VISUAL PROGRAMMING

Submitted By: **Esha Qaiser (01-131222-014)**
Fatima Nasir (01-131222-043)

Submitted To: **Mam Subhas Bilal**

Section: **BSE-5A**

DEPARTMENT OF SOFTWARE ENGINEERING

BAHRIA UNIVERSITY
ISLAMABAD CAMPUS

PROJECT-REPORT

Snake Game

Project Title:

The title of our project is **“Snake Game”**.

Scope of the Software:

The Snake Game aims to provide an interactive and visually appealing implementation of the classic Snake arcade game. The software allows players to control a snake on a grid, guiding it to consume food, grow longer, and avoid collisions with walls and its own body. The game provides an engaging user experience with features such as a dynamic grid, snake movement, his steps scoring, storing highest score, and a game over sequence. We have connected it to database which stores the player’s name and score.

Functional Requirements of the Project:

1. Game Initialization and Player Input

The game should prompt the player to enter their name before starting. The entered name should be stored for tracking the player's progress.

2. Countdown Before Game Start

We have displayed a countdown overlay or message to indicate that the game is about to start. Countdown should be visible for a few seconds before the game begins.

3. Database Connectivity

The game is connected to a database to store and retrieve player data. Player data includes the player's name, score and moves.

4. Player Data Display

It displays the player's name, score, and moves during gameplay.

5. High Scores

It provides an option to display the high scores from the database. High scores should include player names, scores, and moves.

6. User Interface Enhancements

We have designed a visually appealing user interface to incorporate player input, countdown, and player data display.

Screenshots:

Direction.cs:

```
using System;
using System.Collections.Generic;

namespace Snake
{
   public class Direction
   {
      public readonly static Direction Left = new Direction(0, -1);
      public readonly static Direction Right = new Direction(0, 1);
      public readonly static Direction Up = new Direction(-1, 0);
      public readonly static Direction Down = new Direction(1, 0);

      public int RowOffset { get; }        public int ColumnOffset
{ get; }

      private Direction (int rowOffset , int columnOffset)
      {
         RowOffset = rowOffset;
         ColumnOffset = columnOffset;
      }
      public Direction Opposite()
      {
         return new Direction (-RowOffset, -ColumnOffset);
      }
      public override bool Equals(object obj)
      {
         return obj is Direction direction &&  RowOffset == direction.RowOffset &&
               ColumnOffset == direction.ColumnOffset;
      }
      public override int GetHashCode()
      {
         return HashCode.Combine(RowOffset, ColumnOffset);
      }
      public static bool operator ==(Direction left, Direction right)
      {

 return EqualityComparer<Direction>.Default.Equals(left, right);
      }
      public static bool operator !=(Direction left, Direction right)
      {
         return !(left == right);
      }
   }
}
```

