Rock-Paper-Scissors Terminal App

Overview

- Python script for a game of Rock-Paper-Scissors against a computer opponent.
- The script is run entirely in the command-line.
- The script uses several modules, including random, csv, coloured, art, and emoji
- The script defines several function
  - get_player_choice() for getting the player's choice
  - get_computer_choice() for generating the computer's choice
  - play_round() for determining the winner of a round
  - save_scores() for saving scores to a csv file.
  - print_scores() for printing the scores from the csv file.
  - game() for running the game.

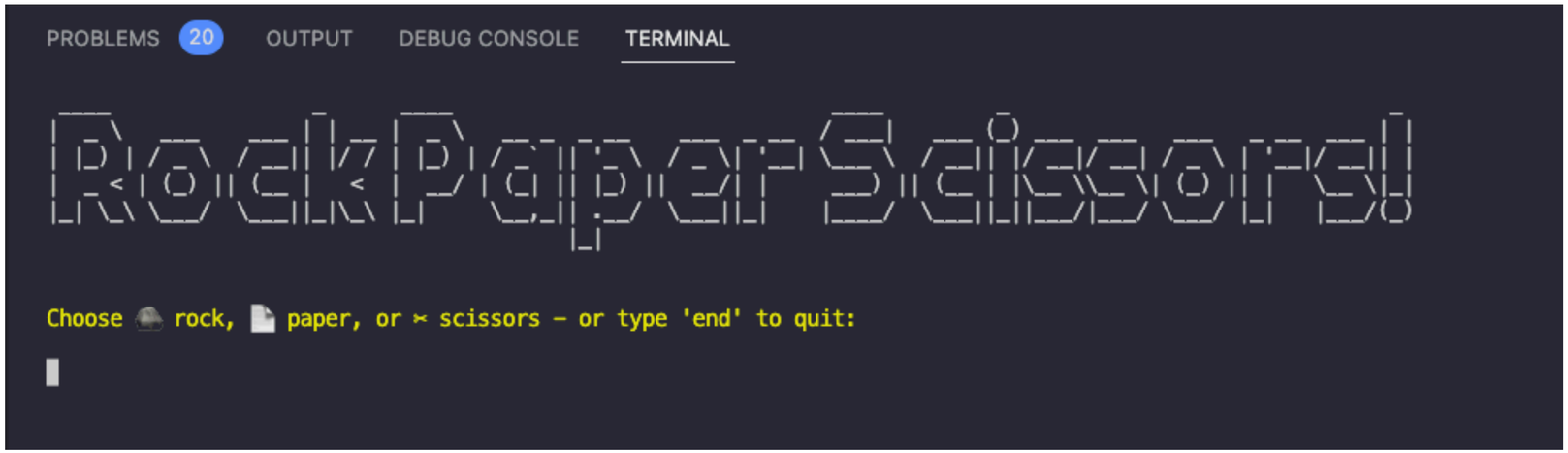
Overview continued…

- The game() functions contains the main game loop.
- In the game loop, the player chooses rock, paper, or scissors, and the computers generates a choice randomly.
- The game loop continues until a player reaches a score of 5.
- At the end of the game, the winner is announced and the script exits.

