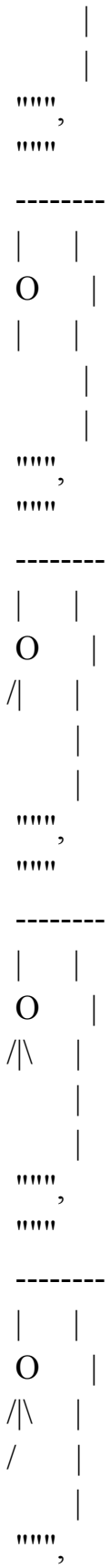
PROJECT CODE:

```
import pdb
import random
import math
import cProfile

def maingame():
    pdb.set_trace()
    print("PYTHON MINI GAMES")
    print("Which game would you like to play?")
    print("1. Random Number Guess")
    print("2. Tic-Tac-Toe")
    print("3. Hangman")
    print("4. Rock Paper Scissors")
    print("5. Dice Simulator")
    print("6. Number Pattern")
    choice = int(input("Enter your choice: "))

    if choice == 1:
        print("RANDOM NUMBER GUESS!")
        number_to_guess = random.randint(1, 100)
        no_of_attempts = 0

        if number_to_guess > 9:
            print("The number to guess is a 2-digit number")
        else:
            print("The number is a 1-digit number")

        while no_of_attempts < 6:
            attempt = int(input("Enter your guess: "))

            if attempt > number_to_guess:
                print("Too high. Guess again!")
```

```
            elif attempt < number_to_guess:
                print("Too low. Guess again!")
            else:
                print("Right guess!!")
                break

            no_of_attempts += 1

        if no_of_attempts == 6:
            print("Oops! You lost!")
            print(f"The number was: {number_to_guess}")

        pdb.set_trace()

    elif choice == 2:

        print("TIC-TAC-TOE")
        board = [" " for _ in range(9)]

        def print_board():
            print(f"{board[0]} | {board[1]} | {board[2]}")
            print("---------")
            print(f"{board[3]} | {board[4]} | {board[5]}")
            print("---------")
            print(f"{board[6]} | {board[7]} | {board[8]}")

        def check_win(player):
            win_combinations = [(0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5,
8), (0, 4, 8), (2, 4, 6)]

            for combo in win_combinations:
                if all(board[i] == player for i in combo):
                    return True
            return False
```

```
        players = ['X', 'O']
        current_player = players[0]

        while True:
            print_board()
            position = int(input(f"Player {current_player}, enter a position (0-8): "))

            if board[position] == " ":
                board[position] = current_player
            else:
                print("Position already occupied. Try again.")
                continue

            if check_win(current_player):
                print_board()
                print(f"Player {current_player} wins!")
                break

            if " " not in board:
                print_board()
                print("It's a draw!")
                break

            current_player = players[1] if current_player == players[0] else players[0]

        pdb.set_trace()

    elif choice == 3:
        print("HANGMAN")
        word_list = ["python", "hangman", "programming", "computer", "code",
"variable", "function", "loop", "list", "dictionary", "class",
                     "object", "exception", "algorithm", "bug", "debugging", "compiler",
"syntax", "logic", "database", "server", "client",
                     "HTTP", "API", "framework", "library", "frontend", "backend", "UI",
"UX", "CSS", "HTML", "JavaScript", "Java", "C++",
```

```
            "Python", "Ruby", "PHP", "SQL", "git", "repository", "merge", "pull request", "deployment", "testing", "requirement",
            "architecture", "design pattern", "module", "package", "namespace", "lambda", "tuple", "inheritance", "polymorphism",
            "encapsulation", "abstraction", "recursion", "variable scope", "data structure", "runtime", "compilation", "testing",
            "debugging", "integration", "efficiency", "scalability", "security", "version control", "algorithm", "data science",
            "machine learning", "artificial intelligence", "cybersecurity", "cloud computing", "Big Data", "data analysis", "IoT",
            "blockchain", "virtualization", "containerization", "microservices", "REST", "SOAP", "JSON", "XML", "web development",
            "mobile app development", "framework", "template", "dependency", "SDK", "IDE", "frontend", "backend", "database", "ORM",
            "CMS", "web server", "hosting", "deployment"]

    word_to_guess = random.choice(word_list)
    guessed_letters = set()
    max_attempts = 6
    attempts = 0

    hangman_stages = [
        """
        --------
        |      |
               |
               |
               |
               |
        """,
        """
        --------
        |      |
        O      |
               |
```



```
        |
        |
 """",
 """"
 --------
 |      |
 O      |
 |      |
        |
        |
 """",
 """"
 --------
 |      |
 O      |
/|      |
        |
        |
 """",
 """"
 --------
 |      |
 O      |
/|\     |
        |
        |
 """",
 """"
 --------
 |      |
 O      |
/|\     |
/       |
        |
 """",
```

```
    """
    --------
    |      |
    O      |
   /|\     |
   / \     |
           |
    """,
]

display_word = ["_"] * len(word_to_guess)

def display():
    print(" ".join(display_word))

while attempts < max_attempts:
    display()
    guess = input("Guess a letter: ").lower()

    if len(guess) != 1 or not guess.isalpha():
        print("Please enter a single letter.")
        continue

    if guess in guessed_letters:
        print("You already guessed that letter.")
        continue

    guessed_letters.add(guess)

    if guess in word_to_guess:
        for i in range(len(word_to_guess)):
            if word_to_guess[i] == guess:
                display_word[i] = guess
    else:
        attempts += 1
```

```
            print("Wrong guess! You have {} attempts left.".format(max_attempts -
attempts))
            print(hangman_stages[attempts])

        if "_" not in display_word:
            display()
            print("Congratulations, you've won!")
            break

    if "_" in display_word:
        print("Out of attempts. The word was: {}".format(word_to_guess))
    pdb.set_trace()

elif choice == 4:
    print("ROCK PAPER SCISSORS")
    possible_choices = ["rock", "paper", "scissors"]
    computer_choice = random.choice(possible_choices)
    no_of_points = int(input("How many points do you want to play for? "))
    no_of_attempts = 0
    player_points = 0
    computer_points = 0
    while no_of_attempts < no_of_points:
        player_choice = input("Enter your choice: rock/paper/scissors: ").lower()
        if player_choice not in possible_choices:
            print("Invalid choice. Please choose from: rock, paper, scissors.")
            continue
        if player_choice == computer_choice:
            print("It's a tie! The computer chose {} ".format(computer_choice))
        elif (
            (player_choice == "rock" and computer_choice == "scissors")
            or (player_choice == "paper" and computer_choice == "rock")
            or (player_choice == "scissors" and computer_choice == "paper")
        ):
            player_points += 1
            print("You won! The computer chose {} ".format(computer_choice))
```

```
            else:
                computer_points += 1
                print("You lost! The computer chose {} ".format(computer_choice))
            print("Points: You: {} Computer: {}".format(player_points,
computer_points))
            no_of_attempts += 1
        print("GAME OVER!")
        if computer_points < player_points:
            print("You won! Congratulations!")
            print("Your points:{}".format(player_points))
            print("Computer's points:{}".format(computer_points))
        elif computer_points > player_points:
            print("Sorry! You lost.")
            print("Your points:{}".format(player_points))
            print("Computer's points:{}".format(computer_points))
        else:
            print("It's a tie!")

