 times or because the user guess the whole phrase).

F) game_over(self)

```
def game_over(self):

    if self.missed >= 6:
        print("\nI'm sorry you've missed too many letters\n")
        self.active_phrase.display(self.guesses)
    else:
        print('\nWell done!!! You got it!!, you guessed the correct phrase: \n')
        self.active_phrase.display(self.guesses)

    while ValueError:
        try:

            play_again = input('would you like to play again[y/n]? ').lower()

            if play_again != 'y' and play_again != 'n':
                raise ValueError("Please enter 'y' (yes) or 'n' (no)")

        except ValueError as err:
            print ('{}'.format(err))

        else:
            if play_again == 'y':
                self.restart_game()
            else:
                print("""Alright as you wish...
            GAME OVER""")
                break
```

As we can see if we miss 6 times, the app will display a message and the game will be over. Or if we guess the whole phrase.

On the other hand, inside this method we ask the user if he/she wants to play again. And we handle this with a while loop and a ‘try - except - else’ block. Inside that block I call the restart_game method that starts the game again.

G) get_guess(self)

```
def get_guess(self):

    while ValueError:

        try:
            answer = input('Guess a letter: ').lower()


            if (set(answer) <= set(string.ascii_lowercase)) == False:
                raise ValueError('You must enter only a letter, try again please')
            elif len(list(answer)) > 1:
                raise ValueError('You must enter only a letter, try again please')

        except ValueError as err:

            print('{}'.format(err))

        else:

            return answer
```

This methods ask the user for a letter and makes sure that the input is given using the correct data type and returns the answer.

H) restart_game(self)

```
def restart_game(self):
    game = Game()
    game.create_phrases()
    game.get_random_phrase()
    game.start()
```

This method restart the game by creating an instance of the class Game and calling the proper methods.

3.1.4. phrase.py

Inside this file I've created the class Phrase that manages the phrase current status. This class is composed by the following methods:

A) __init__(self, phrase)

```
def __init__(self, phrase):

    self.phrase = phrase.lower()
```

When we create the constructor that will build the clase once we call it. We define 'phrase' instance attribute that need to be pass to the class. This is the phrase that we randomly have chosen. And we indicate that we want to work with lowercase.

B) display(self, guesses)

```
def display(self, guesses):

    for letter in self.phrase:
        if letter in guesses:
            print(f'{letter}', end="")
        elif letter == ' ':
            print(" ", end="")
        else:
            print("_", end=" ")

    print('\n')
```

The display method displays the current status of our phrase.

C) check_guess(self, guess)

```
def check_guess(self, guess):

    if guess in self.phrase:
        return True
    else:
        return False
```

Using this method I verify if the user has or has not guess the letter.

D) check_complete(self, guesses)

```
def check_complete(self, guesses):

    for letter in self.phrase:
        if letter not in guesses:
            return False
        else:
            guesses.append(' ')
            if set(guesses) >= set(self.phrase):
                return True
            else:
                return False
```

Here I verify if the user has guessed the whole phrase. To do so I decided to use the data type ‘set’ so I can compare the size of the guesses and the phrase, since sets are suitable when it doesn’t matter if an element is duplicated or not. Here it only matters the element itself, it doesn’t matter how many times it appears.

4. Conclusions

I think the project meets all the requirements and looks great. Investing more time in it might allow me to improve it but I really think it is good enough.