Walkthrough

Welcome screen

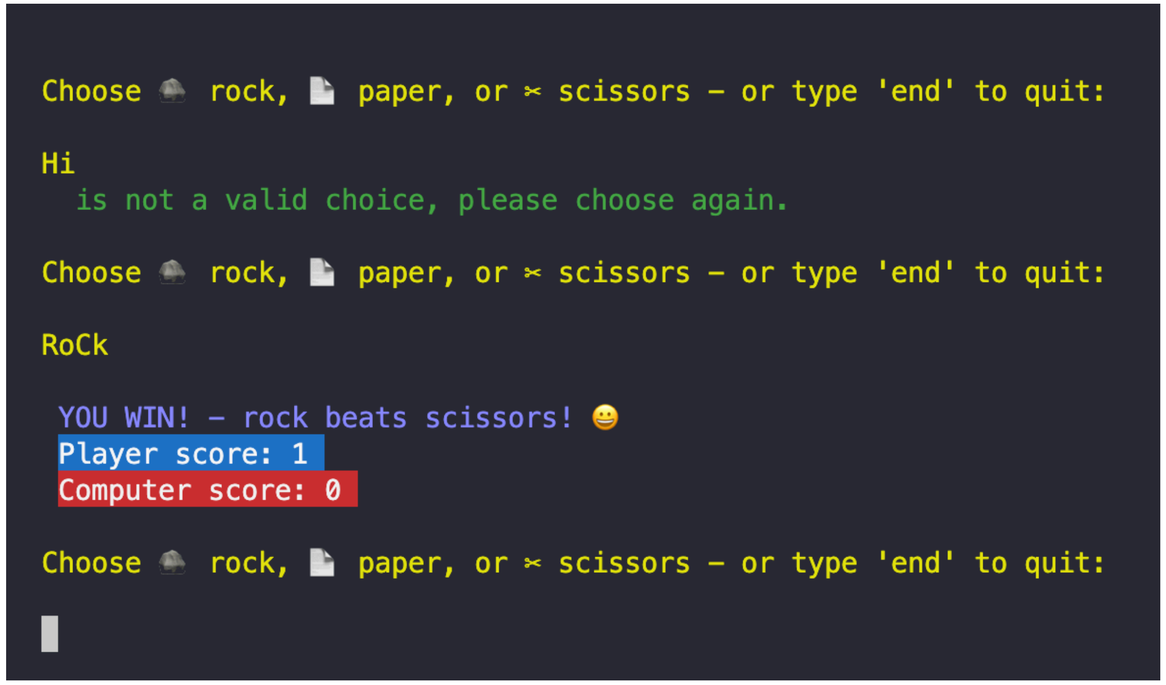
- Upon using the ./run.sh command, this is the screen which welcomes you.
- The Art module is used to print out Rock Paper Scissors! ASCII art to the terminal screen.
- The game commences and the terminal ask's you to type out either, 'Rock', 'paper', or 'scissors', or 'end' to quit the program.
- Emoji and colored module is used throughout for font colours and emoji's in our output.



Walkthrough

Playing a round.

- As you can seen, typing a non-valid input such as ‘Hi’ will alert the user that it’s an invalid response and then re-prompt the user for a choice.
- After typing ‘Rock’ we can see we have won the first round, a message is printed to the screen, advising us we have won the round and the player score is updated by +1 point and both scores are printed to the screen.
- The input is not case-sensitive.

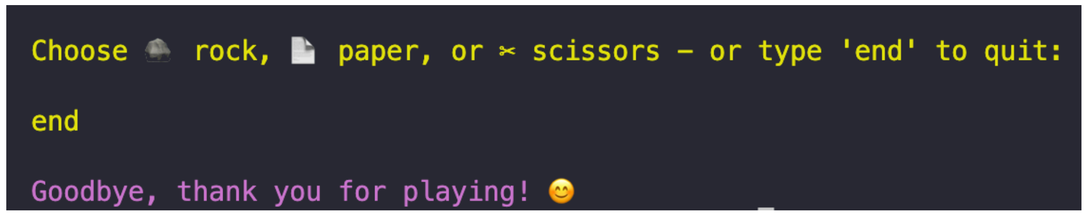


Walkthrough

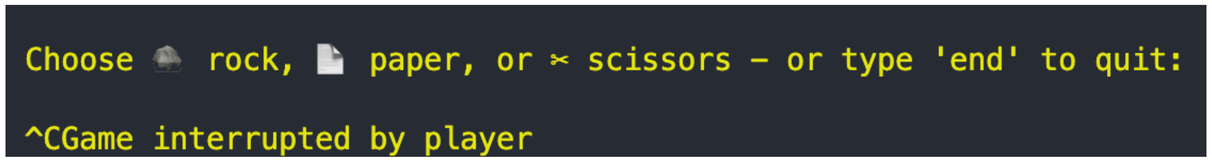
Exiting the game.

- There is two ways we can exit the game.
- The first is by typing ‘end’. Once we type end, a goodbye message is printed to the screen and the game exits.
- The second way is by pressing CTRL + C, which throws a KeyboardInterrupt error, prints that the game was interrupted and then exits the game.

Typing ‘end’



KeyboardInterrupt

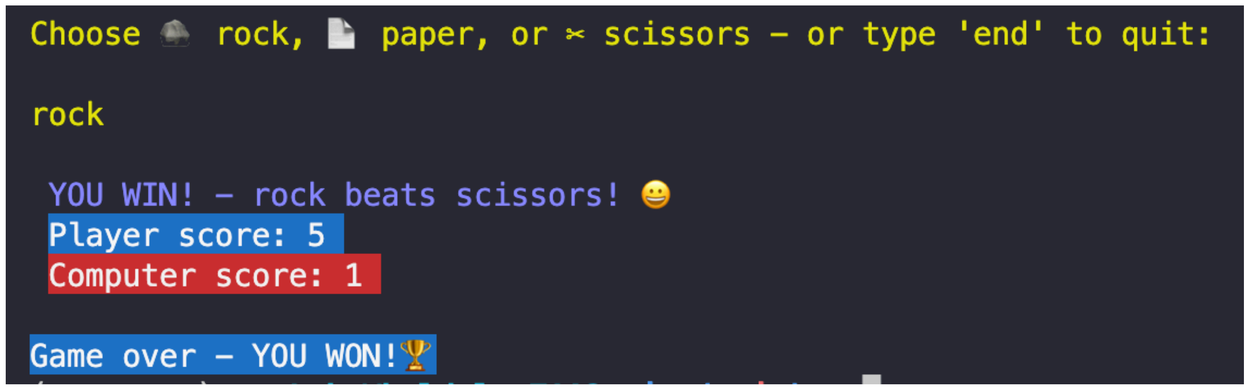


Walkthrough

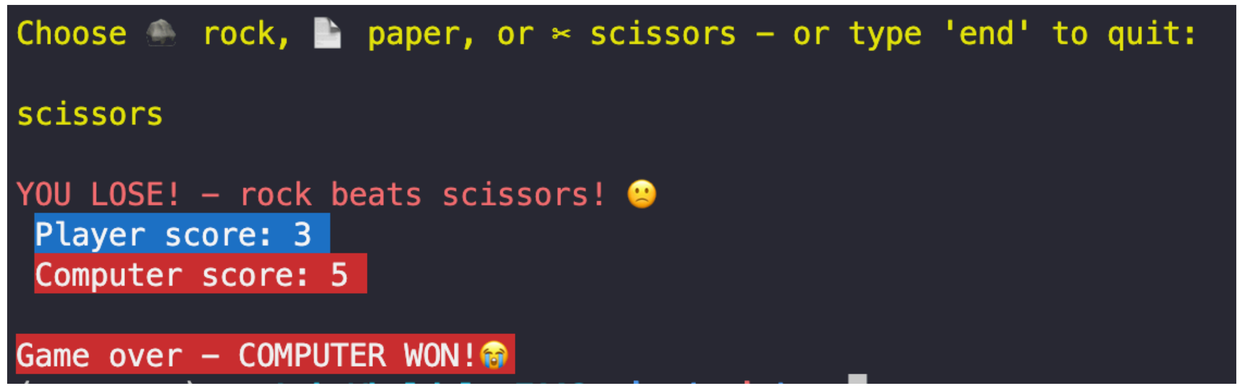
Finishing a game.

- After either the player or computer reaches 5 points, the final score is printed to the screen, and a message print's to the screen saying whether you've won or lost.
- The program then exits, and the scores are saved into a csv file. This file can be deleted before restarting or it can stay so you can observe your previous scores.

Won the game



Lost the game



Code Overview

get_player_choice()

- This function prompts the player to choose between rock, paper or scissors.
- It uses a while loop to keep prompting the player until they provide a valid choice.
- Is the player enters “end”, the function prints a goodbye message and exists the program.
- If the player enters an invalid choice, the function raises a ValueError and prompts the player to choose again.
- Finally, the function returns the player’s choice.

```
def get_player_choice():
    while True:
        try:
            player_choice = input(
                f"\n{fg('yellow')}Choose 🪨  rock, 📄 paper, or ✂ scissors - or type 'end' to
                quit: \n\n").lower()
            if player_choice == "end":
                print(emoji.emojize(
                    f'\n{fg(5)}Goodbye, thank you for playing! 😊', language='alias'))
                exit()
            elif player_choice not in ['rock', 'paper', 'scissors']:
                raise ValueError
            else:
                return player_choice
        except ValueError as e:
            print(f'{fg(28)} {e} is not a valid choice, please choose again.')
```

Code Overview

get_computer_choice()

- This function generates a random choice for the computer (rock, paper, scissors).
- It uses the ‘random’ module to randomly select an item from a list of choices.
- If the element is not in the list of computer_choices an IndexError is thrown and the functions returns None.
- If there is an error generating the choice, the function returns None.

```
def get_computer_choice():
    try:
        computer_choices = ['rock', 'paper', 'scissors']
        computer_choice = random.choice(computer_choices)
        return computer_choice
    except IndexError:
        print('Element not present in list, or list is empty.')
        return None
    except Exception as e:
        print(f'An error occurred: {e}')
        return None
```

Code Overview

play_round()

- This function takes two parameters, ‘player_choice’ and ‘computer_choice’, which represent the choices of the player and the computer respectively.
- It checks if the player and computer have the same choice, if so it returns “draw”.
- Otherwise, it checks the winning combination of choices (rock beats scissors, scissors beats paper, paper beats rock) to determine the winner.
- If the player wins, it returns “player”. Otherwise, if the computer wins, it returns “computer”.

```
def play_round(player_choice, computer_choice):
    if player_choice == computer_choice:
        return ('draw')
    elif player_choice == 'rock' and computer_choice == 'scissors' or player_choice == 'paper'
    and computer_choice == 'rock' or player_choice == 'scissors' and computer_choice == 'paper':
        return 'player'
    else:
        return 'computer'
```

Code Overview

save_scores()

- This function takes two parameters, ‘player_score’ and ‘computer_score’, which represent the current scores of the player and computer respectively.
- It uses the ‘csv’ module to append the scores to a file named “scores.csv”.
- If there is an error writing the scores to the file, the function prints an error message.

```
def save_scores(player_score, computer_score):
    try:
        with open('scores.csv', 'a') as csv_file:
            writer = csv.writer(csv_file)
            writer.writerow([player_score, computer_score])
    except OSError as e:
        print(f'Error writing scores to csv file: {e}')
```

Code Overview

print_scores()

- This function reads the scores from the “scores.csv” file and then prints them to the console.
- It uses the ‘csv’ module to read the scores from the file and calculates the number of wins for each player.
- It then prints the scores to the console, using the ‘colored’ module to colour the text.
- It handles FileNotFoundError incase the file is not being created properly, and also ValueErrors if there was an error reading the file.

```
def print_scores():
    try:
        with open('scores.csv', 'r') as csv_file:
            reader = csv.reader(csv_file)
            player_wins = 0
            computer_wins = 0
            for row in reader:
                player_wins = int(row[0])
                computer_wins = int(row[1])
            print(
                f"{fg(15)} {bg('blue')}Player score: {player_wins} {attr('reset')}")
            print(
                f"{fg(15)} {bg('red')}Computer score: {computer_wins} {attr('reset')}")
    except ValueError:
        print('There was an error reading the scores file.')
    except FileNotFoundError:
        print('The scores file was unable to be found.')
```

Code Overview

- Main function that runs the game.
- It uses a while loop to keep playing rounds until a player has won 5 rounds.
- In each round, it get's the player's choice, generates the computer's choice, and determines the winner using the play_round() function.
- It updates the scores using the save_scores() and prints them to the console using the print_scores() function.
- If either player reaches 5 wins, it prints a message indicating the winner and then exits the game.
- If the player presses CTRL + C, it prints a message and exits the game.
- If there is any other error during the game, it prints an error message and exits the game.

game()

```
def game():
    try:
        # Initial scores
        player_score = 0
        computer_score = 0
        while True:
            player_choice = get_player_choice()
            computer_choice = get_computer_choice()
            winner = play_round(player_choice, computer_choice)
            if winner == 'draw':
                print(
                    f"\n{fg(28)}YOU DREW! - you both picked {player_choice}! {attr('reset')}😮")
            elif winner == 'player':
                player_score += 1
                print(
                    f"\n{fg(21)} YOU WIN! - {player_choice} beats {computer_choice}! {attr('reset')}😀")
            else:
                computer_score += 1
                print(
                    f"\n{fg(9)}YOU LOSE! - {computer_choice} beats {player_choice}! {attr('reset')}🙁")
            save_scores(player_score, computer_score)
            print_scores()

            # End game
            if player_score == 5:
                print(f'\n{bg("blue")}Game over - YOU WON!🏆')
                break
            elif computer_score == 5:
                print(f'\n{bg("red")}Game over - COMPUTER WON!😭')
                break
    except KeyboardInterrupt:
        # CTRL + C to interrupt game at any point
        print('Game interrupted by player')
    except Exception as e:
        print(f'an error occurred during the game: {e}')
```

Code Overview

Testing

- A function to test the play_round() function.
- Tests whether the function returns the correct result when given different inputs.
- There are three types of tests cases: Player winning, computer winning, and draws.

```
def test_play_round():
    # Player wins - This tests whether the play_round() function returns 'player' when the player
    wins.
    assert play_round('rock', 'scissors') == 'player'
    assert play_round('paper', 'rock') == 'player'
    assert play_round('scissors', 'paper') == 'player'

    # Computer wins - This tests whether the play_round() function returns 'computer' when the
    computer wins.
    assert play_round('rock', 'paper') == 'computer'
    assert play_round('paper', 'scissors') == 'computer'
    assert play_round('scissors', 'rock') == 'computer'

    # draw - This tests whether the play_round() function returns 'draw' when both the player and
    the computer choose the same option
    assert play_round('rock', 'rock') == 'draw'
    assert play_round('paper', 'paper') == 'draw'
    assert play_round('scissors', 'scissors') == 'draw'
```

Code Overview

Testing

- A function to test the get_computer_choice() function.
- Uses ‘monkeypatch’ to replace the ‘random.choice()’ function with a ‘lambda’ function that always returns a predefined string.
- Tests whether the function returns the correct string when the random.choice() function is replaced with different strings using the monkey patch module.

```
# Test the get_computer_choice function - using monkeypatch to replace random.choice function and
with a lambda function to always return a predefined string.
def test_get_computer_choice(monkeypatch):
    # Test case 1: simulate the computer choosing 'rock'
    monkeypatch.setattr('random.choice', lambda x: 'rock')
    assert get_computer_choice() == 'rock'

    # Test case 2: simulate the computer choosing 'paper'
    monkeypatch.setattr('random.choice', lambda x: 'paper')
    assert get_computer_choice() == 'paper'

    # Test case 3: simulate the computer choosing 'scissors'
    monkeypatch.setattr('random.choice', lambda x: 'scissors')
    assert get_computer_choice() == 'scissors'
```

Review

Challenges

- User Interface - Designing an intuitive and visually appealing user interface to make it easy for the user to navigate and understand the game.
- Saving scores - Implementing a reliable way of saving the game’s score to a file and retrieving it when needed.
- Error handling - Handling errors was tricky, as I had to think outside the box on certain types of error that may occur rather than just value errors.

Review

Ethical issues

- Accessibility: The application should be accessible to users with disabilities such as visual or hearing impairments. This game is not currently available with sounds, but may look to incorporate this in the future.

- Accuracy of the game: The game should feel fair and unbiased. The user may not trust the application if they feel like the game is rigged.

Review

Favourite parts

- Creative freedom - Having the freedom to design and develop the game with my own ideas and creativity, and using modules such as colored to make it visually appealing.
- Learning opportunities - Developing this application provided me with the ability to learn more about scoping, functions, error handling, testing, file handling, and bash scripting.