GameState.cs:

```
using Microsoft.EntityFrameworkCore;
using Snake.Model;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Snake
{
   public class GameState
   {
      private string playerName;

      public int Rows { get; }
public int Columns { get; }
public GridValue[,] Grid { get; }
public Direction Dir { get; private set; }
public int Score { get; private set; }
public bool GameOver { get; private set; }
public string PlayerName { get; }

      private readonly LinkedList<Direction> dirChanges = new LinkedList<Direction>();
      private readonly LinkedList<Position> snakePositions = new LinkedList<Position>();
      private readonly Random random = new Random();
      private Position head;
      private int turnsSinceLastFood;

      public GameState(int rows, int cols, string playerName)
      {
         Rows = rows;
         Columns = cols;
         Grid = new GridValue[Rows, Columns];
         Dir = Direction.Right;

         AddSnake();
         AddFood();
         head = new Position(0, 0);  // Initialize head position (example, replace with actual initialization)
         turnsSinceLastFood = 0;
        this.PlayerName = playerName;
      }
      private void AddSnake()
      {
         int r = Rows / 2;

         for (int c = 1; c <= 3; c++)
         {
            Grid[r, c] = GridValue.Snake;
            snakePositions.AddFirst(new Position(r, c));
         }
      }
      private IEnumerable<Position> EmptyPositions()
      {
         for (int r = 0; r < Rows; r++)
         {
            for (int c = 0; c < Columns; c++)
            {
```

```
            if (Grid[r, c] == GridValue.Empty)
            {
                yield return new Position(r, c);
            }
        }
    }
}
private void AddFood()
{
    List<Position> empty = new List<Position>(EmptyPositions());

    if (empty.Count == 0)
    {               return;
    }

    Position pos = empty[random.Next(empty.Count)];
    Grid[pos.Row, pos.Column] = GridValue.Food;
}
public Position HeadPosition()
{
    return snakePositions.First.Value;
}
public Position TailPoition()
{
    return snakePositions.Last.Value;
}
public IEnumerable<Position> SnakePositions()
{
    return snakePositions;
}
private void AddHead(Position pos)
{
    snakePositions.AddFirst(pos);
    Grid[pos.Row, pos.Column] = GridValue.Snake;
}
private void RemoveTail()
{
    Position tail = snakePositions.Last.Value;
   Grid[tail.Row, tail.Column] = GridValue.Empty;
   snakePositions.RemoveLast();
}
private Direction GetLastDirection()
{
    if (dirChanges.Count == 0)
    {
        return Dir;
    }
    return dirChanges.Last.Value;
}
private bool CanChangeDirection(Direction newDir)
{
    if (dirChanges.Count == 2)
    {
        return false;
    }
    Direction lastDir = GetLastDirection();
    return newDir != lastDir && newDir != lastDir.Opposite();         }
public void ChangeDirection(Direction dir)
{
```

```
            if (CanChangeDirection(dir))
            {
                dirChanges.AddLast(dir);
            }
        }
        private bool OutsideGrid(Position pos)
        {
            return pos.Row < 0 || pos.Row >= Rows || pos.Column < 0 || pos.Column >= Columns;
        }
        private GridValue WillHit(Position newHeadPos)
        {
            if (OutsideGrid(newHeadPos))
            {
                return GridValue.Outside;
            }

            if (newHeadPos == TailPoition())
            {
                return GridValue.Empty;
            }
            return Grid[newHeadPos.Row, newHeadPos.Column];
        }
        public void Move()
        {
            if (dirChanges.Count > 0)
            {
                Dir = dirChanges.First.Value;
               dirChanges.RemoveFirst();
            }

            Position newHeadPos = HeadPosition().Translate(Dir);
            GridValue hit = WillHit(newHeadPos);

            if (hit == GridValue.Outside || hit == GridValue.Snake)
            {
                GameOver = true;
            }
            else if (hit == GridValue.Empty)
            {
                RemoveTail();                AddHead(newHeadPos);
                head = newHeadPos;  // Update the head position
            }
            else if (hit == GridValue.Food)
            {
                AddHead(newHeadPos);
                Score++;                AddFood();
                head = newHeadPos;  // Update the head position
            }

            if (Grid[head.Row, head.Column] == GridValue.Food)
            {
                // Increment score and reset turnsSinceLastFood
                Score++;
                turnsSinceLastFood = 0;
                // ... (existing code)
            }
else
           {
                turnsSinceLastFood++;  // Increment turns since last food
```

```
        }
    }
    public int TurnsSinceLastFood
    {
        get { return turnsSinceLastFood; }
    }

    public void EndGame()
    {
        var options = new DbContextOptionsBuilder<SnakeDbContext>()
            .UseSqlServer("Server=DESKTOP-
BE80RDB\\SQLEXPRESS;Database=SnakeGameDB;Trusted_Connection=True;")
            .Options;

        using (var context = new SnakeDbContext(options))
        {
            var player = new Player
            {
                Name = playerName, // Use the playerName obtained in MainWindow
                Score = Score,
                Turns = TurnsSinceLastFood,
                HighestScore = GetHighestScore(playerName)
            };

            context.Players.Add(player);
            context.SaveChanges();
        }
    }

    private int GetHighestScore(string playerName)
    {
        var options = new DbContextOptionsBuilder<SnakeDbContext>()
            .UseSqlServer("Server=DESKTOP-
BE80RDB\\SQLEXPRESS;Database=SnakeGameDB;Trusted_Connection=True;")
            .Options;

        using (var context = new SnakeDbContext(options))
        {
            var player = context.Players.FirstOrDefault(p => p.Name == playerName);
            return player?.HighestScore ?? 0;
        }
    }

  }
}
```

