python-assignmentog

May 27, 2025

[1]:

```
# GuessTheNumber.py
import random
n= random.randint(1,10)
print(' Name: Thanushri K S \n USN: 1AY24AI112\n Section: O')
user = int(input('Eanter a number between 1 to 9: '))
print('computer guess: '+str(n))
if (user==n):
    print(' Your choice is correct')
else:
    print('Your choice is incorrect! Try again')
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
```

```
Eanter a number between 1 to 9:  6
```

```
computer guess: 7
Your choice is incorrect! Try again
```

[3]:

```
#RockPaperScissors.py
import random
print(' Name: Thanushri K S \n USN: 1AY24AI112\n Section: O')
name=input("Enter the name of the player:")
choice=int(input("Enter any one of the below(1-ROCK,2-PAPER,3-SCISSORS):"))
while choice > 3 or choice < 1:
        choice = int(input("Enter a valid choice please"))
if choice == 1:
        choice_n = 'Rock'
elif choice == 2:
        choice_n = 'Paper'
else:
     choice_n = 'Scissors'
print(name,"choice is:", choice_n)
print("Now its computer turn")
comp_choice= random.randint(1, 3)
if comp_choice == 1:
        comp_choice_name = 'Rock'
elif comp_choice == 2:
        comp_choice_name = 'Paper'
else:
    comp_choice_name = 'Scissors'
print("Computer choice is:", comp_choice_name)
if choice == comp_choice:
        result = "Draw"
elif (choice == 1 and comp_choice == 2) or (comp_choice == 1 and choice == 2):
        result = 'Paper'
elif (choice == 1 and comp_choice == 3) or (comp_choice == 1 and choice == 3):
        result = 'Rock'
elif (choice == 2 and comp_choice == 3) or (comp_choice == 2 and choice == 3):
        result = 'Scissors'
if result=="Draw":
    print("It's a tie!")
elif result == choice_n:
    print(name,"is the Winner")
else:
    print("Computer wins!")
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
```

```
Enter the name of the player: Thanushri K S
Enter any one of the below(1-ROCK,2-PAPER,3-SCISSORS): 1
```

```
Thanushri K S choice is: Rock
Now its computer turn
Computer choice is: Rock
It's a tie!
```

[7]:

```
# Zigzag.py
def print_zigzag(rows):
    if rows < 3:
        print("Please enter a number of rows greater than or equal to 3 for a␣
 ↪proper zigzag.")
        return

    n = (rows + 1) // 2

    for i in range(rows):
        for j in range(n * (rows - 1)):
            if (i % (rows - 1) == 0 and j % (rows - 1) == 0) or \
               (i % (rows - 1) == (rows - 2) and (j + 1) % (rows - 1) == 0) or \
               (0 < i % (rows - 1) < (rows - 2) and j % (rows - 1) == i % (rows␣
 ↪- 1)):
                print("*", end="")
            else:
                print(" ", end="")
        print()

if __name__ == "__main__":
    print(' Name: Thanushri K S\n USN: 1AY24AI112\n Section: O')

    num_rows = int(input("Enter the number of rows for the zigzag pattern: "))
    print_zigzag(num_rows)
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
```

```
Enter the number of rows for the zigzag pattern:  10
```

```
*        *        *        *        *
 *        *        *        *        *
  *        *        *        *        *
   *        *        *        *        *
    *        *        *        *        *
     *        *        *        *        *
      *        *        *        *        *
       *        *        *        *        *
        *        *        *        *        *
*        *        *        *        *
```

[8]:

```
#CollatzSequence.py
def collatz_sequence(n):
    if not isinstance(n, int) or n <= 0:
        print("Please enter a positive integer.")
        return

    print(n, end=" ")
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        print(n, end=" ")
    print()

if __name__ == "__main__":
    print(' Name: Thanushri K S \n USN: 1AY24AI112 \n Section: O')
    start_number = int(input("Enter a positive integer to start the Collatz␣
 ↪sequence: "))
    collatz_sequence(start_number)
```

```
 Name: Thanushri K S
```

```
USN: 1AY24AI112
Section: O
```

```
Enter a positive integer to start the Collatz sequence:  3
```

```
3 10 5 16 8 4 2 1
```

