ROCK, PAPER, SCISSORS CONSOLE GAME

Release v1.0.0

Buğra Can Aşcı

November 13, 2020

Description

This is a console application for playing Rock, Paper, Scissors solo against the computer, developed in Python by Buğra Can Aşcı.

Prerequisites

- List of all the dependancies:
    1. Python (Version: 3.8.3)
    2. “time” module (Python built-in module)
    3. “abc” module (Python built-in module)
    4. “system” module (Python built-in module)
    5. “random” module (Python built-in module)

Installation

- Installing Python and the modules necessary:
    1. Download Python 3.8.3 from the link and add it to your environment variables: https://www.python.org/downloads/release/python-383/ (If you prefer using a virtual environment, do not add this Python version to your system path).
    2. All the modules used in this project are built-in with Python, further dependancy installation is not necessary.

Executing the Program

Option 1:

Inside the project folder, go to the directory named ‘dist’ and run the file “rock_paper_scissors.exe” to start the game. (for Windows OS)

Option 2:

Using python:

- Run the system console:
    - If you are using a virtual environment:
        1. Activate the virtual environment you set up, containing Python 3.8.3.
        2. Go to the path that the project folder is stored in, go inside the project directory and then type:

            “python rock_paper_scissors.py”
    - If you added Python 3.8.3 to your system path:
        - Go to the path that the project folder is stored in, go inside the project directory and then type:

            “python rock_paper_scissors.py”

