UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

MINOR PROJECT

(Subway Surfers Game)

Student Name: TRISHA **UID: 22BCA10681**
Branch: UIC -BCA **Section/Group:22BCA 3-B**
Semester: 5 **Date of Performance:25/10/2024**
Subject Name: WEB DEVELOPMENT **Subject Code:22CAH 301**

1. AIM/OVERVIEW OF THE PRACTICAL:

The objective of this project is to develop a classic **Subway Surfers Game** using **C#** in a console-based environment. The game allows the player to **navigate them to get collide from X** within a specified grid, which **increases the player's score.** The game **ends** when**the player collides with the boundary walls** or **X**.

2. TASK TO BE DONE:

- **Set Up Game Environment:** Define the game area dimensions and initialize variables for player position, score, and **X** position.
- **Implement Controls:** Set up input detection for arrow keys, allowing the player to change the player's direction without reversing it.
- **Score Generation:** Constantly coming X within the game area, ensuringthey don't overlap with the snake's current position.
- **Collision Detection:** Implement logic to end the game upon collision with walls or the player's own body.
- **Console Rendering:** Continuously update the console display to reflect the player's movement, X position, and current score.

3. ALGORITHM/FLOWCHART:

1. Initialize Game:

- Set initial frog length and position at the center of the game area.
- Generate the first piece of food at a random position.
- Set the initial score to zero and define the game speed.

2. Game Loop:

- Check for player input to update the frog's direction.
- Update the frog's position based on the current direction.
- Food Collision: If the snake reaches the food, increase its length, increment the score, and generate a new piece of food.
- Collision Detection: Check for wall or self-collisions; if detected, end the game.
- Render Game: Display the current game state in the console, including walls, snake, food, and score.
- Speed Control: Use a delay to control game speed and make it more challenging.

3. End Game:

- When a collision is detected, display "Game Over" and show the final score, giving the player an option to exit the game.

4. DATASET:

There is no predefined dataset for this project. The program dynamically collects data from the user, such as expense names and amounts, during execution.

5. CODE FOR EXPERIMENT/PRACTICAL:

```
using System;
using System.Threading;

namespace ConsoleSubwaySurfer
{
   class Program
   {
      static int playerPosition = 1;       // 0 = left, 1 = middle, 2 = right
      static int score = 0;                // Player's score
      static bool gameRunning = true;      // Game status
      static int laneWidth = 5;            // Width of each lane in console characters
      static int[] obstaclePositions = new int[3]; // Array to hold obstacle positions
for each lane

      static void Main(string[] args)
      {
         Console.CursorVisible = false;
         Console.Clear();
         InitializeGame();

         // Start the game loop in a separate thread

         Thread gameThread = new Thread(GameLoop);
         gameThread.Start();

         // Listen for player input

         while (gameRunning)
         {
            if (Console.KeyAvailable)
            {
               ConsoleKeyInfo key = Console.ReadKey(true);
               if (key.Key == ConsoleKey.LeftArrow && playerPosition > 0)
               {
                  playerPosition--; // Move left
```

```
            }
            else if (key.Key == ConsoleKey.RightArrow && playerPosition < 2)
            {
                playerPosition++; // Move right
            }
        }
    }
}
// Initialize the game display and setup

static void InitializeGame()
{
    Console.SetCursorPosition(0, 0);
    Console.Write("Score: 0");

    // Draw lane boundaries

    for (int i = 1; i < Console.WindowHeight; i++)
    {
        Console.SetCursorPosition(laneWidth, i);
        Console.Write("|"); // Left lane boundary
        Console.SetCursorPosition(laneWidth * 2, i);
        Console.Write("|"); // Right lane boundary
    }
}

// Main game loop to update obstacles and score

static void GameLoop()
{
    Random random = new Random();

    while (gameRunning)
    {
        // Move obstacles down and check for collision

        UpdateObstacles();
```

```
        // Check for collision with the player

        if (obstaclePositions[playerPosition] == Console.WindowHeight - 2)
        {
            gameRunning = false;
            Console.Clear();
            Console.WriteLine("Game Over! You hit an obstacle.");
            Console.WriteLine($"Your score: {score}");
            return;
        }

        // Add a new obstacle in a random lane

        if (random.Next(0, 5) == 0)  // Random chance for obstacle appearance
        {
            int lane = random.Next(0, 3);
            if (obstaclePositions[lane] == 0) // Only place if no obstacle is in the lane
            {
                obstaclePositions[lane] = 1; // Place obstacle at the top of the screen
            }
        }
        // Update the score
        score++;
        UpdateScore();

        // Redraw player in current lane
        DrawPlayer();

        Thread.Sleep(150); // Game speed, decrease to make it faster
    }
}

// Update score display at the top

static void UpdateScore()
{
```

```
    Console.SetCursorPosition(7, 0);  // Position to display the score
    Console.Write(score);
}

// Move all obstacles down by one row

static void UpdateObstacles()
{
    for (int i = 0; i < 3; i++)
    {
        if (obstaclePositions[i] >= 1)
        {
            // Erase obstacle at previous position

            Console.SetCursorPosition(i * laneWidth + laneWidth / 2,
obstaclePositions[i]);
            Console.Write(" ");

            // Move obstacle down
            obstaclePositions[i]++;

            // Redraw obstacle at new position

            if (obstaclePositions[i] < Console.WindowHeight - 1)
            {
                Console.SetCursorPosition(i * laneWidth + laneWidth / 2,
obstaclePositions[i]);
                Console.Write("X"); // Obstacle symbol
            }
            else
            {
                // Clear obstacle if it moves off the screen

                obstaclePositions[i] = 0;
            }
        }
    }
```

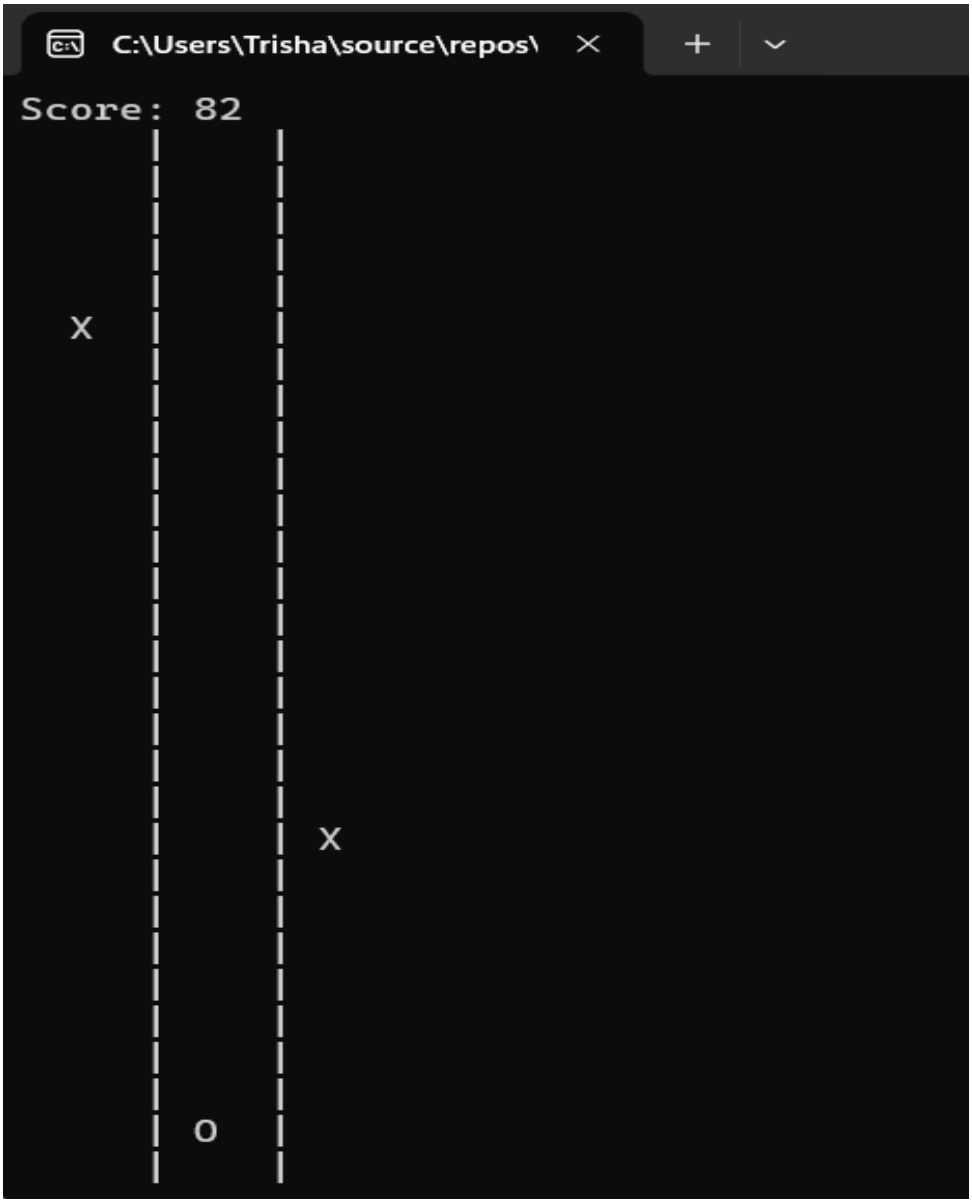
```
}
      // Draw the player in their current lane

      static void DrawPlayer()
      {
         // Clear previous player position
         Console.SetCursorPosition(0 * laneWidth + laneWidth / 2,
Console.WindowHeight - 2);
         Console.Write(" ");
         Console.SetCursorPosition(1 * laneWidth + laneWidth / 2,
Console.WindowHeight - 2);
         Console.Write(" ");
         Console.SetCursorPosition(2 * laneWidth + laneWidth / 2,
Console.WindowHeight - 2);
         Console.Write(" ");

         // Draw player in current lane

         Console.SetCursorPosition(playerPosition * laneWidth + laneWidth / 2,
Console.WindowHeight - 2);
         Console.Write("O");  // Player symbol
      }
   }
```

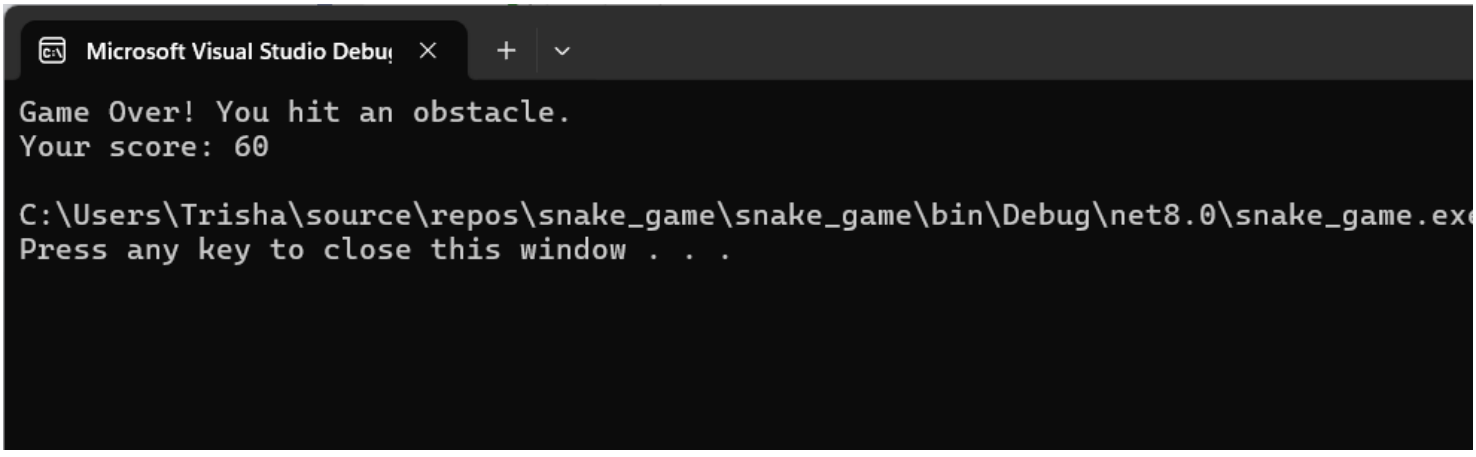
Program.cs

snake_game | ConsoleSubwaySurfer.Program | playerPosition

```
using System;
using System.Threading;

namespace ConsoleSubwaySurfer
{
    0 references
    class Program
    {
        static int playerPosition = 1;        // 0 = left, 1 = middle, 2 = right
        static int score = 0;                 // Player's score
        static bool gameRunning = true;       // Game status
        static int laneWidth = 5;             // Width of each lane in console character
        static int[] obstaclePositions = new int[3]; // Array to hold obstacle position

        0 references
        static void Main(string[] args)
        {
            Console.CursorVisible = false;
            Console.Clear();
            InitializeGame();

            // Start the game loop in a separate thread
            Thread gameThread = new Thread(GameLoop);
            gameThread.Start();

            // Listen for player input
            while (gameRunning)
```

100 % | No issues found | Ln: 1 | Ch: 14 | SPC | LF


C:\Users\Trisha\source\repos\
Score: 82
X
X
O


Microsoft Visual Studio Debu
Game Over! You hit an obstacle.
Your score: 60
C:\Users\Trisha\source\repos\snake_game\snake_game\bin\Debug\net8.0\snake_game.exe
Press any key to close this window . . .

6. LEARNING OUTCOMES (WHAT I HAVE LEARNT):

1. **Advanced C# Fundamentals:** Gained hands-on experience with advanced C# concepts such as multi-threading (for game loop control), lists, and random number generation.

2. **Game Development Principles:** Developed an understanding of game loops, state management, and real-time user interaction, essential skills in game programming.

3. **Collision Detection and Boundary Handling:** Learned how to implement collision detection logic to ensure the frog doesn't pass beyond boundaries or overlap with itself.

4. **Console-based Graphics Rendering:** Mastered the art of rendering a visual representation of the game in the console, including walls, player as 0, and collision from X.

5. **Problem-solving and Logic Building:** Improved logical thinking and problem-solving skills through planning and structuring game mechanics.

7. CONCLUSION

This a frog Game project was a highly rewarding experience, combining elements of game logic, user interaction, and console-based graphics. This project not only improved my understanding of C# programming but also provided an introduction to game development principles. By implementing various mechanics like collision detection, input handling, and real-time score tracking, I gained valuable experience in logic building, debugging, and testing. This project has laid a strong foundation for more advanced game development and programming projects in the future.

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance (Pre Lab Quiz)		5
2.	Worksheet		10
3.	Post Lab Quiz		5