[10]:

```
# ConwaysGameOfLife.py
import time
import random
import os

def create_grid(rows, cols):
    return [[0 for _ in range(cols)] for _ in range(rows)]

def randomize_grid(grid, density=0.3):
    rows = len(grid)
    cols = len(grid[0])
    for i in range(rows):
        for j in range(cols):
            if random.random() < density:
                grid[i][j] = 1

def get_neighbors(grid, row, col):
    rows = len(grid)
    cols = len(grid[0])
    live_neighbors = 0
    for i in range(max(0, row - 1), min(rows, row + 2)):
        for j in range(max(0, col - 1), min(cols, col + 2)):
            if (i, j) != (row, col) and grid[i][j] == 1:
                live_neighbors += 1
    return live_neighbors

def next_generation(grid):
    rows = len(grid)
    cols = len(grid[0])
    new_grid = create_grid(rows, cols)
    for i in range(rows):
        for j in range(cols):
            live_neighbors = get_neighbors(grid, i, j)
            if grid[i][j] == 1:
                if live_neighbors == 2 or live_neighbors == 3:
                    new_grid[i][j] = 1
            else:
                if live_neighbors == 3:
                    new_grid[i][j] = 1
    return new_grid
```

```python
def print_grid(grid):
    os.system('cls' if os.name == 'nt' else 'clear')
    for row in grid:
        print(''.join(['*' if cell == 1 else ' ' for cell in row]))

if __name__ == "__main__":
    print(' Name: Thanushri K S\n USN: 1AY24AI112\n Section: O')
    rows = 20
    cols = 40
    generations = 10
    update_interval = 0.2

    grid = create_grid(rows, cols)
    randomize_grid(grid, density=0.2)

    for generation in range(generations):
        print(f"Generation: {generation + 1}")
        print_grid(grid)
        grid = next_generation(grid)
        time.sleep(update_interval)

    print("Game of Life simulation ended.")
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
Generation: 1
   *       * ***       *   *   * *     *
* * ****       *   **   *    **
  **         *  **  *                *
     * *      ** ***  *      **    *
 * *       *   *          *         *
*       *                  *
              *           *    *      **
      *             *    * *         *
     *     **  *            *      *
      *  *          *   **   **        *
     ***   * *   *          *  * *
    **               **  **    *     * *
      *     ** *    * *   *     *  *
      *     *    *         ** * *
 ** *        **    **  **       **
 *    *  *    **      *        *   *
   * *   *        *         *       *
 * * * * *           *            * ***
**   *  *      *    *  * *  * *  *   *
 **     **   *    *  ** *  *  *    * *
```

```
Generation: 2
   ****       **              *
 ** ***     ** *   **         *
 ***   *         *  **
   **         ** ***                *
              *** *
                          **
                          **          *
                          **          *
     **                 *** **
       *  **                ***
    *  *             *  * *  * **     *
    *  *     **      **  ***   *
      *     **        *  **     *
     *      *      *** *   *    *
 **  *       ***       *        **
 * ***       ***   *   *        *
     *   **                         *
**   *   *                   *     ** *
*   * **            *  **    *    **
***     **           ****    *      *
Generation: 3
  **  *      ***
 *     *     * **  ***
 *    *      * * *   *
   **         *    **
              * * **
               *          **
                         *  *

      *                  **   *
     * *                * *    *
      ***            **   *  * **
     ***    ***      ****  *  **
     **     * *        ***     **
     **     *       ** ** *    **
 *** **     *  *   *   **      ***
 * * **      * *                **
***  **  **   *                    ***
**  **  ***                       * **
  *  *** *           *  *   ***   *  *
**     **            ** *          *
Generation: 4
  *          * **    *
 *    **    **      **
  *          * *  *  *
             ** ** **
              *   ***
```

```
               *          **
                          **
                         **
      *                  **
     *  *                 **   **
        *    *       *  * **    *
        *   * *      *    *
    *      ** *     *     *
  *    *   **       *         *
 * *   *    ***     * ****
   *   *     * *               * ** *
   *   ** *   *                  *** *
*  **                        *    *   *
  * **    *          ***     *    *  *
 *     **            ***     *
Generation: 5
            ***     **
 **         ** **  ***
               ****  *
             *  **   *
             ** *** *
                   *      **



       *                       **
       ***   *                 **
           ** *     **    *
                    **
  **          *    **  ***
   *  ***  *  *      * **
   ** **    *  *       **       **  *
  **   **     *                 *   **
  *  *   *            *             ***
 ** **               * *    ***
                     * *
Generation: 6
            * **   * *
            *     **  *
            **    ** **
             *     ***
             **** ***
                 ***



       *                       **
       **   **                 **
```

```
        *   **      **
             *        * **
  **   *           *   * *
     ** *    ***     *
    **      ***      ***       **  **
  *  * **              *        ** *  *
     ** *             *      *      * *
 *****               * *     *       *
                             *
Generation: 7
             *    ***
           ** *       *
            **        *
               * *    *
             ****    *
              ****  *
                  *


       **                      **
       **   **                 **
       **     *      *
            **      ******
      **     *         * *
   * ***     * *      *
    *   *    * *      ***       *** **
       **     *         *       ****  *
 *      *             **            * *
  *****               *     ***      *
  ***
Generation: 8
            **     *
           *  *    * *
           ****      ***
            *  *     **
             *       *
             *   *
               ***


       **                      **
      *  *   *                 **
       **     *     ** **
      * *   ***     ** * *
     * *     *           *
    **  *   **        *          *
    **  *    * *      * *       *   **
       ***    *         *       *     *
  ***** *             **     *   **** *
 *   *                **     *       *
  *                          *
Generation: 9
            **      *
           *  *      *
           *  **       *
           *        *  *
            ***      **
              *  *
                **
                *

       **                      **
      *  *                     **
      * **  * *     ** **
      * *   * *     ** * *
    ** **            ** *
       **   **         *
    ***     **                  **   *
         *    *       * *       * *   *
  ***** **            * *        **** *
 *   **                **    ***   ****

Generation: 10
            **
           *  **
          *** **      *
           *   *     * *
            ***      **
              ****
               ***
                **

       **                      **
      *  *                     **
     ** **           ** **
      *                  *
     *   *  *       **  *
        *   **         **
     ****   * *        *        **
       ***   *                  *   * *
  *** ****           ** *    *        *
  **  **              **     *   *   *
                             *     **
Game of Life simulation ended.
```

[11]:

```python
# CommaCode.py
def comma_code(input_list):
    if not input_list:
        return ""
    elif len(input_list) == 1:
        return str(input_list[0])
    else:
        first_part = ', '.join(map(str, input_list[:-1]))

        last_part = 'and ' + str(input_list[-1])
        return f"{first_part}, {last_part}"

print(' Name: Thanushri K S\n USN: 1AY24AI112\n Section: O')
spam = ['apples', 'bananas', 'tofu', 'cats']
print(comma_code(spam))

empty_list = []
print(comma_code(empty_list))

single_item_list = ['hello']
print(comma_code(single_item_list))

numbers_list = [1, 2, 3, 4]
print(comma_code(numbers_list))
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
apples, bananas, tofu, and cats

hello
1, 2, 3, and 4
```

[12]:

```python
# CoinFlipStreaks.py
import random

def coin_flip_streaks(num_flips):
    flips = []
    for _ in range(num_flips):
        if random.randint(0, 1) == 0:
            flips.append('T')
        else:
            flips.append('H')

    print("List of flips:", ' '.join(flips))

    streak_count = 0
    for i in range(len(flips) - 5):
        if (flips[i] == flips[i+1] == flips[i+2] == flips[i+3] == flips[i+4] ==␣
 ↪flips[i+5]):
            streak_count += 1

    return streak_count

if __name__ == "__main__":
    print(' Name: Thanushri K S\n USN: 1AY24AI112\n Section: O')
    number_of_flips = 6
    streaks = coin_flip_streaks(number_of_flips)
    print(f"\nNumber of flips: {number_of_flips}")
    print(f"Number of streaks of 6 consecutive heads or tails: {streaks}")
    num_simulations = 10
    total_streaks = 0
    for i in range(num_simulations):
        streaks = coin_flip_streaks(number_of_flips)
        total_streaks += streaks
        print(f"Simulation {i+1}: Streaks found = {streaks}")

    average_streaks = total_streaks / num_simulations
    print(f"\nAverage number of streaks over {num_simulations} simulations:␣
 ↪{average_streaks:.2f}")
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
List of flips: T H T H H H

Number of flips: 6
Number of streaks of 6 consecutive heads or tails: 0
List of flips: T T T H H T
Simulation 1: Streaks found = 0
List of flips: H T T T T T
Simulation 2: Streaks found = 0
List of flips: H T H H T T
Simulation 3: Streaks found = 0
List of flips: T H H T H H
Simulation 4: Streaks found = 0
List of flips: H H H H H H
Simulation 5: Streaks found = 1
List of flips: H T T T H T
Simulation 6: Streaks found = 0
List of flips: H T H H H H
Simulation 7: Streaks found = 0
List of flips: T T T H H H
Simulation 8: Streaks found = 0
```

```
List of flips: T H T H H T
Simulation 9: Streaks found = 0
List of flips: T H T H H H
Simulation 10: Streaks found = 0

Average number of streaks over 10 simulations: 0.10
```

[13]:

```python
#CharacterPictureGrid.py
def print_character_grid(grid):
    for row in grid:
        print(''.join(row))

if __name__ == "__main__":
    print(' Name: Thanushri K S \n USN: 1AY24AI112 \n Section: O')
    example_grid = [
        ['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['.', '.', 'O', 'O', 'O', 'O'],
        ['.', '.', '.', 'O', 'O', 'O']
    ]

    print("Example Grid:")
    print_character_grid(example_grid)

    custom_grid = [
        ['#', '#', '#'],
        ['#', ' ', '#'],
        ['#', '#', '#']
    ]

    print("\nCustom Grid:")
    print_character_grid(custom_grid)

    text_grid = [
        ['P', 'y', 't', 'h', 'o', 'n'],
        ['i', 's', ' ', 'f', 'u', 'n'],
        ['!', '!', '!', ' ', ':', ')']
    ]

    print("\nText Grid:")
    print_character_grid(text_grid)
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
```

```
Example Grid:
…
.OO…
OOOO..
OOOOO.
.OOOOO
..OOOO
…OOO

Custom Grid:
###
# #
###

Text Grid:
Python
is fun
!!! :)
```

[15]:

```python
# ChessDictionaryValidator.py
def is_valid_chess_board(board):
    valid_pieces = {
        'wpawn', 'wrook', 'wknight', 'wbishop', 'wqueen', 'wking',
        'bpawn', 'brook', 'bknight', 'bbishop', 'bqueen', 'bking'
    }
    valid_positions = set()
    for letter in 'abcdefgh':
        for number in '12345678':
            valid_positions.add(letter + number)

    piece_counts = {}
    for pos, piece in board.items():
        if pos not in valid_positions:
            print(f"Error: Invalid position '{pos}'.")
            return False
        if piece not in valid_pieces:
            print(f"Error: Invalid piece '{piece}' at '{pos}'.")
            return False

        piece_counts[piece] = piece_counts.get(piece, 0) + 1
    if piece_counts.get('wking', 0) != 1 or piece_counts.get('bking', 0) != 1:
        print("Error: There must be exactly one white king and one black king.")
        return False

    if piece_counts.get('wqueen', 0) > 1 or piece_counts.get('bqueen', 0) > 1:
        print("Error: There can be at most one white queen and one black queen␣
 ↪(initially).")
```

```
        return False

    if piece_counts.get('wrook', 0) > 2 or piece_counts.get('brook', 0) > 2:
        print("Error: There can be at most two white rooks and two black rooks␣
 ↪(initially).")
        return False

    if piece_counts.get('wknight', 0) > 2 or piece_counts.get('bknight', 0) > 2:
        print("Error: There can be at most two white knights and two black␣
 ↪knights (initially).")
        return False

    if piece_counts.get('wbishop', 0) > 2 or piece_counts.get('bbishop', 0) > 2:
        print("Error: There can be at most two white bishops and two black␣
 ↪bishops (initially).")
        return False

    if piece_counts.get('wpawn', 0) > 8 or piece_counts.get('bpawn', 0) > 8:
        print("Error: There can be at most eight white pawns and eight black␣
 ↪pawns.")
        return False
    if len(board) != len(set(board.keys())):
        print("Error: Multiple pieces on the same position.")
        return False

    return True

if __name__ == "__main__":
    print(' Name: Thanushri K S \n USN: 1AY24AI112\n Section: O')
    valid_board = {
        'a1': 'wrook', 'a2': 'wpawn', 'a3': ' ', 'a4': ' ', 'a5': ' ', 'a6': '␣
 ↪', 'a7': 'bpawn', 'a8': 'brook',
        'b1': 'wknight', 'b2': 'wpawn', 'b3': ' ', 'b4': ' ', 'b5': ' ', 'b6':␣
 ↪' ', 'b7': 'bpawn', 'b8': 'bknight',
        'c1': 'wbishop', 'c2': 'wpawn', 'c3': ' ', 'c4': ' ', 'c5': ' ', 'c6':␣
 ↪' ', 'c7': 'bpawn', 'c8': 'bbishop',
        'd1': 'wqueen', 'd2': 'wpawn', 'd3': ' ', 'd4': ' ', 'd5': ' ', 'd6': '␣
 ↪', 'd7': 'bpawn', 'd8': 'bqueen',
        'e1': 'wking', 'e2': 'wpawn', 'e3': ' ', 'e4': ' ', 'e5': ' ', 'e6': '␣
 ↪', 'e7': 'bpawn', 'e8': 'bking',
        'f1': 'wbishop', 'f2': 'wpawn', 'f3': ' ', 'f4': ' ', 'f5': ' ', 'f6':␣
 ↪' ', 'f7': 'bpawn', 'f8': 'bbishop',
        'g1': 'wknight', 'g2': 'wpawn', 'g3': ' ', 'g4': ' ', 'g5': ' ', 'g6':␣
 ↪' ', 'g7': 'bpawn', 'g8': 'bknight',
        'h1': 'wrook', 'h2': 'wpawn', 'h3': ' ', 'h4': ' ', 'h5': ' ', 'h6': '␣
 ↪', 'h7': 'bpawn', 'h8': 'brook'
```

```
    }
    print("Valid Board Check:", is_valid_chess_board(valid_board))

    invalid_position_board = {'a9': 'wpawn'}
    print("Invalid Position Check:",␣
 ↪is_valid_chess_board(invalid_position_board))

    invalid_piece_board = {'a1': 'wkingg'}
    print("Invalid Piece Check:", is_valid_chess_board(invalid_piece_board))

    multiple_kings_board = {'a1': 'wking', 'h8': 'bking', 'e5': 'wking'}
    print("Multiple Kings Check:", is_valid_chess_board(multiple_kings_board))

    too_many_pawns_board = {f'{chr(ord("a") + i)}2': 'wpawn' for i in range(9)}
    too_many_pawns_board['a1'] = 'wking'
    too_many_pawns_board['h8'] = 'bking'
    print("Too Many Pawns Check:", is_valid_chess_board(too_many_pawns_board))

    occupied_position_board = {'a1': 'wrook', 'a1': 'wpawn'}
    print("Occupied Position Check:",␣
 ↪is_valid_chess_board(occupied_position_board))
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
Error: Invalid piece ' ' at 'a3'.
Valid Board Check: False
Error: Invalid position 'a9'.
Invalid Position Check: False
Error: Invalid piece 'wkingg' at 'a1'.
Invalid Piece Check: False
Error: There must be exactly one white king and one black king.
Multiple Kings Check: False
Error: Invalid position 'i2'.
Too Many Pawns Check: False
Error: There must be exactly one white king and one black king.
Occupied Position Check: False
```

[16]:

```
# FantasyGameInventory.py
def display_inventory(inventory):
    print("Inventory:")
    total_items = 0
    for item, count in inventory.items():
        print(f"{count} {item}")
        total_items += count
    print(f"Total number of items: {total_items}")
```

```
def add_to_inventory(inventory, added_items):
    for item in added_items:
        inventory[item] = inventory.get(item, 0) + 1
    return inventory

if __name__ == "__main__":
    print(' Name: Thanushri K S \n USN: 1AY24AI112 \n Section: O')
    player_inventory = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1,␣
 ↪'arrow': 12}
    display_inventory(player_inventory)

    dragon_loot = ['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']
    print("\nYou found the following loot:")
    print(dragon_loot)

    player_inventory = add_to_inventory(player_inventory, dragon_loot)
    print("\nUpdated inventory:")
    display_inventory(player_inventory)
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
Inventory:
1 rope
6 torch
42 gold coin
1 dagger
12 arrow
Total number of items: 62

You found the following loot:
['gold coin', 'dagger', 'gold coin', 'gold coin', 'ruby']

Updated inventory:
Inventory:
1 rope
6 torch
45 gold coin
2 dagger
12 arrow
1 ruby
Total number of items: 67
```

[17]:

```
#TablePrinter.py
print(' Name: Thanushri K S \n USN: 1AY24AI112 \n Section: O\n')
row=int(input("Enter a number:"))
col=int(input("Enter a number:"))
```

```
for i in  range (1,row+1):
    print('Table for ',+i)
    for j in range(1,col+1):
        k=i*j
        print(str(i),'*',str(j),str('='),str(k))
    print('\n')
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O
```

```
Enter a number: 6
Enter a number: 4
```

```
Table for  1
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4


Table for  2
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8


Table for  3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12


Table for  4
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16


Table for  5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
```

```
Table for  6
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
```

[18]:

```python
#ZombieDiceBots.py
import random

class ZombieDiceBot:

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

    def should_roll(self, brain_count, shotguns_count, turn_rolls_history):
        raise NotImplementedError("Subclasses must implement the should_roll␣
 ↪method.")

    def __str__(self):
        return self.name

class BasicBot(ZombieDiceBot):
    def should_roll(self, brain_count, shotguns_count, turn_rolls_history):
        return brain_count < 1

class RiskyBot(ZombieDiceBot):
    def should_roll(self, brain_count, shotguns_count, turn_rolls_history):
        return shotguns_count < 3

class CautiousBot(ZombieDiceBot):
    def should_roll(self, brain_count, shotguns_count, turn_rolls_history):
        return brain_count < 2

class RandomBot(ZombieDiceBot):
    def should_roll(self, brain_count, shotguns_count, turn_rolls_history):
        return random.choice([True, False])

class BrainGreedyBot(ZombieDiceBot):
    def should_roll(self, brain_count, shotguns_count, turn_rolls_history):
        return shotguns_count < 3

def roll_dice():
    dice_colors = ['green'] * 6 + ['yellow'] * 4 + ['red'] * 3
    rolled_dice = random.sample(dice_colors, 3)
    results = []
    for color in rolled_dice:
        if color == 'green':
            outcomes = ['brain'] * 3 + ['shotgun'] * 1 + ['runner'] * 2
        elif color == 'yellow':
            outcomes = ['brain'] * 2 + ['shotgun'] * 2 + ['runner'] * 2
        else:  # red
            outcomes = ['brain'] * 1 + ['shotgun'] * 3 + ['runner'] * 2
        results.append(random.choice(outcomes))
    return tuple(results)

def play_turn(bot):
    print(f"\n--- {bot.name}'s turn ---")
    brains_this_turn = 0
    shotguns_this_turn = 0
    turn_rolls_history = []

    while shotguns_this_turn < 3 and bot.should_roll(brains_this_turn,␣
 ↪shotguns_this_turn, turn_rolls_history):
        input(f"{bot.name} decides to roll. Press Enter to roll...")
        roll_result = roll_dice()
        turn_rolls_history.append(roll_result)
        print(f"{bot.name} rolled: {', '.join(roll_result)}")

        for result in roll_result:
            if result == 'brain':
                brains_this_turn += 1
            elif result == 'shotgun':
                shotguns_this_turn += 1

        print(f"Brains this turn: {brains_this_turn}")
        print(f"Shotguns this turn: {shotguns_this_turn}")

        if shotguns_this_turn >= 3:
            print(f"{bot.name} got zombied out!")
            return 0

    print(f"{bot.name} decided to stop. Total brains this turn:␣
 ↪{brains_this_turn}")
    return brains_this_turn

def run_game(bots, num_turns=5):
    scores = {bot.name: 0 for bot in bots}

    for turn in range(1, num_turns + 1):
        for bot in bots:
            brains_earned = play_turn(bot)
            scores[bot.name] += brains_earned
            print(f"{bot.name}'s total score: {scores[bot.name]}")
        print(f"\n--- End of Turn {turn} ---")
        print("Current Scores:")
        for name, score in scores.items():
            print(f"{name}: {score}")
        break

    print("\n--- Game Over ---")
    print("Final Scores:")
    for name, score in scores.items():
        print(f"{name}: {score}")

if __name__ == "__main__":
    print(' Name: Thanushri K S \n USN: 1AY24AI112\n Section: O')
    bot1 = BasicBot("Basic Bot")
    bot2 = RiskyBot("Risky Bot")
    players = [bot1, bot2]
    run_game(players, num_turns=3)
```

```
 Name: Thanushri K S
 USN: 1AY24AI112
 Section: O

--- Basic Bot's turn ---

Basic Bot decides to roll. Press Enter to roll…

Basic Bot rolled: brain, shotgun, brain
Brains this turn: 2
Shotguns this turn: 1
Basic Bot decided to stop. Total brains this turn: 2
Basic Bot's total score: 2

--- Risky Bot's turn ---

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: runner, runner, shotgun
Brains this turn: 0
Shotguns this turn: 1

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: runner, brain, brain
Brains this turn: 2
Shotguns this turn: 1

Risky Bot decides to roll. Press Enter to roll…
Risky Bot rolled: brain, runner, runner
Brains this turn: 3
Shotguns this turn: 1

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: brain, runner, brain
Brains this turn: 5
Shotguns this turn: 1

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: brain, brain, runner
Brains this turn: 7
Shotguns this turn: 1

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: brain, runner, brain
Brains this turn: 9
Shotguns this turn: 1

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: runner, shotgun, brain
Brains this turn: 10
Shotguns this turn: 2

Risky Bot decides to roll. Press Enter to roll…

Risky Bot rolled: shotgun, runner, shotgun
Brains this turn: 10
Shotguns this turn: 4
Risky Bot got zombied out!
Risky Bot's total score: 0

--- End of Turn 1 ---
Current Scores:
Basic Bot: 2
Risky Bot: 0

--- Game Over ---
Final Scores:
Basic Bot: 2
Risky Bot: 0
```

[ ]: