UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

MINOR PROJECT

(Blockade Game)

Student Name: RASHMI KUMARI **UID: 22BCA10226**
Branch: UIC -BCA **Section/Group:22BCA 3-A**
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 **Blockade Game** using **C#** in a console-based environment. The game allows the player to **navigate a frog to eat** randomly placed food items within a specified grid, which **increases both the frog's and the player's score.** The game **ends** when **the frog collides with the boundary walls** or itself.

2. TASK TO BE DONE:

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

3. ALGORITHM/FLOWCHART:

1. Initialize Game:

- Set initial frog length and position at the centre 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;
namespace SnakeGame
{
   internal class Program
   {
      static void Main(string[] args)
      {
         Random rand = new Random();
         Console.CursorVisible = false;
         bool ShouldExit = false;
         // Snake
         string snake = "-<===|:D-";
         int snakeX = 0;
         int snakeY = 0;
         int frogEat = 0;
         // Frog
         string frog = "^(..)^";
         int frogX = 0;
         int frogY = 0;
         // Canvas dimensions
         int height = Console.WindowHeight;
```

```
int width = Console.WindowWidth - snake.Length;
// Start the game
StartGame();
while (!ShouldExit)
{
   if (TerminalResized())
   {
      Console.Clear();
      Console.WriteLine("Canvas size changed!!\nGAME
END!!");
      ShouldExit = true;
   }
   else
   {
      Move();

      if (GotFrog())
      {
         frogEat++;
         Console.SetCursorPosition(width - 2, 3);
         Console.Write($"FROG EAT:{frogEat}");
         ShowLevel();
```

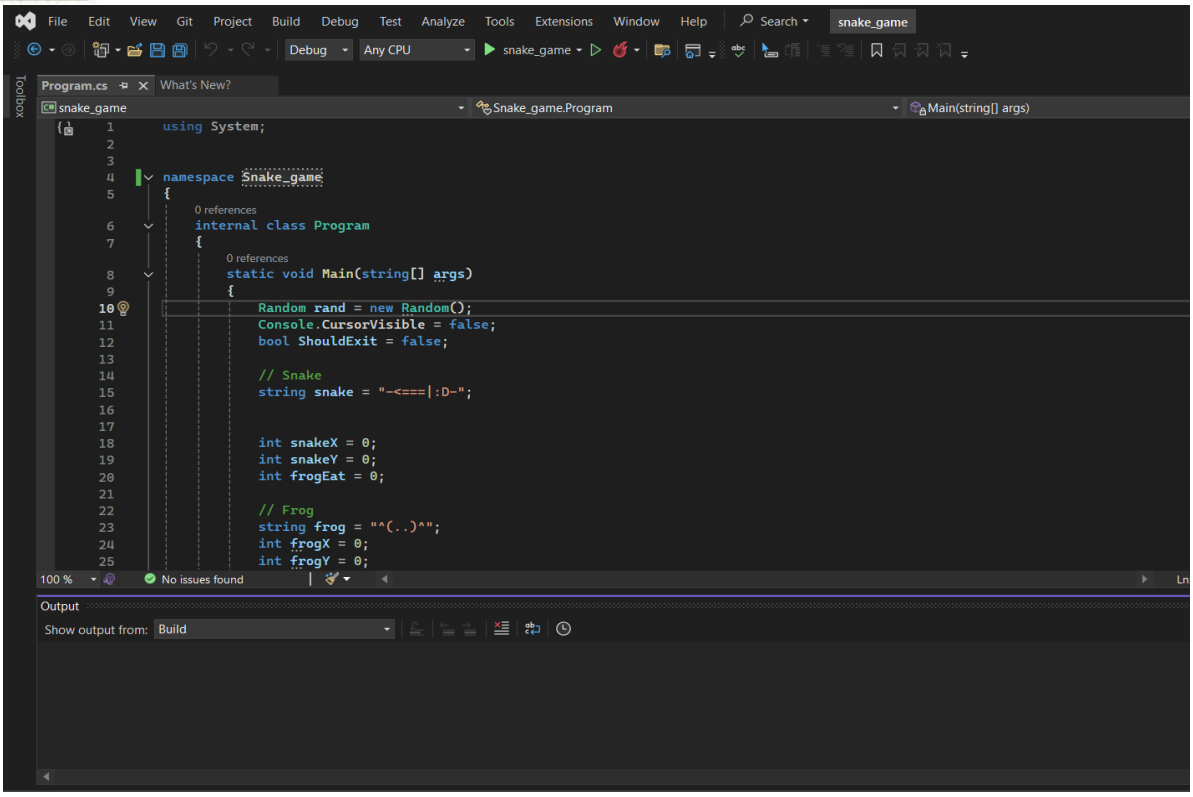
```
            ShowFood();
        }
    }
}
//GotFrog
bool GotFrog()
{
    return snakeX == frogX && snakeY == frogY;
}
// Function to handle snake movement
void Move()
{
    int lastSnakeX = snakeX;
    int lastSnakeY = snakeY;

    // Handle user input
    switch (Console.ReadKey(true).Key)
    {
        case ConsoleKey.UpArrow:
            snakeY--;
            break;
        case ConsoleKey.DownArrow:
```

```
            snakeY++;
            break;
        case ConsoleKey.LeftArrow:
            snakeX--;
            break;
        case ConsoleKey.RightArrow:
            snake = "-<===|:D-";
            snakeX++;
            break;
        default:
            Console.Clear();
            Console.SetCursorPosition(width / 2, height / 2);
            Console.Write("!! GAME END !!");
            ShouldExit = true;
            break;
    }

    // Clear the previous snake position
    Console.SetCursorPosition(lastSnakeX, lastSnakeY);
    for (int i = 0; i < snake.Length; i++)
    {
        Console.Write(" ");
```

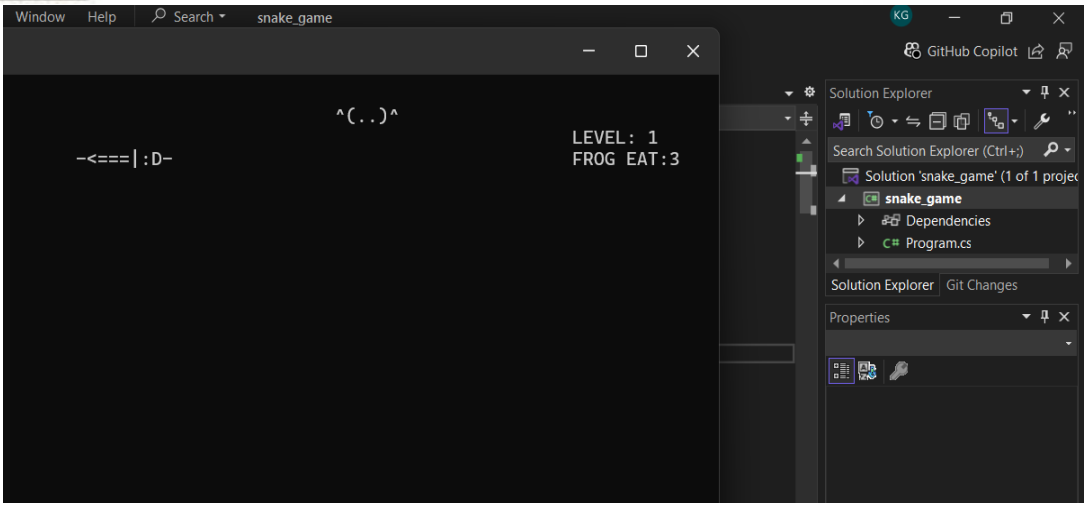
```
            }
            // Ensure snake stays within boundaries
            if (snakeX < 0 || snakeX > width || snakeY < 0 || snakeY > height)
            {
                Console.Clear();
                Console.SetCursorPosition(width / 2, height / 2);
                Console.Write("GAME OVER!!");
                ShouldExit = true;
                return;
            }
            // Draw the snake at the new location
            Console.SetCursorPosition(snakeX, snakeY);
            Console.Write(snake);
        }
        // Returns true if the terminal was resized
        bool TerminalResized()
        {
            return height != Console.WindowHeight || width !=
Console.WindowWidth - snake.Length;
        }
        // Display the frog (food)
        void ShowFood()
```

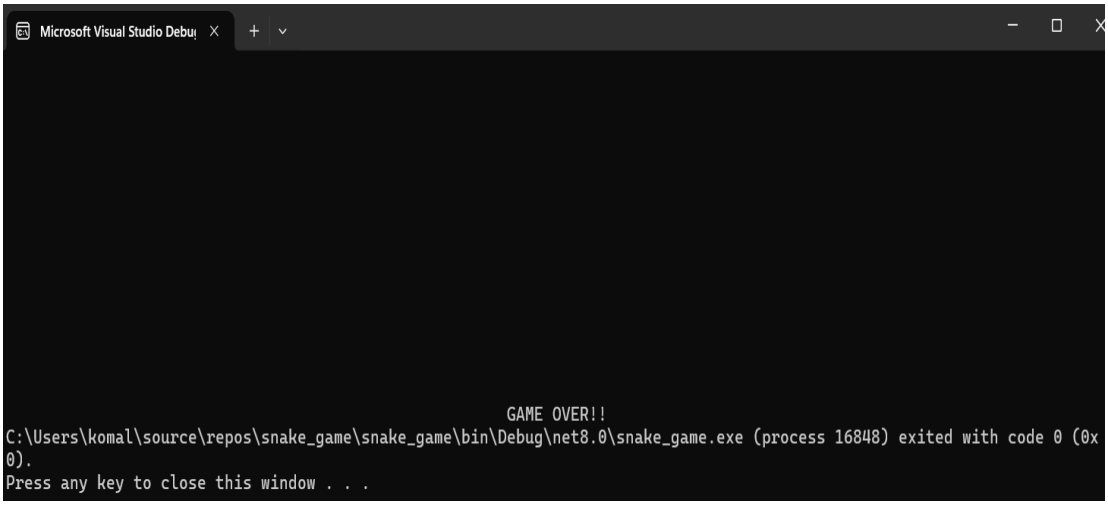
```
{
    frogX = rand.Next(0, width);
    frogY = rand.Next(0, height);

    // Display the frog at a random position
    Console.SetCursorPosition(frogX, frogY);
    Console.Write(frog);
}
void ShowLevel()
{
    Console.SetCursorPosition(width - 2, 2);
    if(frogEat >= 0 && frogEat < 5)
    {
        Console.Write($"LEVEL: 1");
        Console.SetCursorPosition(width - 2, 2);


    }
    else if (frogEat >= 5 && frogEat < 10)
    {
        Console.Write($"LEVEL: 2");
        Console.SetCursorPosition(width - 2, 2);
```

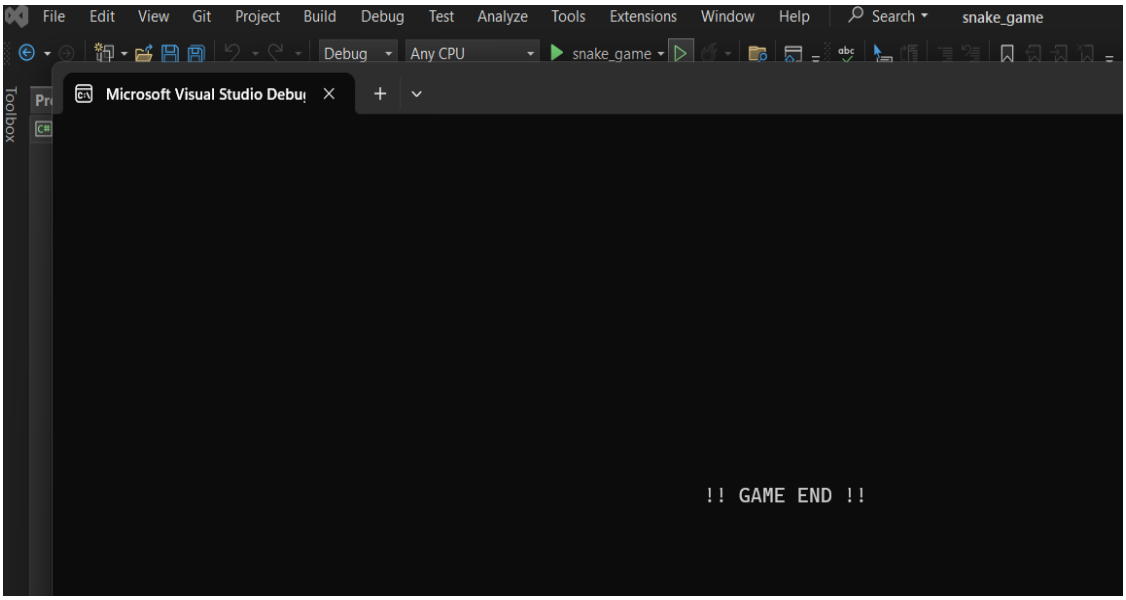
```
            }
            else if (frogEat >= 10 && frogEat < 15)
            {
                Console.Write($"LEVEL: 3");
                Console.SetCursorPosition(width - 2, 2);
            }
        }
        // Start the game
        void StartGame()
        {
            Console.Clear();
            ShowFood();
            ShowLevel();
            Console.SetCursorPosition(0, 0);
            Console.Write(snake);
        }
    }
  }
}
```




```
using System;


namespace Snake_game
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            Random rand = new Random();
            Console.CursorVisible = false;
            bool ShouldExit = false;

            // Snake
            string snake = "-<===|:D-";


            int snakeX = 0;
            int snakeY = 0;
            int frogEat = 0;

            // Frog
            string frog = "^(..)^";
            int frogX = 0;
            int frogY = 0;
```


Window
Help
Search
snake_game
GitHub Copilot
^(..)^
LEVEL: 1
FROG EAT:3
-<===|:D-
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'snake_game' (1 of 1 projec
snake_game
Dependencies
Program.cs
Solution Explorer
Git Changes
Properties


Microsoft Visual Studio Debu
GAME OVER!!
C:\Users\komal\source\repos\snake_game\snake_game\bin\Debug\net8.0\snake_game.exe (process 16848) exited with code 0 (0x0).
Press any key to close this window . . .


File
Edit
View
Git
Project
Build
Debug
Test
Analyze
Tools
Extensions
Window
Help
Search
snake_game
Debug
Any CPU
snake_game
Microsoft Visual Studio Debu
!! GAME END !!

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, frog body, and food placement.

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