Images.cs:

```
using System;
using System.Windows.Media; using
System.Windows.Media.Imaging;

namespace Snake
{
  public static class Images
  {
```

```
        public readonly static ImageSource Empty = LoadImage("Empty.png");
        public readonly static ImageSource Body = LoadImage("Body.png");
        public readonly static ImageSource Head = LoadImage("Head.png");
        public readonly static ImageSource Food = LoadImage("Food.png");
        public readonly static ImageSource DeadBody = LoadImage("DeadBody.png");
        public readonly static ImageSource DeadHead = LoadImage("DeadHead.png");
        private static ImageSource LoadImage(string fileName)
        {
            return new BitmapImage(new Uri($"Assets/{fileName}",UriKind.Relative));
        }
    }
}
```

MainWindow.XML.cs:

```
using System;
using System.Collections.Generic; using
System.Linq; using System.Text; using
System.Threading.Tasks; using System.Windows;
using System.Windows.Controls; using
System.Windows.Data; using
System.Windows.Documents; using
System.Windows.Input; using
System.Windows.Media; using
System.Windows.Media.Imaging; using
System.Windows.Navigation; using
System.Windows.Shapes;

namespace Snake
{
    // Interaction logic for MainWindow.xaml
    public partial class MainWindow : Window
    {
        private readonly Dictionary<GridValue, ImageSource> gridValToImage = new()
    {
            {GridValue.Empty, Images.Empty },
            {GridValue.Snake, Images.Body },
            {GridValue.Food, Images.Food }

        };
        private readonly Dictionary<Direction, int> dirToRotation = new()
        {
            {Direction.Up, 0},
            {Direction.Right, 90},
            {Direction.Down, 180},
            {Direction.Left, 270 }
        };
        private readonly int rows= 15, cols=15;
        private readonly Image[,] gridImages;
        private GameState gameState;
        private bool gameRunning;
        private string playerName;
        private readonly SnakeDbContext dbContext;
        public MainWindow()
        {
            InitializeComponent();            playerName =
GetPlayerName();            gridImages = SetupGrid();
```

```
            gameState = new GameState(rows, cols, playerName);
        }
        private async Task RunGame()

        {

            Draw();
            await ShowCountDown();
            Overlay.Visibility = Visibility.Hidden;            await GameLoop();
await ShowGameOver();
            gameState = new GameState(rows, cols, playerName);

        }
        private string GetPlayerName()
        {
            // Show a dialog or prompt to get the player's name
           // For simplicity, a MessageBox is used in this example
            return Microsoft.VisualBasic.Interaction.InputBox("Enter your name:",
"Player Name", "");
        }
        private async void Window_PreviewKeyDown(object sender, KeyEventArgs e)
        {
            if(Overlay.Visibility == Visibility.Visible)
            {
                e.Handled = true;
             }
            if(!gameRunning)
            {
                gameRunning = true;
                await RunGame();
                gameRunning = false;
            }

        }
        private void Window_KeyDown(object sender, KeyEventArgs e)
        {
            if(gameState.GameOver)
            {
                return;
            }

        switch (e.Key)
{
case Key.Left:
                    gameState.ChangeDirection(Direction.Left);
break;
case Key.Right:
                    gameState.ChangeDirection(Direction.Right);
break;
case Key.Up:
                    gameState.ChangeDirection(Direction.Up);
break;
case Key.Down:
                    gameState.ChangeDirection(Direction.Down);
break;
            }
        }
        private async Task GameLoop()
        {
```

```
            while (!gameState.GameOver)
            {
                await Task.Delay(100);
                gameState.Move();
                Draw();
            }
        }
        private Image[,] SetupGrid()
        {
            Image[,]images = new Image[rows, cols];
            GameGrid.Rows = rows;
            GameGrid.Columns = cols;

            for(int r= 0; r < rows; r++)
            {
                for(int c=0;  c < cols; c++)
                {
                    Image image = new Image
                    {
                        Source = Images.Empty,
                        RenderTransformOrigin = new Point(0.5,0.5)
                    };
                    images[r, c] = image;
                    GameGrid.Children.Add(image);
                }
            }

            return images;
        }
        private void Draw()
        {
            DrawGrid();
            DrawSnakeHead();
            ScoreText.Text = $"PLAYER: {playerName} | SCORE: {gameState.Score} |
TURNS: {gameState.TurnsSinceLastFood}";  // Display turns since last food
        }
        private void DrawGrid()
        {
            for(int r= 0;r < rows; r++)
            {
                for(int c= 0; c < cols; c++)
                {
                    GridValue gridVal = gameState.Grid[r, c];
                    gridImages[r, c].Source = gridValToImage[gridVal];
                    gridImages[r, c].RenderTransform = Transform.Identity;
                }
            }       }
        private void DrawSnakeHead()
        {
            Position headPos = gameState.HeadPosition();
            Image image = gridImages[headPos.Row, headPos.Column];
            image.Source = Images.Head;
            int rotation = dirToRotation[gameState.Dir];
            image.RenderTransform = new RotateTransform (rotation);
        }
        private async Task DrawDeadSnake()
        {
            List<Position>positions = new List<Position>(gameState.SnakePositions());
```

```
            for ( int i= 0; i< positions.Count; i++)
            {
                Position pos = positions[i];
                ImageSource source = (i == 0) ? Images.DeadHead : Images.DeadBody;
                gridImages[pos.Row, pos.Column].Source = source;
                await Task.Delay(500);
            }       }
        private async Task ShowCountDown()
        {
            for(int i = 3;  i >=1; i--)
            {
                OverlayText.Text = i.ToString();
                await Task.Delay(500);
            }
        }
        private async Task ShowGameOver()
        {
            await DrawDeadSnake();
                await Task.Delay(1000);
            Overlay.Visibility = Visibility.Visible;
            OverlayText.Text = "PRESS ANY KEY TO START";
        }
    }
}
```

Player.cs:

```
using System; using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq; using System.Text;
using System.Threading.Tasks;

namespace Snake.Model
{
    public class Player
    {       [Key]
        public int PlayerId { get; set; }

        [Required]
        public string Name { get; set; }

        public int Score { get; set; }

        public int Turns { get; set; }

        public int HighestScore { get; set; }
    }
}
```

Position.cs:

```
using System;
using System.Collections.Generic;

namespace Snake
{
```

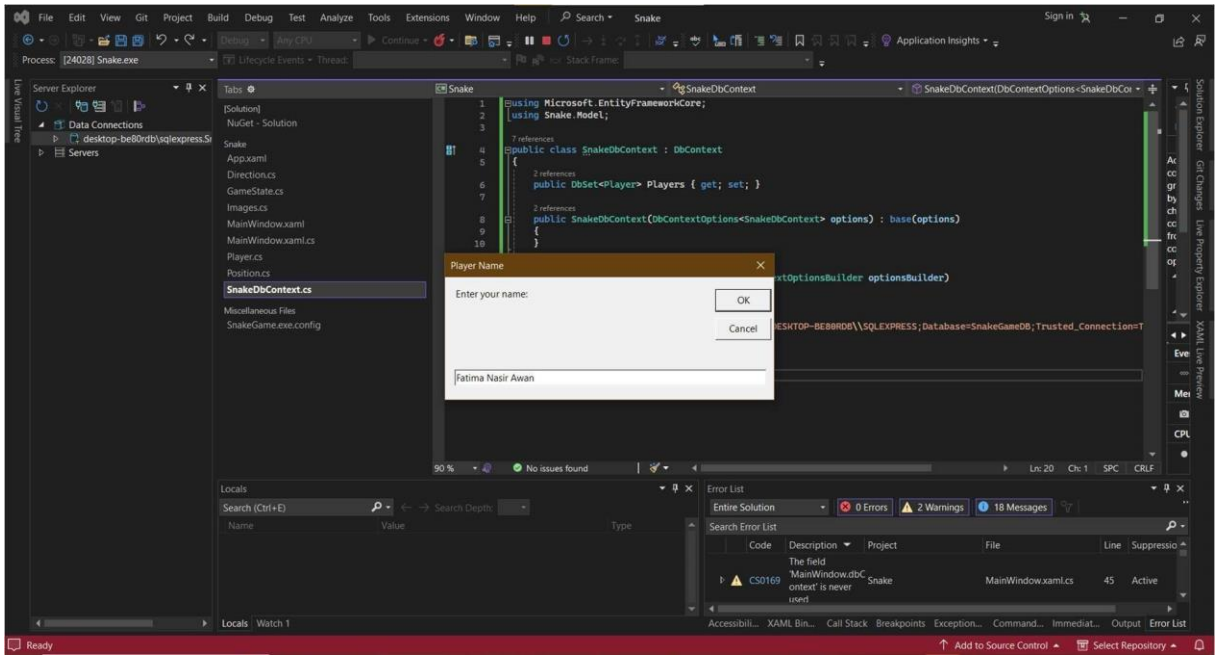
```
    public class Position
    {
        public int Row { get; }
public int Column { get; }

        public Position(int row, int column)
        {
            Row = row;
            Column = column;
        }
        public Position Translate(Direction dir)
        {
            return new(Row + dir.RowOffset, Column + dir.ColumnOffset);
        }
        public override bool Equals(object obj)
        {
            return obj is Position position && Row == position.Row &&
                   Column == position.Column;
        }
        public override int GetHashCode()
        {
            return HashCode.Combine(Row, Column);
        }
        public static bool operator ==(Position left, Position right)
        {
            return EqualityComparer<Position>.Default.Equals(left, right);
        }
        public static bool operator !=(Position left, Position right)
        {
            return !(left == right);
        }
    }
}
```

SnakeDBContext.cs:

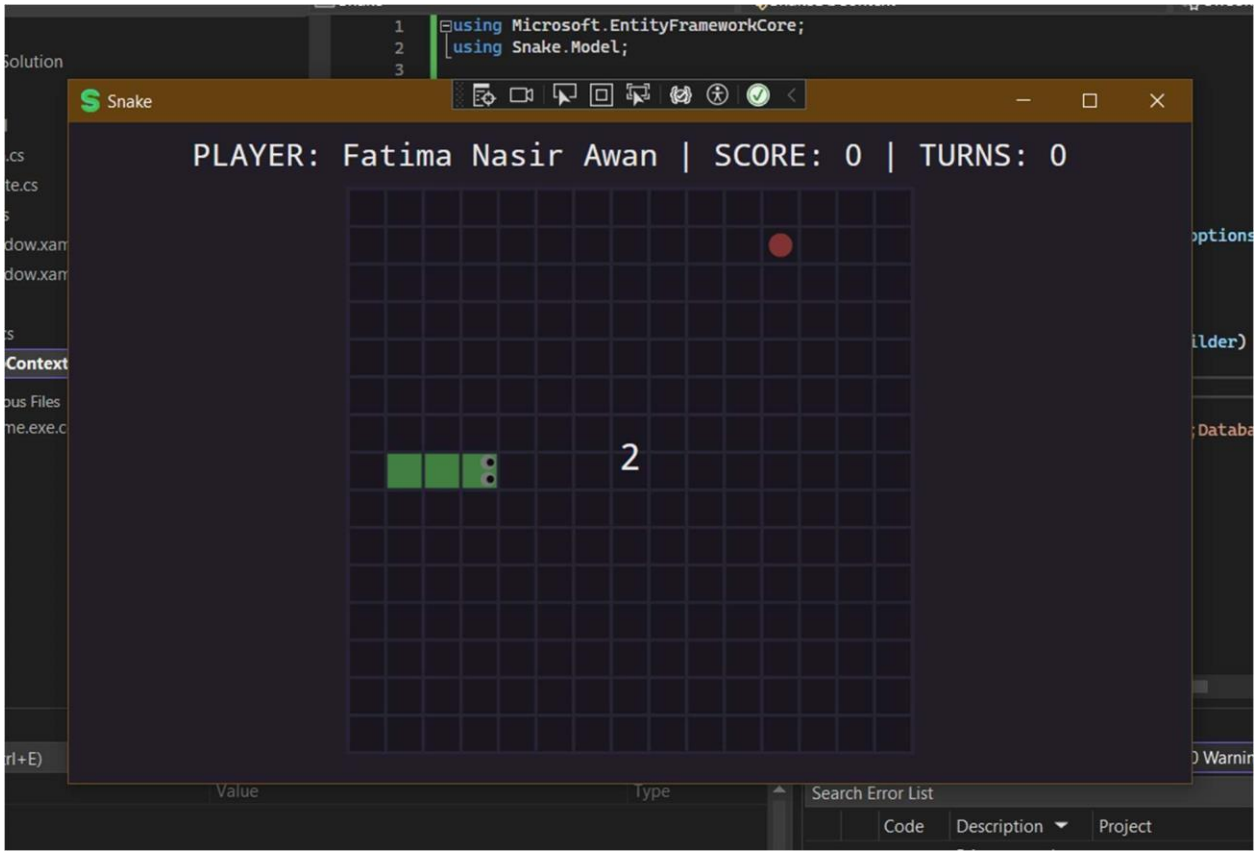
```
using Microsoft.EntityFrameworkCore;
using Snake.Model;

public class SnakeDbContext : DbContext
{
    public DbSet<Player> Players { get; set; }

    public SnakeDbContext(DbContextOptions<SnakeDbContext> options) : base(options)
    {
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            optionsBuilder.UseSqlServer("Server=DESKTOP-
BE80RDB\\SQLEXPRESS;Database=SnakeGameDB;Trusted_Connection=True;");
        }
    } }
```

Output:

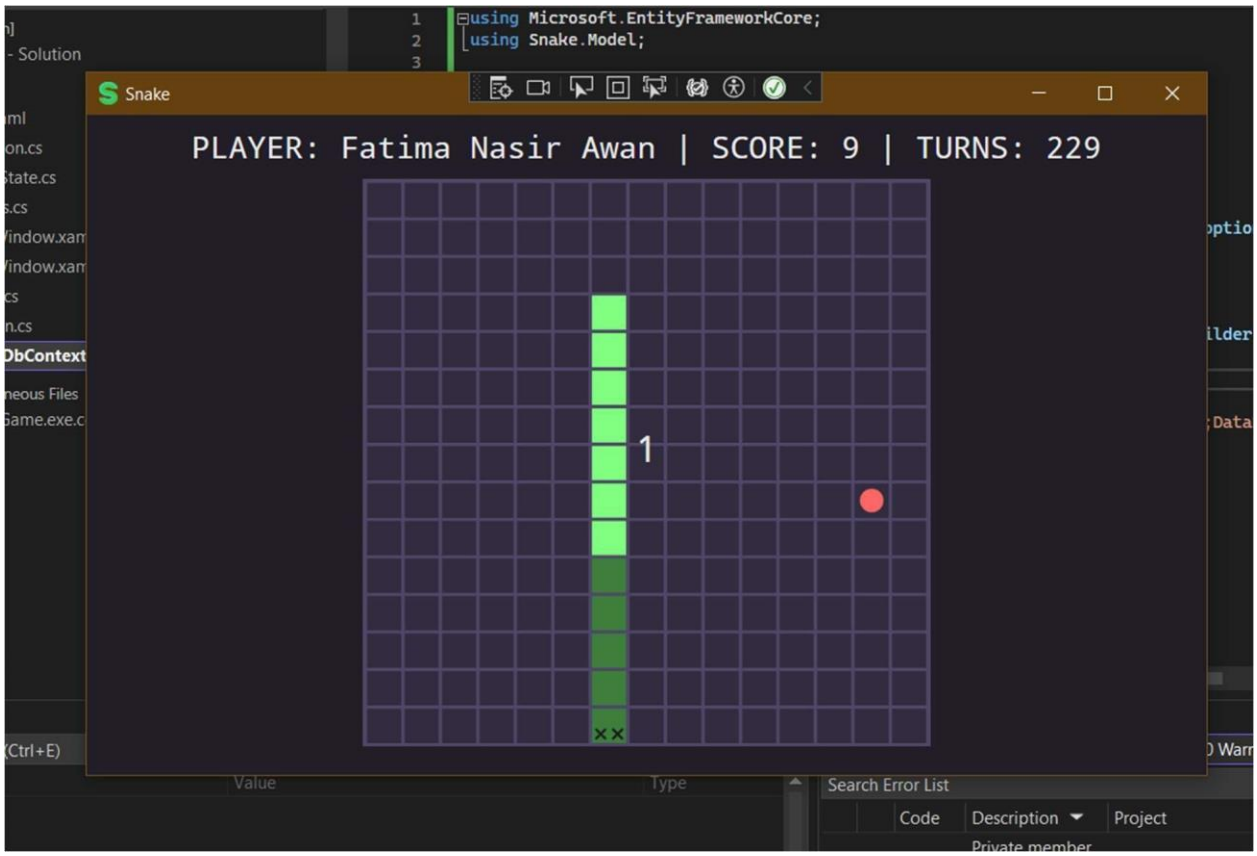
- When you run the program this dialogue box appears in which player enter his/her name:



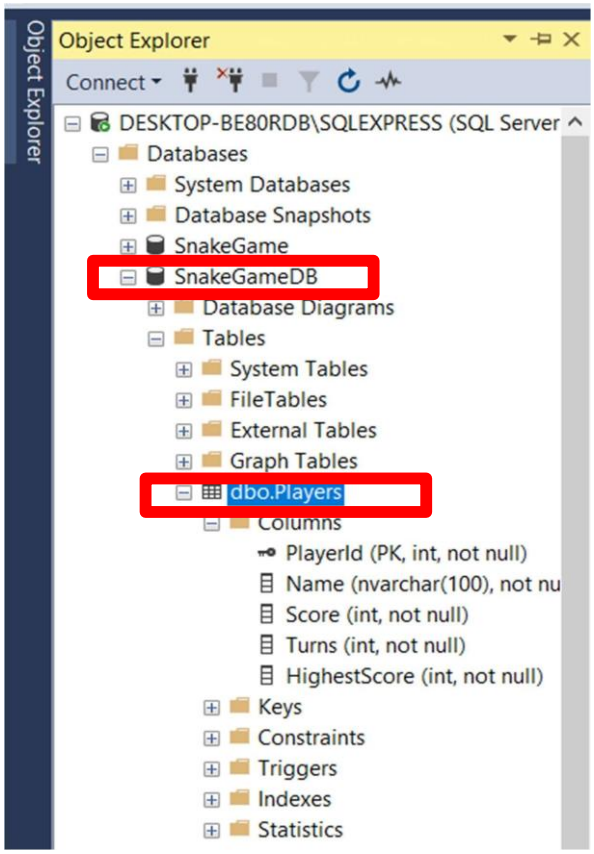
- After Pressing any key count down of 3 begins:



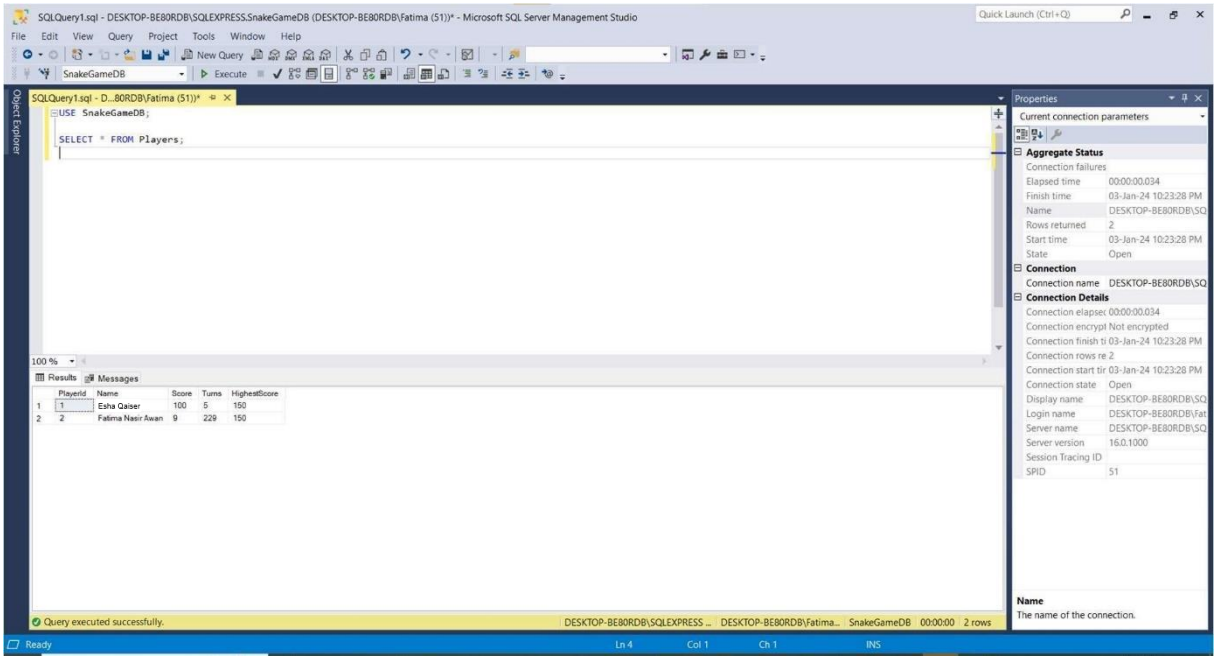
- As you can see when player out player name , his/her score and turns means number of steps of snake recorded.



- At the end Game Over message prints. As we created database “SnakeGameDB” with table:



- Now you can see the result by applying query. Successfully stored connection to SQL Database.



Class Diagram: