# RV COLLEGE OF ENGINEERING®

(Autonomous Institution Affiliated to VTU, Belagavi)

R.V. Vidyaniketan, Bengaluru-560059

**DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS**

****

**Experiential Learning Report on**

**GAMES(ROCK PAPER SCISSOR,TIC TAC TOE , BULL AND COW)**

**I Semester**

**Object-Oriented Programming(22MCA14TL)**

Submitted by

**PRATHAMESH M HIREMATH(1RV22MC069)**

**PRAVEENGOUDA BIRADAR(1RV22MC071)**

**V VIJAY SUKUMAR(1RV22MC108)**

**May-2023**

**RV COLLEGE OF ENGINEERING®**

(Autonomous Institution Affiliated to VTU, Belagavi)

R.V. Vidyaniketan, Bengaluru-560059

# DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

+

**CERTIFICATE**

This is to certify PRATHAMESH HIREMATH (1RV22MC069) ,PRAVEENGOUDA BIRADAR (1RV22MC071) , V VIJAY SUKUMAR (1RV22MC108) that have successfully completed Experiential Learning on **Object-Oriented Programming(22MCA14TL)** in partial fulfilment of I semester MCA for academic year 2022-2023

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **USN** | **NAME** | **PHASE-I**  **(15)** | **PHASE-II**  **(15)** | **Final Marks**  **(30)** |
| 1RV22MC069 | PRATHAMESH HIREMATH |  |  |  |
| 1RV22MC071 | PRAVEENGOUDA BIRADAR |  |  |  |
| 1RV22MC108 | V VIJAY SUKUMAR |  |  |  |

**Director, MCA**

Dr**.** Andhe Dharani

Department of MCA,

RV College of Engineering®, Bengaluru–560059

**Faculty In-charge**

Dr. Prof. SS Nagamuthu Krishnan

Department of MCA,

RV College of Engineering®

Bengaluru –560059

|  |  |  |
| --- | --- | --- |
| **S No.** | **Content** | **Page No.** |
| 1 | Synopsis | 4 |
| 2 | Architecture Diagram | 5 |
| 3 | Modular Description in Detail | 7-10 |
| 4 | Technical Description | 10-12 |
| 5 | Code Snippets | 13-28 |
| 6 | Conclusion and Future Enhancement | 29-30 |

**INDEX**

# SYNOPSIS

### DOMAIN OF THE PROJECT

The Python Game Suite is a collection of three classic games developed using Python programming language: Tic Tac Toe, Rock Paper Scissors, and Bull and Cow. This suite aims to provide an entertaining and interactive experience for players of all ages, with each game offering its unique gameplay mechanics and challenges.

1. Tic Tac Toe:

Tic Tac Toe is a well-known strategy game where two players take turns placing their marks (X or O) on a 3x3 grid. The objective is to form a horizontal, vertical, or diagonal line with three of their marks before the opponent does. The Python implementation of Tic Tac Toe provides an intuitive interface, allowing players to compete against each other in a friendly and competitive environment.

2. Rock Paper Scissors:

Rock Paper Scissors is a classic hand game played between two participants. Players simultaneously choose one of three options: rock, paper, or scissors. The outcome of the game is determined by the rules: rock beats scissors, scissors beat paper, and paper beats rock. The Python Game Suite incorporates an interactive user interface where players can compete against the computer, testing their strategic thinking and quick decision-making skills.

3. Bull and Cow:

Bull and Cow, also known as Mastermind, is a challenging code-breaking game. In this game, one player creates a secret code consisting of a sequence of numbers, and the other player attempts to guess the code within a limited number of attempts. The game provides feedback in the form of "bulls" and "cows," indicating correct numbers in the correct positions (bulls) and correct numbers in the wrong positions (cows). The Python implementation of Bull and Cow offers a user-friendly interface, allowing players to enjoy the excitement of cracking codes and improving their logical reasoning.

The Python Game Suite offers a convenient and engaging platform for playing these classic games. It demonstrates the power and versatility of Python in creating interactive experiences. Whether you're looking to challenge a friend in Tic Tac Toe, test your luck in Rock Paper Scissors against the computer, or engage in a thrilling code-breaking pursuit in Bull and Cow, this game suite has something for everyone. So grab a friend or go solo and embark on a journey filled with fun, strategy, and entertainment with the Python Game Suite!

### OBJECTIVES

**1. Develop a collection of classic games**: The main objective of this project is to create a Python Game Suite that includes three popular games: Tic Tac Toe, Rock Paper Scissors, and Bull and Cow. The aim is to provide a diverse range of gaming experiences and cater to a wide audience.

**2. Enhance user interaction and engagement**: The project aims to create an interactive and engaging environment for players. The games should have user-friendly interfaces, clear instructions, and smooth gameplay, ensuring that players can easily understand and enjoy the gaming experience.

**3. Demonstrate Python programming skills**: The project serves as an opportunity to showcase proficiency in Python programming language. By implementing the game suite using Python, the objective is to demonstrate the ability to utilize key concepts, such as variables, functions, conditional statements, and loops, in creating functional and enjoyable games.

**4. Promote logical thinking and decision-making**: Each game within the suite offers unique challenges that require players to employ strategic thinking, logic, and decision-making skills. The objective is to provide an entertaining platform that also stimulates cognitive abilities, allowing players to improve their problem-solving capabilities while having fun.

**5. Provide options for multiplayer and single-player experiences**: The project aims to accommodate different gaming preferences by offering both multiplayer and single-player modes. This allows players to compete against each other or challenge computer opponents, providing flexibility and ensuring enjoyment regardless of the availability of other players.

**6. Offer a comprehensive and polished gaming experience**: The objective is to create a high-quality game suite with polished interfaces, error handling, and smooth gameplay mechanics. Attention to detail and robust testing will be employed to ensure a seamless and enjoyable experience for players.

Overall, the objective of the Python Game Suite project is to develop a well-rounded collection of classic games, leveraging Python programming skills, while providing engaging, interactive, and enjoyable experiences for players.

**ARCHITECTURE DIAGRAM**

**1**

**2**

Class scoreboard():

//statements

Class grid():

//statements

Class validate():

//statements

Class start(grid,validate):

//statements

class tic\_tac\_toe(start,scoreboard):

Obj1=tic\_tac\_toe

**3**

Obj2=Game()

Obj2.play() //for Single

Obj2=paly()//for Multiple

Class Game():

//statements

Class Multiplayer(Game):

//statements

Class diagram():

//statements

Class user(diagram):

//statements

Class result(user):

//statements

Obj=againplay()

Obj.rules()

Obj.user()

Ob.result1

Obj.again()

1)Rock Paper Scissor

2)Tic Tac Toe

3)Bull and Cow

### MODULE DESCRIPTION

**Rock Paper Scissor**

**draw\_diagram**

The method uses conditional statements (if, elif, and else) to determine which diagram to print based on the value of ch. If ch is equal to 1 or "Rock", it prints a diagram of a rock. If ch is equal to 2 or "Paper", it prints a diagram of a piece of paper. If ch is equal to 3 or "Scissor", it prints a diagram of a pair of scissors. If none of the above conditions are met, it means the input is invalid, and the method prints a message indicating that the input was wrong and asks the user to choose again

**rules(module**)

The code snippet you provided defines a method named rules within a class. This method is responsible for printing the rules and instructions of the Rock Paper Scissors game.