Contributing

https://github.com/canasci/rock_paper_scissors_console_game

Source Code:

```
import random
import time
import sys
from abc import ABC, abstractmethod


class GameItems:
    """ This class contains the context dictionary of
        available game moves and their status against their
        counter-moves """

    context_dict = {'rock': {'beats': "scissors", 'ties': "rock", 'loses_against': "paper"},
                    'paper': {'beats': "rock", 'ties': "paper", 'loses_against': "scissors"},
                    'scissors': {'beats': "paper", 'ties': "scissors", 'loses_against': "rock"}
                    }

    def __init__(self, key: str):
        self.name = key
        self.beats = self.context_dict[key]['beats']
        self.ties = self.context_dict[key]['ties']
        self.loses_against = self.context_dict[key]['loses_against']

    def __str__(self) -> str:
        return self.name.upper()
```

```
class Rock(GameItems):
    """ This class extends GameItems class,
        expecting the 'rock' key for initialization"""

    def __init__(self, choice: str):
        super().__init__(choice)


class Paper(GameItems):
    """ This class extends GameItems class,
        expecting the 'paper' key for initialization """

    def __init__(self, choice: str):
        super().__init__(choice)


class Scissors(GameItems):
    """ This class extends GameItems class,
        expecting the 'scissors' key for initialization """

    def __init__(self, choice: str):
        super().__init__(choice)
```

```
class Player(ABC):
    """ This is an abstract class for
        other player classes to extend """

    def __init__(self, name='', choice=''):
        self.name = name
        self.choice = choice

    @abstractmethod
    def action(self):
        """ abstract method to be implemented in the corresponding sub/child-classes """
        pass

class User(Player):
    """ This class extends Player abstract class,
        and implements the constructor and 'action' method
        to be used to represent program user actions """

    def __init__(self):
        super().__init__(name='', choice='')

    def action(self) -> object:
        """ action method is used to represent the choice the
            user will make amongst available game items in the context dictionary provided in GameItems
class,
            and returns the object representing the user's choice """

        if self.choice == 'rock':
            return Rock(self.choice)
        elif self.choice == 'paper':
            return Paper(self.choice)
        elif self.choice == 'scissors':
            return Scissors(self.choice)
```

```
class Computer(Player):
    """ This class extends Player abstract class,
        and implements the constructor and 'action' method
        to be used to represent the randomized computer actions"""

    def __init__(self):
        super().__init__(name='', choice='')

    def action(self) -> object:
        """ action method is used to represent the randomized choice
            the computer will make amongst available game items in the context dictionary provided in
GameItems class,
            and returns the object representing the computer's choice """

        self.choice = random.randint(1, 3)
        if self.choice == 1:
            self.choice = 'rock'
            return Rock(self.choice)
        elif self.choice == 2:
            self.choice = 'paper'
            return Paper(self.choice)
        else:
            self.choice = 'scissors'
            return Scissors(self.choice)
```

```
class RpsGame:
    """ This is the class representing the game attributes and methods """

    context_dict = {'rock': {'beats': "scissors"},
                    'paper': {'beats': "rock"},
                    'scissors': {'beats': "paper"}
                    }

    def __init__(self):
        self.player1 = User()      # User object get instantiated as player1 attribute of the game class
        self.player2 = Computer()  # Computer object get instantiated as player2 attribute of the game
        self.game_count = 1        # Game count is stored
        self.player1_wins = 0      # Amount of times the user wins
        self.player2_wins = 0      # Amount of times the computer wins

    def start_game(self):
        """ start_game method prompts the user their name and move, and sets related objects' attributes
to be used
            in the play method of the game """

        if self.game_count == 1:
            print("ROCK, PAPER, SCISSORS CONSOLE GAME\n")
            self.rules()
            print("\nGame {} Starting...\n ".format(self.game_count))
            self.player1.name = input("Please enter your name: ")
        else:
            print("\nGame {} Starting...\n".format(self.game_count))

        self.player1.choice = str.lower(input("Please choose your move,"
                                              " {} (Rock, Paper, Scissors): ".format(self.player1.name)))
        while self.player1.choice not in ['rock', 'paper', 'scissors']:
            self.player1.choice = input("Please enter a valid move, "
                                        "(Rock, Paper, or Scissors): ".format(self.player1.name)).lower()
```

```
    def rules(self):
        rock_beats = self.context_dict['rock']['beats']
        paper_beats = self.context_dict['paper']['beats']
        scissors_beats = self.context_dict['scissors']['beats']

        print("Rock beats {0}\nPaper beats {1}\nScissors beat {2}\n".format(rock_beats, paper_beats,
scissors_beats))

    def play(self) -> dict:
        """ play method calls the action method of the player objects
            instantiated and returns a dictionary containing the players'
            decisions as game item objects """

        return {'User': self.player1.action(), 'Computer': self.player2.action()}

    def game_result(self) -> str:
        """ game_result method calls the play method of the game class and stores the dictionary returned
by the play
            call in a variable called moves. 'moves' variable is used to get and print out the moves made by
players
            in order to determine and return the result of the game """

        moves = self.play()
        print("\nUser played {0}, \nComputer played {1}".format(moves['User'], moves['Computer']))

        if moves["User"].name in moves["Computer"].loses_against:
            self.player1_wins += 1
            return "\nUser wins! {0} beats {1}".format(self.player1.choice.upper(),
self.player2.choice.upper())
        elif moves["User"].name in moves["Computer"].beats:
            self.player2_wins += 1
            return "\nComputer wins! {1} beats {0} \nEZ PZ!".format(self.player1.choice.upper(),
```

```
self.player2.choice.upper())
        elif moves["User"].name == moves["Computer"].name:
            return "\nGame tied! NICE TRY!"

    def play_again(self):
        """ play_again method asks the users
            calls the action method of the player objects
            instantiated and returns a dictionary containing the players'
            decisions as game item objects """

        user_decision = input("\nWould you like to play again? [y/n]:\n").lower()

        while user_decision not in ['yes', 'no', 'y', 'n']:
            user_decision = input("Please enter 'y' for 'Yes', 'n' for 'No'!\n")

        if user_decision == 'n':
            print("\nUser won {0} of the {1} game/s played!".format(self.player1_wins, self.game_count))
            print("Thank you for playin', NOOB!\n")
            exit_key = "exit"
            while exit_key != "":
                exit_key = input("Please press 'Enter' to exit the game!")

            print("\nProgram terminating...")
            time.sleep(3)   # program sleeps 3 seconds before program exits!
            sys.exit(0)     # program exits

        elif user_decision == 'y':
            self.game_count += 1
            self.start_game()
            self.play()
            print(self.game_result())
            self.play_again()
```

```
if __name__ == '__main__':
    game = RpsGame()          # instantiates a game object when the program executes
    game.start_game()         # game starts, using the start_game method of the game object
    print(game.game_result())  # game result is evaluated and printed to the console using the game method of the game object
    game.play_again()         # the user is asked if they want to play again, and the game continues accordingly
```