    elif choice == 5:
        print("DICE SIMULATOR")
        dice_numbers = int(input("Enter the number of faces on your dice:"))
        random_choice = random.randint(1, dice_numbers)
        print("Your random choice is: {}".format(random_choice))
        pdb.set_trace()

    elif choice == 6:
        def arithmetic_sequence(start, diff, length):
            return [start + diff * i for i in range(length)]

        def geometric_sequence(start, ratio, length):
            return [start * (ratio ** i) for i in range(length)]

        def fibonacci_sequence(length):
            sequence = [0, 1]
            while len(sequence) < length:
```

```
        next_term = sequence[-1] + sequence[-2]
        sequence.append(next_term)
    return sequence

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            return False
    return True

def prime_sequence(length):
    sequence = []
    num = 2
    while len(sequence) < length:
        if is_prime(num):
            sequence.append(num)
        num += 1
    return sequence

def squares_sequence(length):
    return [i**2 for i in range(1, length + 1)]

def powers_of_2(length):
    return [2**i for i in range(length)]

def powers_of_3(length):
    return [3**i for i in range(length)]

def even_numbers(length):
    return [i for i in range(2, 2 * length + 2, 2)]

def odd_numbers(length):
    return [i for i in range(1, 2 * length + 1, 2)]
```

```
    def generate_pattern():
        pattern_types = [
            ("Arithmetic", lambda: arithmetic_sequence(random.randint(1, 10),
random.randint(1, 5), random.randint(5, 10))),
            ("Geometric", lambda: geometric_sequence(random.randint(2, 5),
random.randint(2, 4), random.randint(5, 10))),
            ("Fibonacci", lambda: fibonacci_sequence(random.randint(5, 10))),
            ("Prime", lambda: prime_sequence(random.randint(5, 10))),
            ("Squares", lambda: squares_sequence(random.randint(5, 10))),
            ("Powers of 2", lambda: powers_of_2(random.randint(5, 10))),
            ("Powers of 3", lambda: powers_of_3(random.randint(5, 10))),
            ("Even Numbers", lambda: even_numbers(random.randint(5, 10))),
            ("Odd Numbers", lambda: odd_numbers(random.randint(5, 10)))
        ]

        pattern_type, generate_func = random.choice(pattern_types)
        pattern = generate_func()
        return pattern, pattern_type

    def calculate_expected_next_term(pattern, pattern_type):
        last_term = pattern[-1]

        if pattern_type == "Arithmetic":
            common_difference = pattern[1] - pattern[0]
            expected_next_term = last_term + common_difference

        elif pattern_type == "Geometric":
            common_ratio = pattern[1] / pattern[0]
            expected_next_term = last_term * common_ratio

        elif pattern_type == "Fibonacci":
            expected_next_term = pattern[-2] + last_term

        elif pattern_type == "Prime":
```

```
        num = last_term + 1
        while not is_prime(num):
            num += 1
        expected_next_term = num

    elif pattern_type == "Squares":
        expected_next_term = int(math.sqrt(last_term)) + 1
        expected_next_term = expected_next_term ** 2

    elif pattern_type == "Powers of 2":
        expected_next_term = last_term * 2

    elif pattern_type == "Powers of 3":
        expected_next_term = last_term * 3

    elif pattern_type == "Even Numbers":
        expected_next_term = last_term + 2

    elif pattern_type == "Odd Numbers":
        expected_next_term = last_term + 2

    return expected_next_term

print("Welcome to the Number Pattern Game!")
score = 0

while True:
    pattern, pattern_type = generate_pattern()
    print(f"Find the next term in this {pattern_type} pattern:")
    print(pattern)

    user_input = input("Your answer: ")

    try:
        user_answer = int(user_input)
```

```
            except ValueError:
                print("Invalid input. Please enter an integer.")
                continue

            expected_next_term = calculate_expected_next_term(pattern, pattern_type)

            if user_answer == expected_next_term:
                print("Correct!")
                score += 1
            else:
                print("Wrong. The correct answer is {}.".format(expected_next_term))

            play_again = input("Play again? (y/n): ").strip().lower()
            if play_again != 'y':
                break

        print("Game Over!")
        print("Your final score: {}".format(score))

    play_again = input("Play again? (y/n): ").strip().lower()
    if play_again != 'y':
        print("Bye!")
        exit()
    else:
        print("\n")
        maingame()
        pdb.set_trace()

maingame()

profiler = pstats.Stats('profile_results.pstats')
profiler.strip_dirs().sort_stats('cumulative').print_stats()
```