**user1(module)**

The user1 method is used to play a round of the Rock Paper Scissors game between the user and the computer. It follows these steps:

Asks the user to enter their name and stores it. Initializes variables to track the number of wins for the user and the computer. Repeats the following steps five times for five rounds: Asks the user to select an option (rock, paper, or scissors) by entering a number (1, 2, or 3).Randomly selects the computer's choice. Prints the user's choice and the computer's choice using a diagram. Determines the winner of the round and updates the win counts. If the user enters an invalid choice (not 1, 2, or 3), it displays an error message**.**

**Result(module)**

The result class contains two methods: results and result1.The results method determines the winner of a round in the Rock Paper Scissors game. It compares the user's choice and the computer's choice, updates the win counts, and prints the result of the round. The result1 method determines the overall result of the game by comparing the total win counts. It prints the final result, declaring the winner or indicating a draw**.**

**Againplay(module)**

The againplay class inherits from the result class and contains a method called again. This method asks the user if they want to play again or exit the game. If the user chooses to play again, it creates a new instance of the againplay class, plays the game rounds again, prints the final result, and then asks the user again if they want to play. If the user chooses to exit, it prints "Bye" to indicate the end of the game.

**TicTacToe**

**Scorecard(module)**

The scoreboard class has a method called my\_scoreboard that displays a scoreboard for a Tic Tac Toe Python game. The method takes a score\_board dictionary as input, which contains the scores of the two players. The my\_scoreboard method prints a header for the scoreboard and then lists the names and scores of the players based on the score\_board dictionary. It separates the entries with a line of dashes.

**Grid(module)**

The modified tictactoe\_grid() method in the grid class is responsible for displaying the Tic Tac Toe game board. It takes a value parameter which represents the current state of the board. Additionally, it has two optional parameters start\_time and end\_time which can be used to display the start and end times of the game.The method first prints the top row of the game board, followed by the first three boxes. Then, it prints the middle row of the game board and the second set of three boxes. Finally, it prints the bottom row of the game board and the last set of three boxes.The optional start\_time and end\_time parameters can be used to display the start and end times of the game if provided.

**Validate(module)**

The `validate` class has two methods for validating the win or tie conditions in a Tic Tac Toe game. The `win\_validate()` method checks if any of the predefined winning combinations are present in the positions occupied by the current player. It returns `True` if a winning combination is found, indicating that the player has won the game. Otherwise, it returns `False`. The `tie\_validate()` method checks if all positions on the game board are occupied by both players, resulting in a tie. If all positions are filled, it returns `True`, indicating a tie. Otherwise, it returns `False`.

**Start(module)**

The `game\_single()` method in the `start` class executes a single game of Tic Tac Toe. Here's a short explanation of the code:

It initializes the game board and creates a dictionary to track the positions chosen by the players. It enters a game loop that continues until a win or tie is achieved. In each iteration of the loop, it displays the game board, prompts the current player for their move, and validates the input. If the input is valid, it updates the game board and the player's position. It then checks if the current player has won the game by calling the `win\_validate()` method from the `validate` class. If a win is detected, it displays a victory message and returns the winning player. It also checks if the game is tied by calling the `tie\_validate()` method from the `validate` class. If a tie is detected, it displays a tie message and returns "D" to indicate a draw. Finally, it switches the current player for the next turn and continues to the next iteration of the game loop.

**Tictactoe(module)**

The `tic\_tac\_toe` class represents a game of tic-tac-toe. Here's a summary of the code: The `\_\_init\_\_` method initializes the game by getting player names and setting up game-related variables.It captures the start time of the game and prints it. It prompts the players to enter their names. It sets up variables to track the current player, their choices, and the scoreboard. It enters a loop to allow multiple games to be played.In each iteration, it displays a menu for the current player to choose "X", "O", or exit. It handles the player's choice and updates the game accordingly. It determines the winner of the game and updates the scoreboard. It swaps the current player for the next iteration.

**BULL AND COW**

**Game**

This code represents a game of Bulls and Cows. The `Game` class allows players to guess a secret word. Here's a brief explanation:

The `\_\_init\_\_` method initializes the game by setting the secret word and the number of guesses to 0. The `play` method starts the game loop. Inside the loop, the player makes a guess and the guess is evaluated. The number of bulls (correct letters in the correct position) and cows (correct letters in the wrong position) are printed. If the player guesses the word correctly (4 bulls), a congratulatory message is displayed and the game ends. If an invalid guess is made (not a 4-letter word), an error message is shown. If the player interrupts the game, a message is displayed and the game ends.

**Get\_secret\_word()&get\_guess()**

The`get\_secret\_word` method randomly selects a 4-letter word from a predefined list and returns it as the secret word. The `get\_guess` method prompts the user to enter a 4-letter word as their guess. If the input is not a 4-letter word, it raises a `ValueError`. The method returns the valid guess.

**Evaluate\_guess**

The `evaluate\_guess` method compares the user's guess with the secret word and determines the number of bulls and cows.It initializes `bulls` and `cows` variables to 0.It iterates through each character in the guess.If the character at the same index in the guess matches the character in the secret word, it increments the `bulls` count by 1.If the character at the same index in the guess is present elsewhere in the secret word (but not in the same position), it increments the `cows` count by 1.Finally, it returns the counts of bulls and cows.

**MultiplayerGame**

The `MultiplayerGame` class is a subclass of the `Game` class and overrides the `get\_secret\_word` method.It inherits the `\_\_init\_\_` method from the `Game` class using `super().\_\_init\_\_()`.The `get\_secret\_word` method prompts the user to enter the secret word instead of generating a random word.It validates that the secret word has exactly 4 letters, and if not, it prompts the user to enter a 4-letter word until a valid input is provided.Finally, it returns the valid secret word**.**

**TECHNICAL DESCRIPTION**

**Class**

A class is a blueprint or a template for creating objects. It serves as a way to define a new data type with its own properties (attributes) and behaviors (methods).

Attributes are variables that hold data associated with an object. They represent the state or characteristics of an object. For example, a `Car` class might have attributes such as `brand`, `model`, and `year`.

Methods, on the other hand, are functions defined within a class that perform certain actions or operations. They define the behaviors or capabilities of objects. For example, a `Car` class might have methods such as `start\_engine()` and `drive()`.Classes allow us to create multiple instances (objects) based on the same blueprint, each with its own unique set of attribute values. Objects created from a class can access and modify their own attributes and invoke their own methods.

Classes provide several benefits:

Code organization and reusability: Classes help organize code by grouping related data and functions together. They promote reusability, as you can create multiple objects based on the same class.

Encapsulation: Classes encapsulate data and methods, providing a way to hide the internal implementation details and exposing only what's necessary.

Inheritance and polymorphism: Inheritance allows classes to inherit attributes and methods from other classes, promoting code reuse and creating hierarchies of related classes. Polymorphism enables objects of different classes to be used interchangeably, as long as they adhere to the same interface.

Classes are a fundamental concept in object-oriented programming (OOP), which is a programming paradigm focused on organizing code around objects and their interactions. Python is an object-oriented programming language, and classes play a significant role in its design and usage.

**If else**:- In programming, the if-else statement is a control flow statement that allows you to execute different blocks of code based on certain conditions.

The if statement is used to check a condition. If the condition evaluates to True the code block following the if statement is executed. If the condition is False, the code block is skipped.

The else statement is optional and follows an if statement. It is used to specify an alternative code block to execute when the if condition is False. If the if condition is True, the code block under the else statement is skipped.

The condition is an expression that evaluates to either True or False. If the condition is True, the code block under the if statement is executed. Otherwise, the code block under the else statement (if present) is executed.

It's also possible to have multiple elif (short for "else if") statements between the if and else statements. The `elif` statement allows you to specify additional conditions to be evaluated if the preceding `if` or `elif` conditions are `False`. The code block under the first True condition is executed, and the subsequent conditions are not evaluated.

if condition The `if-else` statement is widely used for decision-making and branching in programming, allowing you to create logic that adapts to different situations based on the evaluation of conditions.

**Try-except:-** In Python, the `try-except` statement is used for error handling and allows you to gracefully handle exceptions or errors that may occur during the execution of your code. It helps prevent the program from crashing and allows you to take alternative actions when an error occurs. Here's the general syntax of a `try-except` statement:

Here's how the `try-except` statement works:

1. The code within the `try` block is executed.

2. If an exception occurs during the execution of the `try` block, the code is immediately interrupted, and the exception is raised.

3. The program then looks for an appropriate `except` block that can handle the specific type of exception that was raised.

4. If a matching `except` block is found, the code within that block is executed to handle the exception. Once the exception is handled, the program continues executing from the point immediately after the `try-except` block.

5. If no matching `except` block is found, the exception is propagated up the call stack, and if unhandled, it may cause the program to terminate with an error message.

Error handling with `try-except` statements allows you to anticipate potential errors, handle them appropriately, and maintain the stability and robustness of your program.

**Bulid in Modules**

**Random:** Apologies for the misunderstanding. Here's an explanation of the `random` module in Python without providing examples:

The `random` module in Python is a built-in module that provides functions for generating random numbers, selecting random elements, and shuffling sequences. It is commonly used when you need to introduce randomness or simulate random events in your program.

The `random` module offers various functions to achieve different randomization tasks. Some of the commonly used functions include:

* random(): Generates a random floating-point number between 0 and 1 (inclusive of 0 but exclusive of 1).
* randint(a, b): Generates a random integer between `a` and `b` (inclusive of both a and b).
* choice(sequence): Returns a random element from a given sequence, such as a list, tuple, or string.
* shuffle(sequence): Randomly shuffles the elements in a given sequence.
* sample(sequence, k): Returns a random sample of k elements from a given sequence without replacement.

These functions allow you to introduce randomness in different ways, whether it's generating random numbers within a specific range, selecting random elements, or shuffling sequences randomly. By utilizing the functions from the random module, you can create programs that exhibit random behavior or simulate real-world randomness.

**Time:**

In Python, the time module is a built-in module that provides various functions related to time-related operations. It allows you to work with time values, measure elapsed time, and perform other time-related tasks. The time module is commonly used in applications that require time-related calculations, scheduling, or performance measurements.

**Math:**

In Python, the math module is a built-in module that provides mathematical functions and constants. It allows you to perform various mathematical operations and access commonly used mathematical constants. The math module is widely used in applications that involve mathematical calculations and computations

**CODE SNIPPET**

**Rock paper scissor:-**

import random

import time

Cchoice = ["Rock", "Paper", "Scissor"]

class diagram:

def draw\_diagrams(self, ch):

if (ch == 1) or (ch == "Rock"):

print("""

\_\_\_\_\_\_\_

---' \_\_\_\_)

(\_\_\_\_\_)

(\_\_\_\_\_)

(\_\_\_\_)

---.\_\_(\_\_\_)

""")

elif (ch == 2) or (ch == "Paper"):

print("""

\_\_\_\_\_\_\_

---' \_\_\_\_)\_\_\_\_

\_\_\_\_\_\_)

\_\_\_\_\_\_\_)

\_\_\_\_\_\_\_)

---.\_\_\_\_\_\_\_\_\_\_)

""")

elif (ch == 3) or (ch == "Scissor"):

print("""

\_\_\_\_\_\_\_

---' \_\_\_\_)\_\_\_\_

\_\_\_\_\_\_)

\_\_\_\_\_\_\_\_\_\_)

(\_\_\_\_)

---.\_\_(\_\_\_)

""")

else:

print("WRONG INPUT! CHOOSE AGAIN PLEASE!\n")

class user(diagram):

def rules(self):

print("\tHELLO THERE! WELCOME TO ROCK PAPER SCISSORS GAME!")

print("INSTRUCTIONS:\nBoth the players have three choices namely rock, paper and scissors.")

print("\nGAME RULES:")

print("\tSCISSORS beats PAPER")

print("\tPAPER beats ROCK")

print("\tROCK beats SCISSORS")

print("--------------------------------------------------------------------------")

ts=time.time()

print("Start Time",ts)

def user1(self):

self.a = input("Enter Your Name: ")

self.youwin = 0

self.computerwin = 0

for i in range(1, 6): # 5 Round game

try:

print("Round", i, "Start:")

print("Please select any one option:")

print("1.Rock", "2.Paper", "3.Scissor", sep="\n") # choice any option but in number 1,2,3

self.uch = int(input("Enter Choices: "))

self.cch = random.choice(Cchoice)

print(f"\n{self.a} CHOICE: ")

super().draw\_diagrams(self.uch)

print("\nCOMPUTER's CHOICE: ")

super().draw\_diagrams(self.cch)

result.results(self)

except ValueError:

print("Sorry you Enter Wrong Input Please choose 1, 2, or 3")

class result(user):

def results(self):

if (self.uch==1 and self.cch=="Rock") or (self.uch==2 and self.cch=="Paper") or(self.uch==3 and self.cch=="Scissor"):

self.youwin += 1

self.computerwin += 1

print("This Round is Draw:")

elif (self.uch == 2 and self.cch == "Rock") or (self.uch == 1 and self.cch == "Scissor") or (self.uch == 3 and self.cch == "Paper"):

self.youwin += 1

print(f"{self.a} win this Round")

else:

self.computerwin += 1

print("Computer win this Round")

def result1(self):

if self.youwin > self.computerwin:

print(f"{self.a} Win the game:")

print("Score is:", f"{self.a} score:", self.youwin, "Computer score:", self.computerwin, sep=" ")

elif self.computerwin > self.youwin:

print(f"{self.a} Lose the game. Computer win the game:")

print("Score is:", f"{self.a} score :", self.youwin, "Computer score:", self.computerwin, sep=" ")

else:

print("Match Draw")

print("Score is:", f"{self.a} score:", self.youwin, ",Computer score:", self.computerwin, sep=" ")

te=time.time()

print("End Time",te)

class againplay(result):

def again(self):

self.user\_choice = input("Want to Play again?(Yes/Exit)")

if self.user\_choice == 'yes' or self.user\_choice == 'Yes' or self.user\_choice=='YES':

obj=againplay()

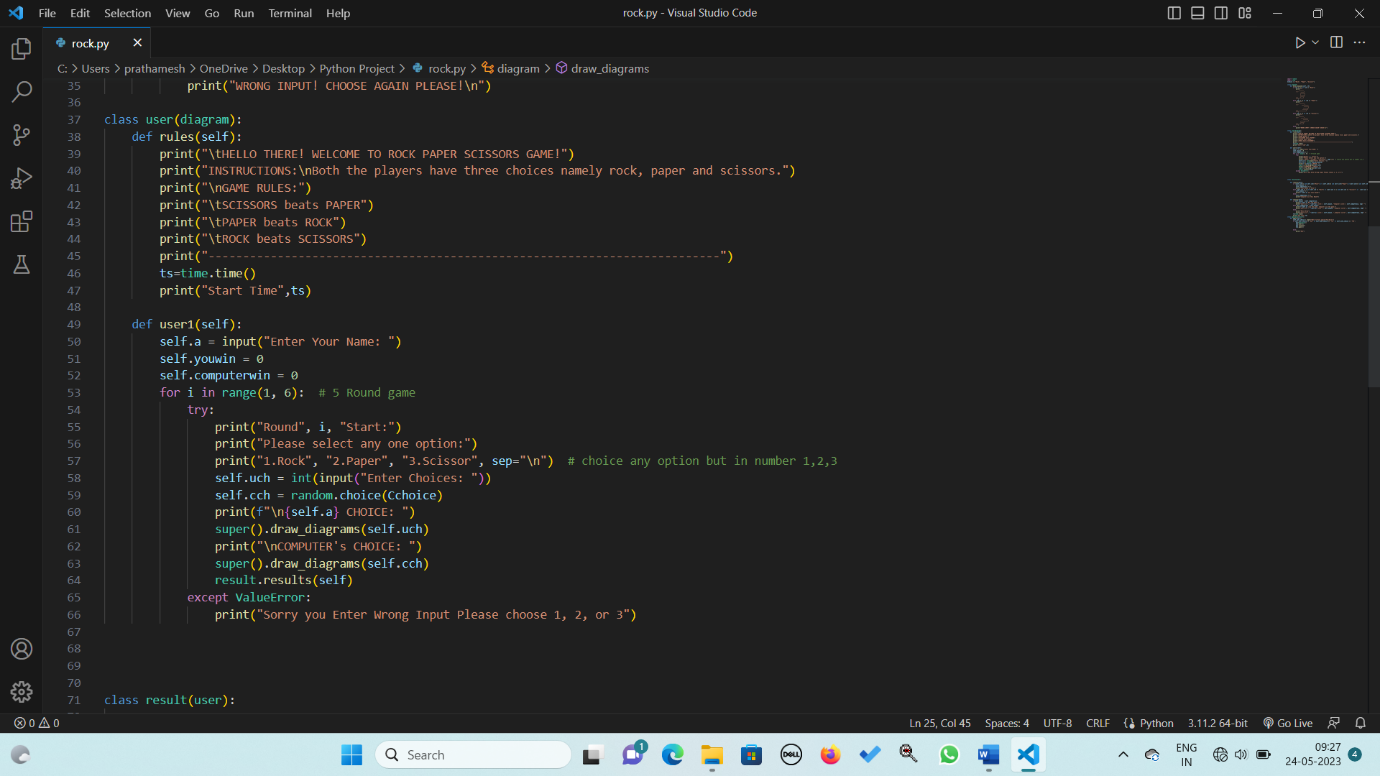
obj.user1()

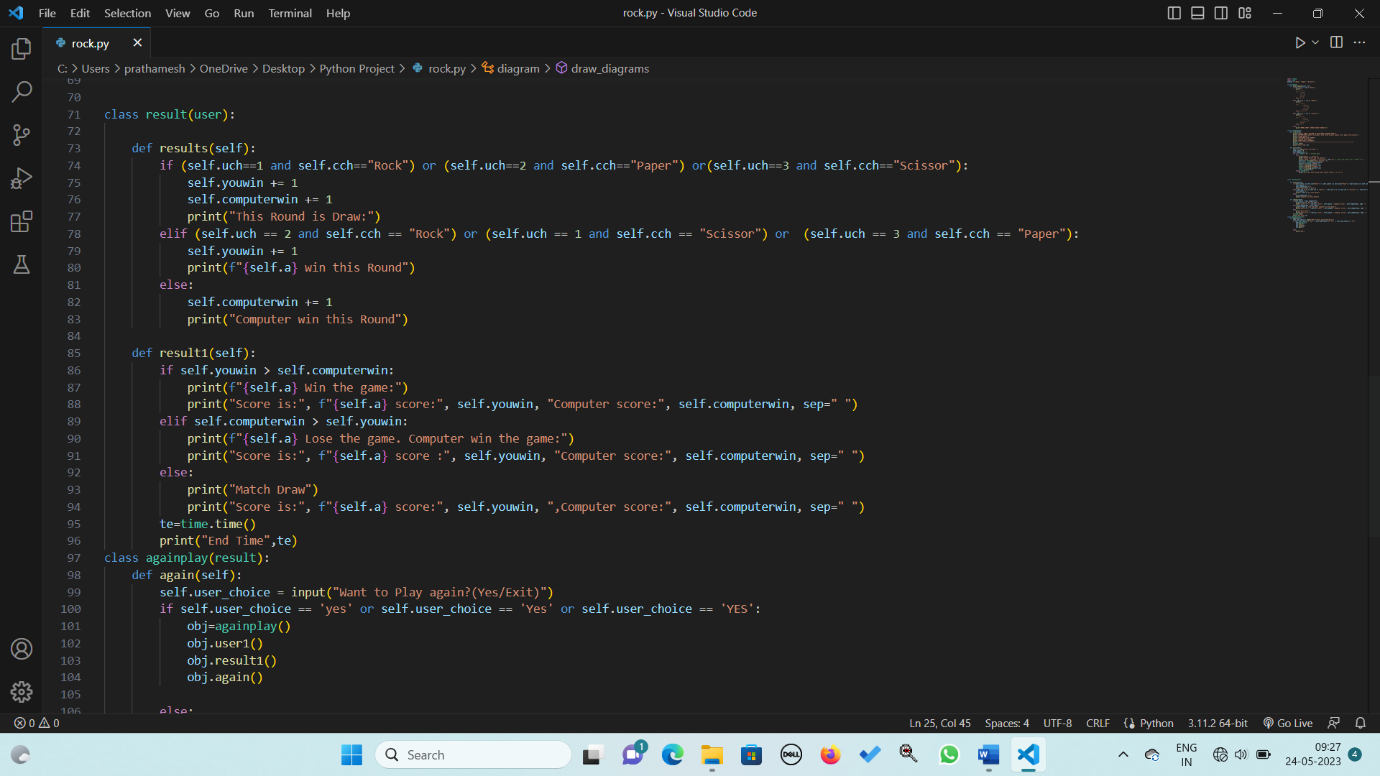
obj.result1()

obj.again()

else:

print("Bye")





**TIC TAC TOE**

import time

###Designing the tic tac toe Python 3X3 grid gameboard of tic tac toe Python

class scoreboard:

# To Design the Scoreboard Use the user-defined function Python to create a scoreboard to capture the scores made

# during the game and display this when the game ends.

def my\_scoreboard(self,score\_board):

print("\t--------------------------------------------")

print("\t The SCOREBOARD for TIC TAC TOE PYTHON GAME")

print("\t--------------------------------------------")

list\_of\_the\_two\_players = list(score\_board.keys())

print(

"\t ",

list\_of\_the\_two\_players[0],

"\t\t",

score\_board[list\_of\_the\_two\_players[0]],

)

print(

"\t ",

list\_of\_the\_two\_players[1],

"\t\t",

score\_board[list\_of\_the\_two\_players[1]],

)

print("\t--------------------------------------------\n")

te=time.time()

print("End Time",te)

class grid:

def tictactoe\_grid(self,value):

print("\n")

print("\t | |")

print("\t {} | {} | {}".format(value[0], value[1], value[2]))

print("\t\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_")

# printing the first three boxes of the 3X3 game board

print("\t | |")

print("\t {} | {} | {}".format(value[3], value[4], value[5]))

print("\t\_\_\_\_\_\_|\_\_\_\_\_\_|\_\_\_\_\_\_")

print("\t | |")

# printing the second three boxes of the 3X3 game board

print("\t {} | {} | {}".format(value[6], value[7], value[8]))

print("\t | |")

print("\n")

class validate:

###To validate the Win or Tie situation of tic tac toe Python

# User-defined function Python for validating the win combinations in the entire tic tac toe Python game

def win\_validate(self,position\_player, player\_current):

# Below are all the possible winning combinations that were analyzed to win the tic tac toe Python game

win\_combinations = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9],

[1, 4, 7],

[2, 5, 8],

[3, 6, 9],

[1, 5, 9],

[3, 5, 7],

]

# using the for loop in Python to validate if any winning combination is getting validated as TRUE or not

for i in win\_combinations:

if all(j in position\_player[player\_current] for j in i):

# If any winning combination is getting validated as TRUE as the function shall return TRUE

return True

# If any winning combination is not getting validated as TRUE as the function shall return FALSE

return False

# User-defined function Python for validating is the tic ta toe Python game is a tie

def tie\_validate(self,position\_player):

if len(position\_player["X"]) + len(position\_player["O"]) == 9:

return True

return False

class start(grid,validate):

def game\_single(self,player\_current):

# function to highlight the tic tac toe Python game

value = [" " for i in range(9)]

position\_player = {"X": [], "O": []}

###To Understand the design of Game-Loop of tic tac toe Python

while True:

super().tictactoe\_grid(value)

# using the while loop to call the tictactoe\_grid function whenever it remains TRUE.

try:

print(

"The player ",

player\_current,

" turn. Now you need to choose your block : ",

end="",

)

chance = int(input())

except ValueError:

print("This is an Invalid Input!!! Please try again!")

continue

if chance < 1 or chance > 9:

print("This is an Invalid Input!!! Please try again!")

continue

if value[chance - 1] != " ":

print("Oops! The position is already filled. Please try again!")

continue

###To update the inputs from a player into the game information of tic tac toe Python

value[chance - 1] = player\_current

# By adding the above code, we update the status of the gameboard

position\_player[player\_current].append(chance)

# By adding the above code, we update the player's position on the grid.

# calling the UDF to check for Win

if super().win\_validate(position\_player, player\_current):

super().tictactoe\_grid(value)

print(

"Hurray! You nailed it! ",

player\_current,

" has won the tic tac toe Python game!",

)

print("\n")

return player\_current

# calling the UDF to check for Tie

if super().tie\_validate(position\_player):

super().tictactoe\_grid(value)

print("It was close! Game is Tied")

print("\n")

return "D"

###To validate switching between the player once the chance of one player is executed

# using the if-else loop in Python to make the switch between the player

if player\_current == "X":

player\_current = "O"

else:

player\_current = "X"

class tic\_tac\_toe(start,scoreboard):

def \_\_init\_\_(self):

ts=time.time()

print("Start Time ",ts)

player\_first = input("Enter first player name: ")

# implmenting the input function in Python to allow the user of the game to input its name.

player\_second = input("Enter second player name: ")

###To capture the game information

# Capturing the player who chooses the X and O

player\_current = player\_first

# Capturing the players' choice

player\_choice = {"X": "", "O": ""}

# Storing the two possible options available for the tic tac toe Python game

option = ["X", "O"]

# Storing the information that needs to be captured in the scoreboard

score\_board = {player\_first: 0, player\_second: 0}

super().my\_scoreboard(score\_board)

###For creating an Outer loop to make our game flexible and allow multiple games of tic tac toe Python to be played

# Using the while function in Python for adding multiple series of games until the players call it an exit.

while True:

# Menu displayed to the players

print(

player\_current,

", you get the chance to make the choice for the series of the Tic tac toe Python game:",

)

print("Please press 1 for X")

print("Please press 2 for O")

print("Please press 3 for Exit")

###To Handle Exception and Allot the selection of the symbol for the current player for each iteration of the game

# The try-except block for the\_choice input from the player

try:

the\_choice = int(input())

except ValueError:

print("This input is Invalid!!! Please Try Again\n")

continue

# if Elif else loop in Python to define the condition for the selection made.

if the\_choice == 1:

player\_choice["X"] = player\_current

if player\_current == player\_first:

player\_choice["O"] = player\_second

else:

player\_choice["O"] = player\_first

elif the\_choice == 2:

player\_choice["O"] = player\_current

if player\_current == player\_first:

player\_choice["X"] = player\_second

else:

player\_choice["X"] = player\_first

elif the\_choice == 3:

print("Thanks for playing the game!!!")

print("The final scores are")

super().my\_scoreboard(score\_board)

break

else:

print("This is an Invalid choice!! Please try again\n")

#Execution of the tic-tac-toe Python game and updating the scoreboard simultaneously

# Capturing the winner of the single game of Tic-Tac-Toe Python

winner = super().game\_single(option[the\_choice - 1])

# As per the scores obtained the winner is getting updated on the scoreboard

if winner != "D":

player\_won = player\_choice[winner]

score\_board[player\_won] = score\_board[player\_won] + 1

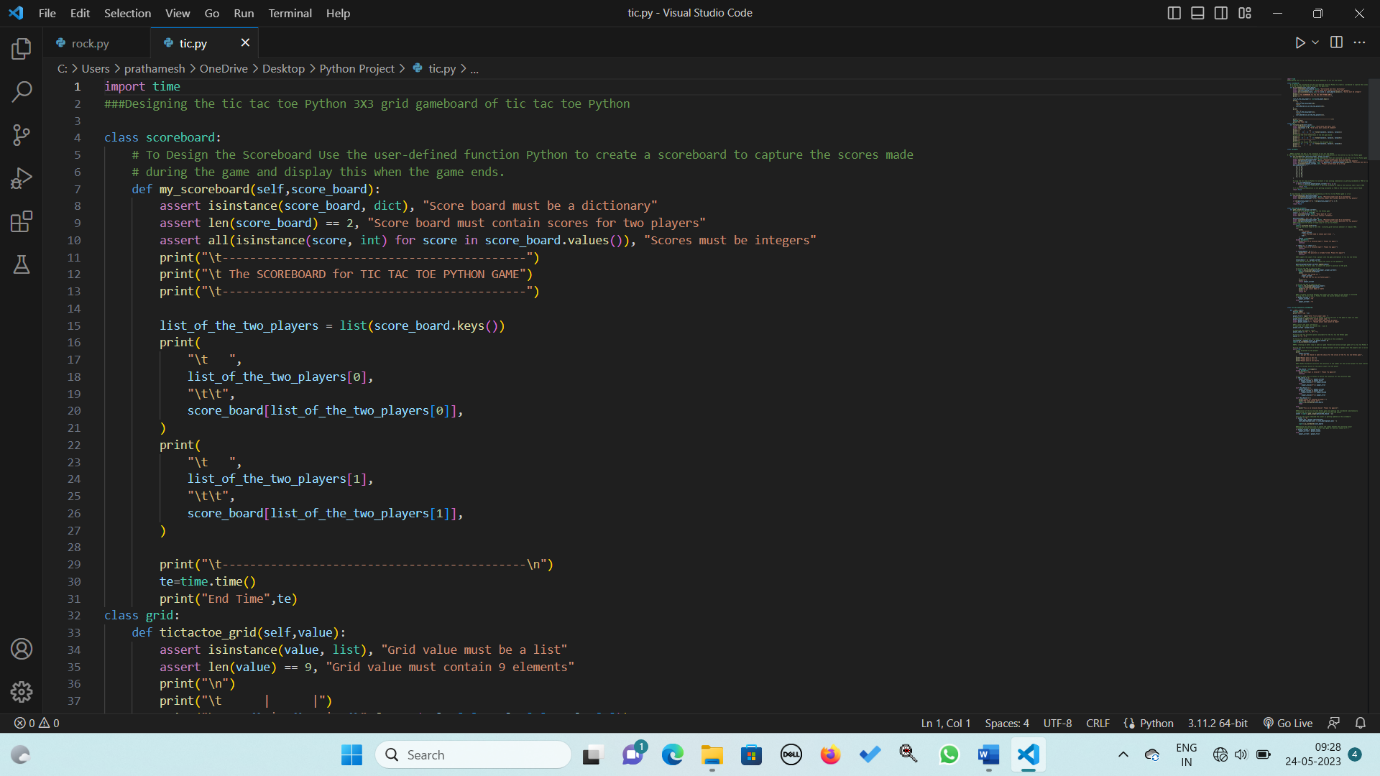
super().my\_scoreboard(score\_board)

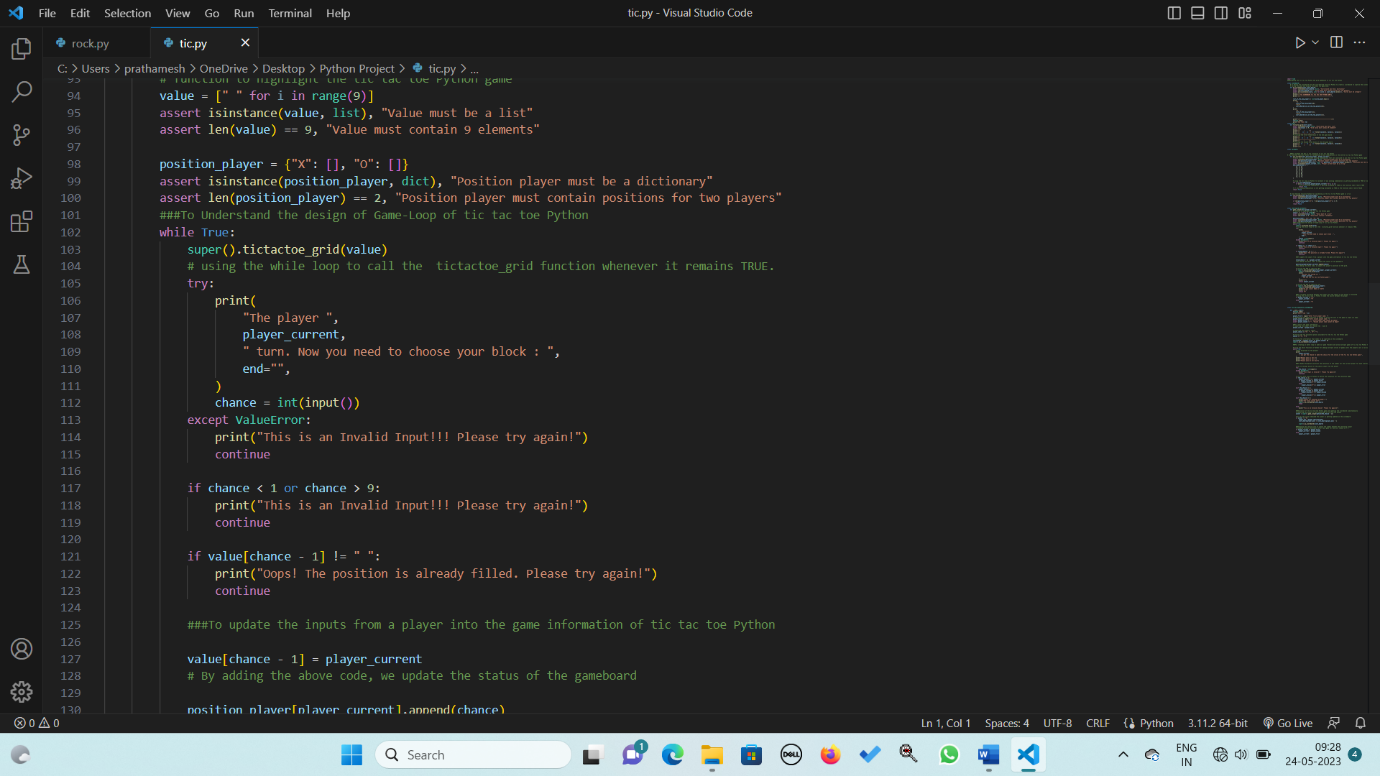
if player\_current == player\_first:

player\_current = player\_second

else:

player\_current = player\_first





**Bull and Cow:-**

import random

class Game:

def \_\_init\_\_(self):

self.secret\_word = self.get\_secret\_word()

self.guesses = 0

def play(self):

print("Let's play Bulls and Cows!")

while True:

try:

self.guesses += 1

guess = self.get\_guess()

bulls, cows = self.evaluate\_guess(guess)

print(f"{bulls} bulls, {cows} cows")

if bulls == 4:

print(f"Congratulations, you guessed the word in {self.guesses} guesses!")

break

except ValueError:

print("Your guess must be a 4-letter word. Please try again.")

except KeyboardInterrupt:

print("\nQuitting game...")

break

def get\_secret\_word(self):

# Define a list of possible 4-letter words

words = ["game", "time", "work", "home", "baby", "city", "idea", "life", "away", "word"]

# Choose a random word from the list

secret\_word = random.choice(words)

return secret\_word

def get\_guess(self):

# Prompt the user to enter their guess

guess = input("Enter your guess (a 4-letter word): ")

# Make sure the guess is a 4-letter word

if len(guess) != 4:

raise ValueError

return guess

def evaluate\_guess(self, guess):

bulls = 0

cows = 0

for i in range(len(guess)):

if guess[i] == self.secret\_word[i]:

bulls += 1

elif guess[i] in self.secret\_word:

cows += 1

return bulls, cows

class MultiplayerGame(Game):

def \_\_init\_\_(self):

super().\_\_init\_\_()

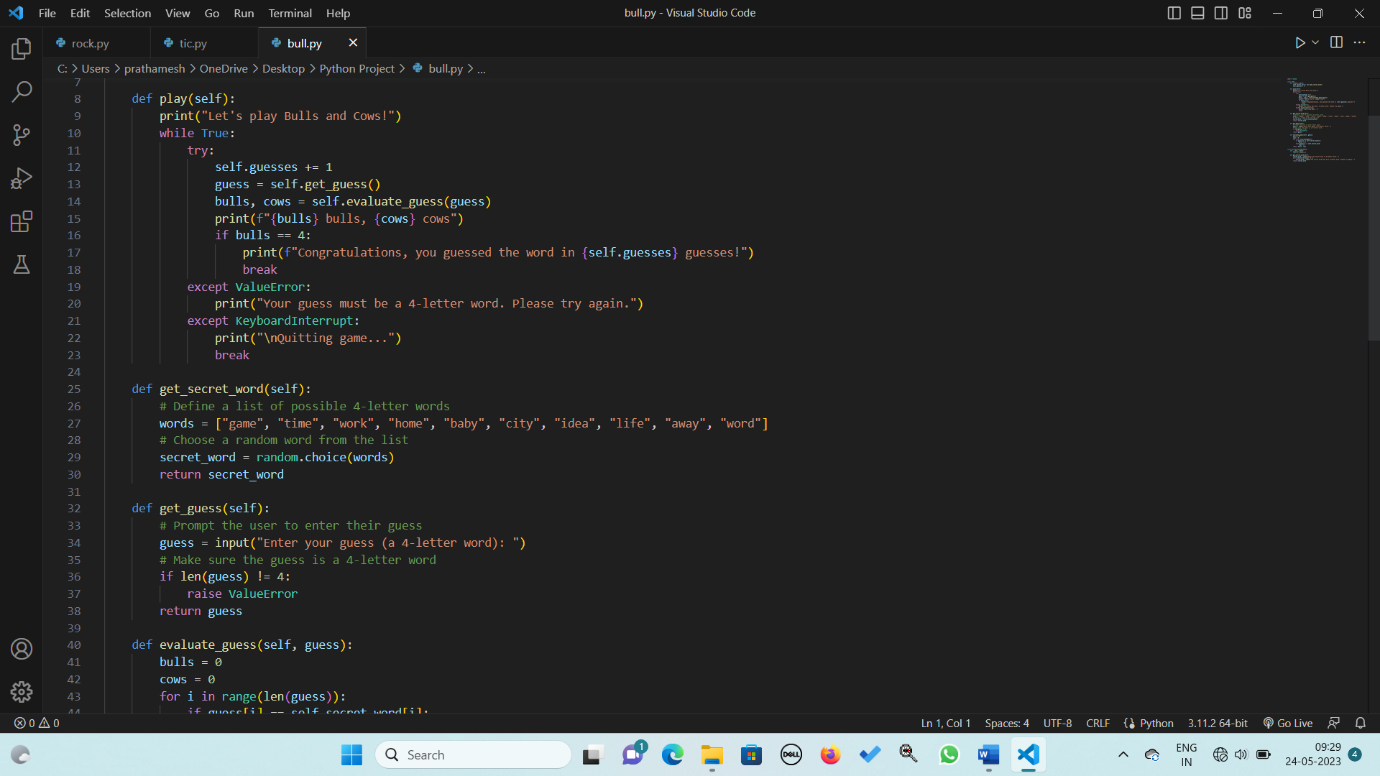
def get\_secret\_word(self)

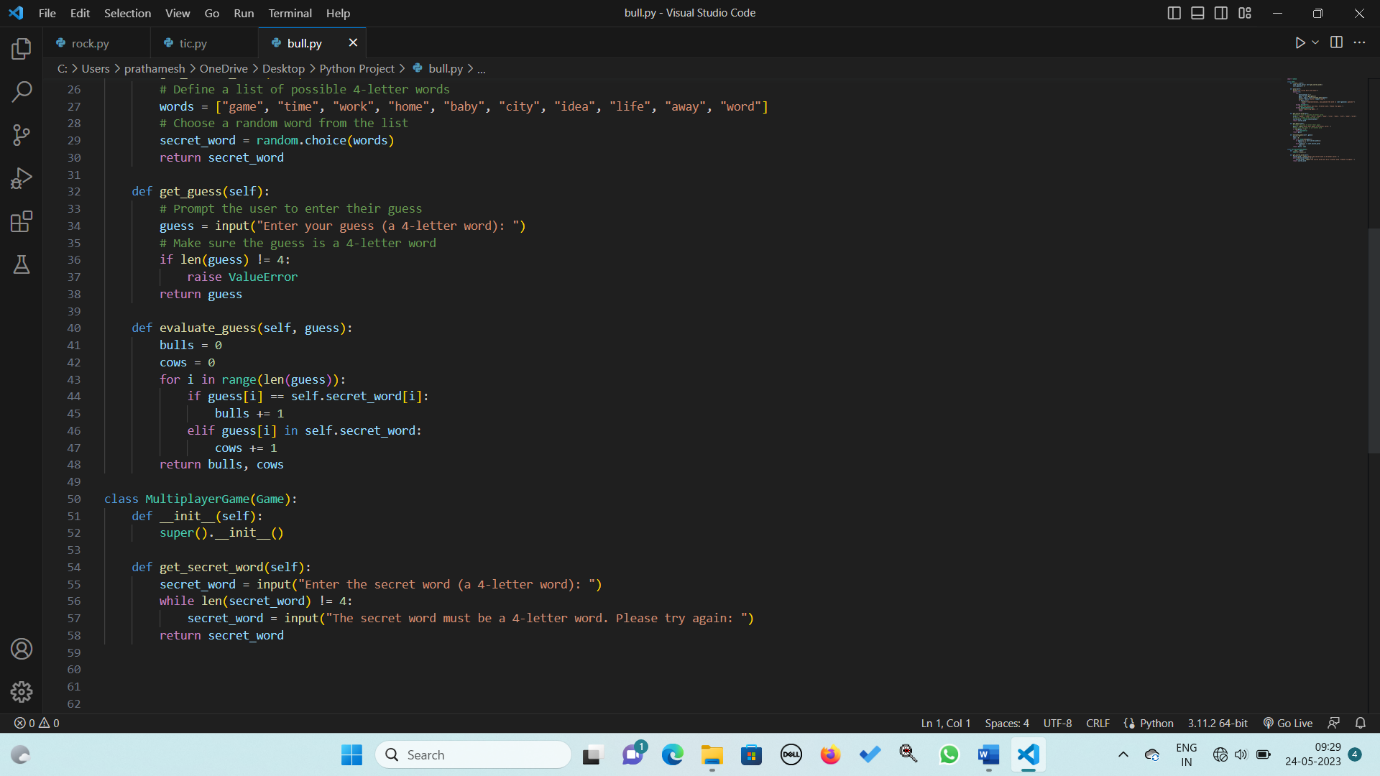
secret\_word = input("Enter the secret word (a 4-letter word): ")

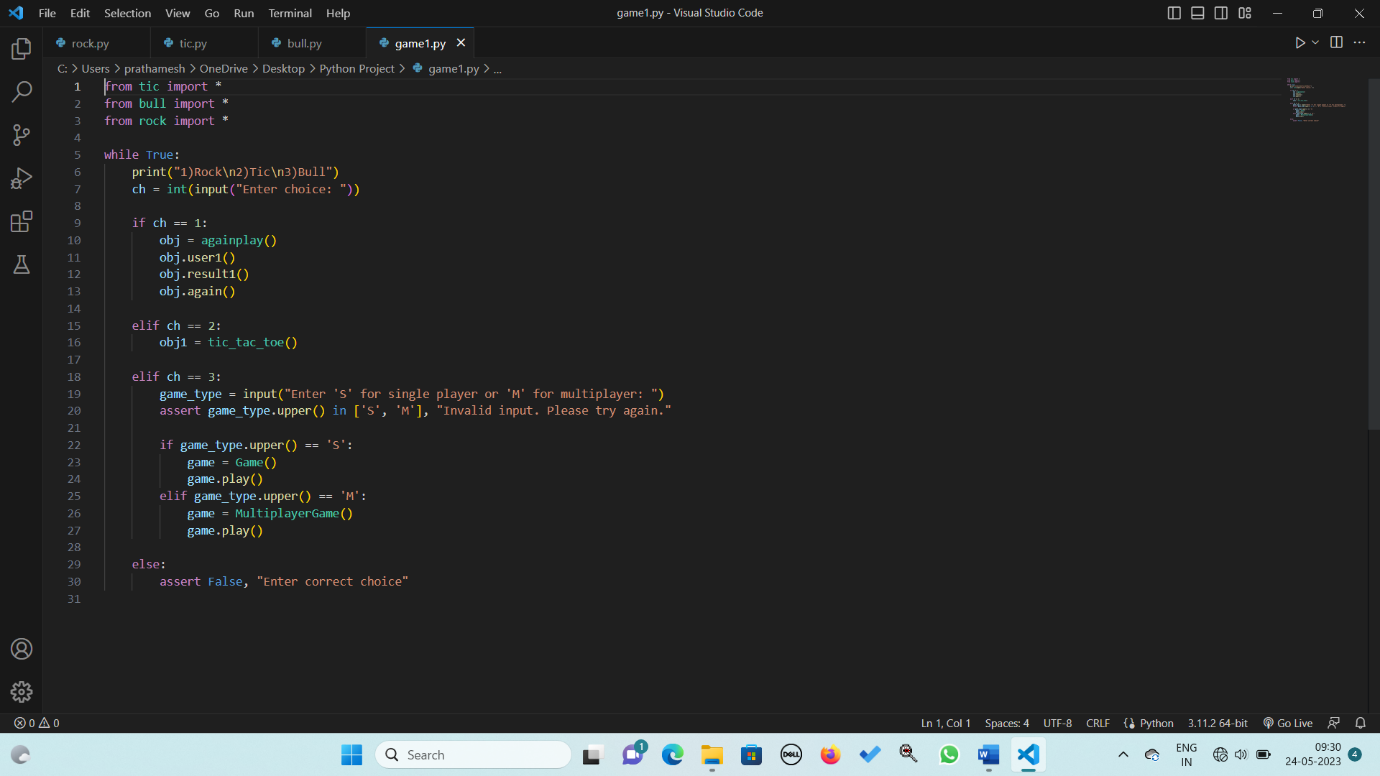
while len(secret\_word) != 4:

secret\_word = input("The secret word must be a 4-letter word. Please try again: ")

return secret\_word

****

****

****

**TEST CASES**

**ROCK PAPER SCISSOR:-**

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.no | DESCRIPTTION | INPUT | EXPECTED OUTPUT |
| 1 | PLAYER WINS | User name: Praveen  User choice: Rock  Computer choice: Scissors | Praveen wins this round.  Updated scores:  Praveen score: 1  Computer score: 0 |
| 2 | COMPUTER WINS | User name: Pratham  User choice: Scissors  Computer choice: Rock | Computer wins this round.  Updated scores:  Pratham score: 0  Computer score: 1 |
| 3 | DRAW | User name: Vijay  User choice: Paper  Computer choice: Paper | This round is a draw.  Updated scores:  Vijay score: 0  Computer score: 0 |
| 4 | MULTIPLE ROUNDS | User name: Praveen  User choices: [Rock, Scissors, Paper, Scissors, Rock]  Computer choices: [Scissors, Paper, Rock, Rock, Scissors] | Praveen wins rounds 1, 3, and 5.  Updated scores:  Praveen score: 3  Computer score: 2 |
| 5 | PLAY AGAIN | User name: Prathamesh  User choices: [Rock, Paper, Scissors]  Computer choices: [Scissors, Rock, Paper]  User chooses to play again  User choices: [Rock, Scissors, Paper]  Computer choices: [Scissors, Rock, Paper]  User chooses to exit | Prathamesh wins rounds 1 and 3.  Updated scores:  Pratham score: 2  Computer score: 1  Program prompts the user to play again.  Pratham wins rounds 1 and 3 (in the second game).  Updated scores:  Pratham score: 4  Computer score: 2  Program prompts the user to play again.  The program ends with a "Bye" message. |

**TEST CASES OF TIC TAC TOE:-**

|  |  |  |  |
| --- | --- | --- | --- |
| SL.NO | DESCRIPTION | INPUT | OUTPUT |
| 1 | Player X Wins | First player name: Praveen  Second player name: Pratham  First player chooses X, Second player chooses O  Game moves: X chooses position 1, O chooses position 5, X chooses position 2, O chooses position 6, X chooses position 3 | Player X (Praveen) wins the game.  Updated scoreboard:  Praveen: 1 point  Pratham: 0 points |
| 2 | Player O Wins | First player name: Praveen  Second player name: Pratham  First player chooses O, Second player chooses X  Game moves: O chooses position 1, X chooses position 5, O chooses position 2, X chooses position 6, O chooses position 3, X chooses position 8 | Player O (Praveen) wins the game.  Updated scoreboard:  Praveen: 1 point  Pratham: 0 points |
| 3 | Tied Game | First player name: Praveen  Second player name: Pratham  First player chooses X, Second player chooses O  Game moves: X chooses position 1, O chooses position 2, X chooses position 3, O chooses position 5, X chooses position 4, O chooses position 6, X chooses position 8, O chooses position 7, X chooses position 9 | Expected Output:  Game ends in a tie.  Updated scoreboard:  Praveen: 0 points  Pratham: 0 points |
| 4 | Empty Scoreboard | No player names or scores are entered. | "The SCOREBOARD for TIC TAC TOE PYTHON GAME" is displayed with no player names or scores.  Test Case 5: Multiple Games - X and O Win |
| 5 | Multiple Games - X and O Win | First player name: Vijay  Second player name: Praveen  First player chooses X, Second player chooses O  Game 1 moves: X chooses position 1, O chooses position 5, X chooses position 2, O chooses position 6, X chooses position 3  Game 2 moves: X chooses position 4, O chooses position 1, X chooses position 5, O chooses position 2, X chooses position 6 | Game 1 ends with Player X (Vijay) winning.  Game 2 ends with Player O (Praveen) winning.  Updated scoreboard:  Vijay: 1 point  Praveen: 1 point |

**TEST CASES OF BULL AND COW:-**

|  |  |  |  |
| --- | --- | --- | --- |
| SL.NO | DESCRIPTION | INPUT | OUTPUT |
| 1 | Secret Word Guessed in First Attempt | Secret word: "game"  Guess: "game" | Congratulations, you guessed the word in 1 guess!  Test Case 2: Secret Word Not Guessed |
| 2 | Word Not Guessed | Secret word: "baby"  Guesses: ["work", "time", "city", "idea"] | 0 bulls, 1 cow  0 bulls, 0 cows  0 bulls, 0 cows  0 bulls, 1 cow  Game continues without a successful guess. |
| 3 | Invalid Guess Length | Secret word: "home"  Guess: "town" | Expected Output:  ValueError: Your guess must be a 4-letter word. Please try again |
| 4 | Interrupted Gameplay | Secret word: "word"  Guesses: ["baby", "idea"]  Interrupted by KeyboardInterrupt (Ctrl+C) | 0 bulls, 0 cows  1 bull, 0 cows  KeyboardInterrupt: Quitting game... |
| 5 | Multiplayer Mode - Secret Word Guessed | Secret word: "work"  Guess: "work" | Congratulations, you guessed the word in 1 guess! |

**CONCLUSION:-**

The program of Rock Paper Scissors game implementation. It allows the user to play against the computer. The user can enter their name, select one of the three options (Rock, Paper, or Scissors), and compete against the computer in a best-of-five rounds game. The program displays the choices made by the user and the computer, determines the winner of each round based on the game rules, and keeps track of the scores. At the end of the game, it declares the overall winner and provides the final scores. The user has the option to play again or exit the game.

The program of tic-tac-toe game implementation. It allows two players to play the game on a 3x3 grid. The program displays the game board, prompts the players to make their moves by selecting a position on the grid, validates the moves, checks for a win or tie condition, and updates the scoreboard accordingly. The players take turns choosing either "X" or "O" symbols. The program allows multiple games to be played in a series and keeps track of the overall scores. The players can exit the game at any time.

The program of Bulls and Cows game implementation. It includes a `Game` class for single-player mode and a `MultiplayerGame` class for multiplayer mode. The player tries to guess a 4-letter secret word, and the program provides feedback in the form of "bulls" and "cows" to indicate the accuracy of the guesses. The game continues until the word is guessed correctly or the player decides to quit.

**Future enhancements for the Python Game :**

1. Additional Games: Expand the game suite by adding more classic games or creating original game concepts. This could include popular games like Sudoku, Hangman, Chess, or card games like Blackjack or Solitaire. Adding a variety of games will further diversify the gaming experience and cater to different preferences.

2. Multiplayer Online Mode: Implement an online multiplayer feature that allows players to compete or collaborate with others over the internet. This could involve integrating network functionality into the games, enabling players to connect with friends or other players worldwide and enjoy multiplayer gameplay.

3. AI Opponents: Enhance the computer opponents in the games by incorporating artificial intelligence algorithms. This would provide players with more challenging and dynamic gameplay experiences. Implementing AI opponents that adapt and learn from player behavior can make the games more engaging and realistic.

4. High Scores and Leaderboards: Create a system to track and display high scores for each game. Implement leaderboards that allow players to compare their scores with others and provide a competitive element. This feature can add replay value and motivate players to improve their skills.

5. Customization Options: Provide players with the ability to customize game settings, such as difficulty levels, game themes, sound effects, or graphics. Allowing players to personalize their gaming experience adds a sense of ownership and increases player engagement.

6. Mobile or Web Versions: Develop mobile or web versions of the Python Game Suite, making it accessible across different platforms and devices. This would enable players to enjoy the games on smartphones, tablets, or through web browsers, broadening the reach and user base.

7. Tutorial or Help System: Include an in-game tutorial or help system that guides new players through the rules and gameplay mechanics of each game. This would improve accessibility for beginners and provide assistance whenever needed.

8. Enhanced Graphics and Visual Effects: Improve the visual presentation of the games by enhancing graphics, animations, and visual effects. This can create a more immersive and visually appealing experience for players.

9. Sound and Music Integration: Add sound effects and background music to enhance the overall gaming atmosphere. Appropriate audio elements can contribute to the enjoyment and engagement of players.

10. Accessibility Features: Implement accessibility features such as adjustable font sizes, colorblind mode, or keyboard shortcuts to make the games more inclusive and accessible to a wider range of players.

These future enhancements can further enrich the Python Game Suite, providing more features, variety, and an enhanced user experience for players.

REFERENCES

SECTION-I BOOKS

1. Beginning Python.
2. Professional Python.
3. Practical Programming.
4. Core Python Programming.

SECTION-II WEBSITES

1. <https://docs.python.org/3/contents.html>
2. <https://www.dataquest.io/blog/using-classes-in-python>
3. <https://www.scaler.com/event/learn-python-from-scratch>