Games

Using C#

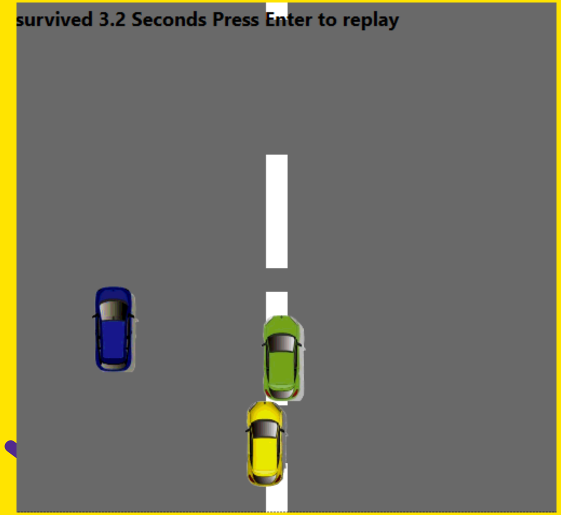
TABLE OF CONTENTS

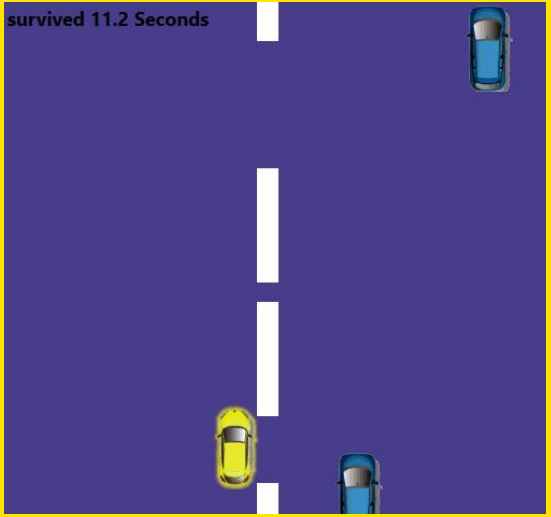
Intro

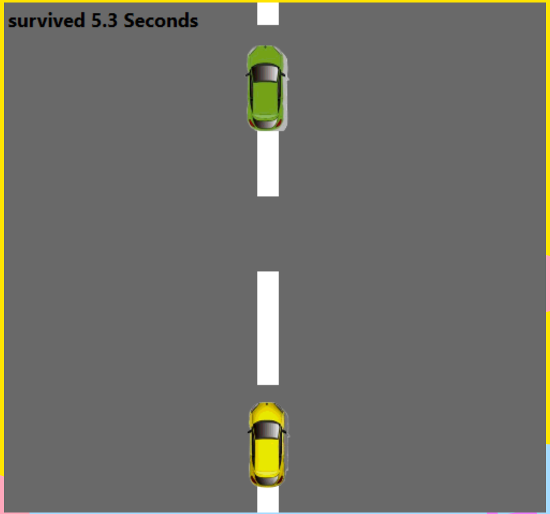
This project explores the development of four interactive games using Visual Studio, with a focus on **enhancing programming skills** and applying software development techniques.
Each game showcases **unique gameplay mechanics** and design, highlighting **creativity** and technical knowledge in C# programming and the WPF (Windows Presentation Foundation) framework. The project aims to develop functional and **engaging games** that demonstrate proficiency in coding, user interface design, and **problem-solving**. The games created include a Snake game, Ping pong game, floppy bird game and a car racing game. Throughout the project, we faced challenges in debugging, and creating the games , we aimed for a **great performance** and ensuring a **user-friendly interface**. By creating these games, we gained valuable experiences in game development, teamwork, and the Visual Studio environment.

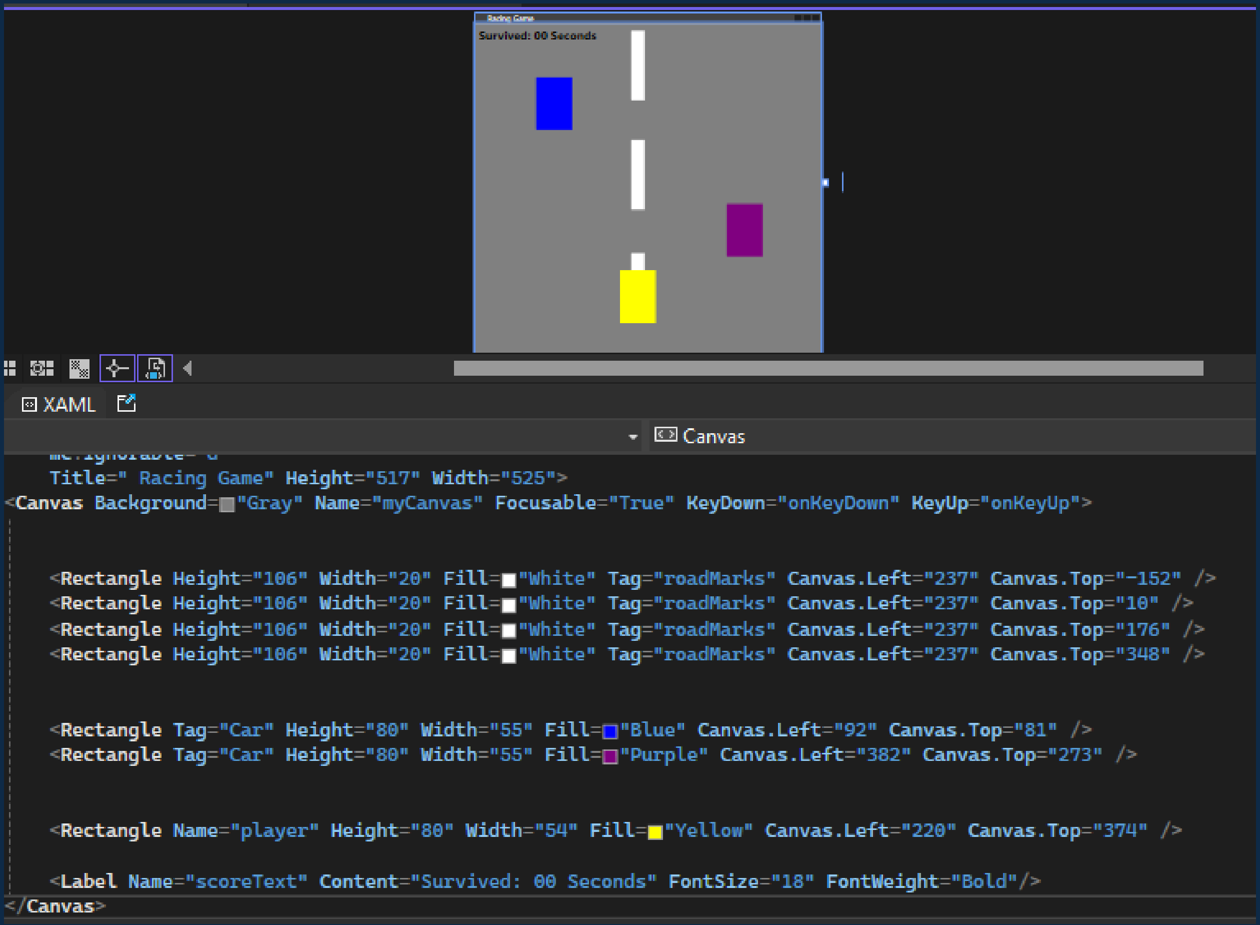
02.Car Racing Game

The Car Racing Game is a **simple** yet engaging **arcade-style** game developed in **C#** using WPF. The game places the player in control of a car, with the goal of navigating through a road filled with obstacles. The objective is to survive **as long as possible** by avoiding collisions with other cars, collecting **power-ups**, and achieving a high score. The game demonstrates key programming concepts such as event handling, **animation**, and collision detection, providing an **interactive** experience **that challenges** the player’s reflexes and timing.

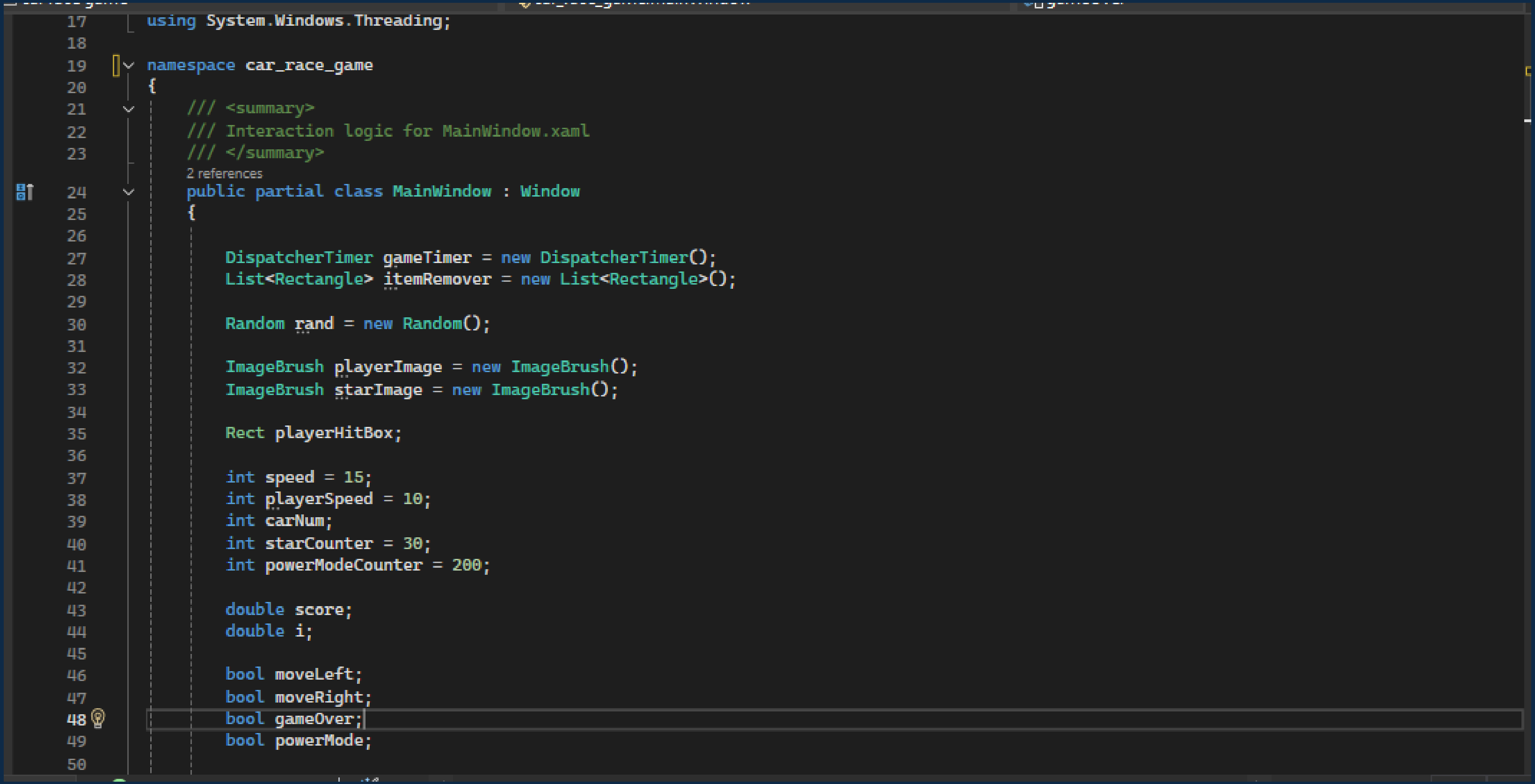

survived 3.2 Seconds Press Enter to replay


survived 11.2 Seconds

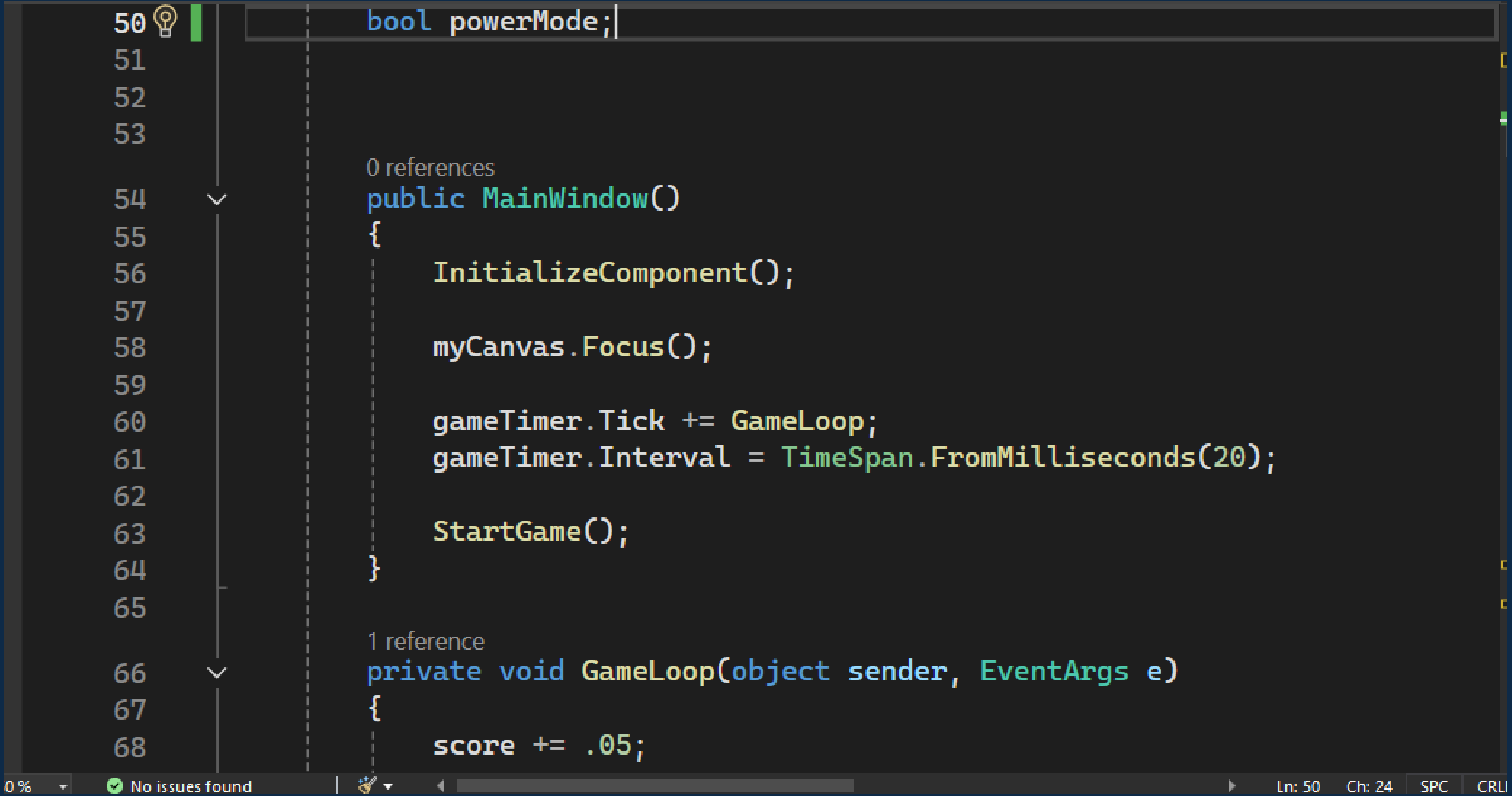

survived 5.3 Seconds



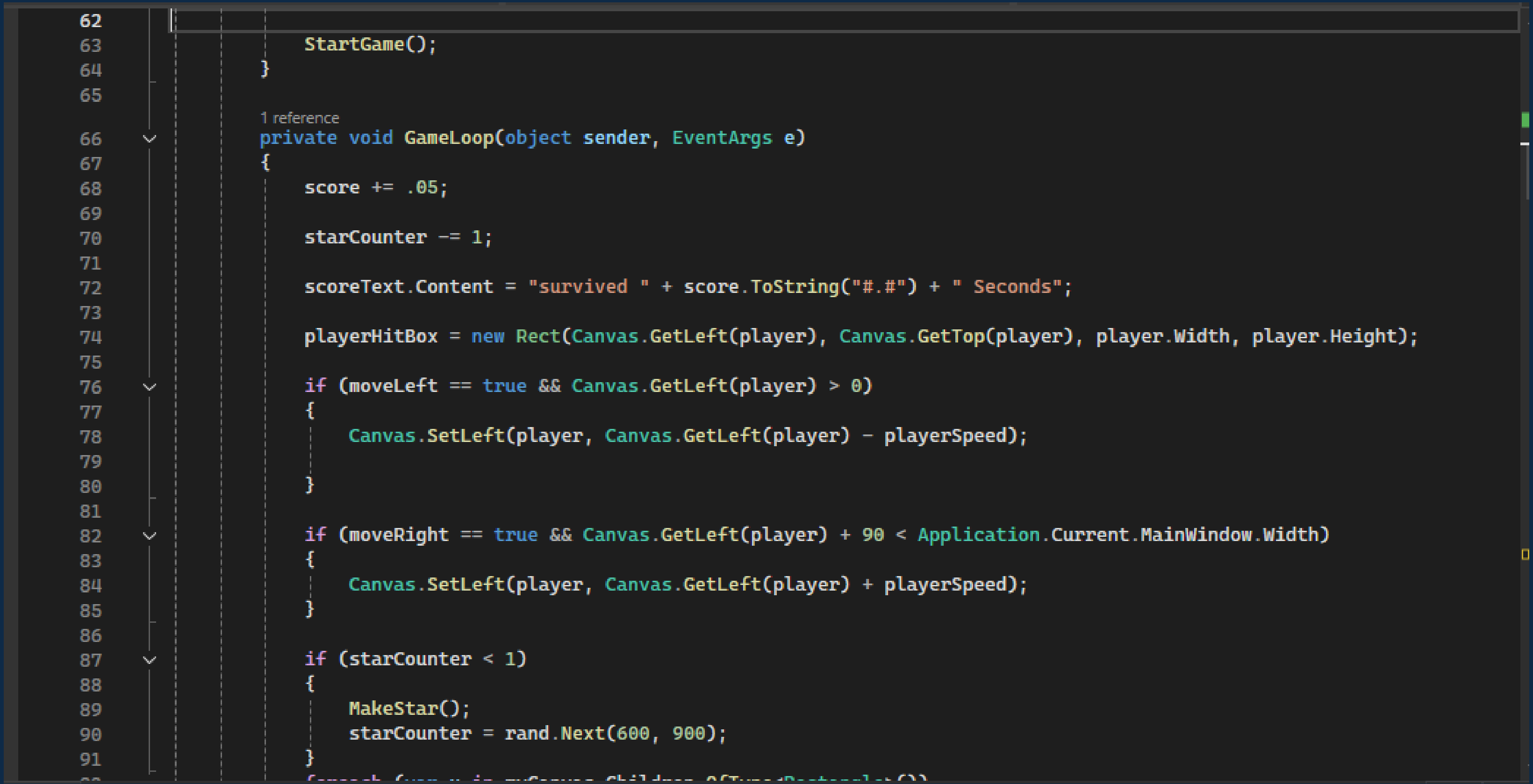
```xml
    Title=" Racing Game" Height="517" Width="525">
<Canvas Background="Gray" Name="myCanvas" Focusable="True" KeyDown="onKeyDown" KeyUp="onKeyUp">


    <Rectangle Height="106" Width="20" Fill="White" Tag="roadMarks" Canvas.Left="237" Canvas.Top="-152" />
    <Rectangle Height="106" Width="20" Fill="White" Tag="roadMarks" Canvas.Left="237" Canvas.Top="10" />
    <Rectangle Height="106" Width="20" Fill="White" Tag="roadMarks" Canvas.Left="237" Canvas.Top="176" />
    <Rectangle Height="106" Width="20" Fill="White" Tag="roadMarks" Canvas.Left="237" Canvas.Top="348" />


    <Rectangle Tag="Car" Height="80" Width="55" Fill="Blue" Canvas.Left="92" Canvas.Top="81" />
    <Rectangle Tag="Car" Height="80" Width="55" Fill="Purple" Canvas.Left="382" Canvas.Top="273" />


    <Rectangle Name="player" Height="80" Width="54" Fill="Yellow" Canvas.Left="220" Canvas.Top="374" />

    <Label Name="scoreText" Content="Survived: 00 Seconds" FontSize="18" FontWeight="Bold"/>
</Canvas>
```



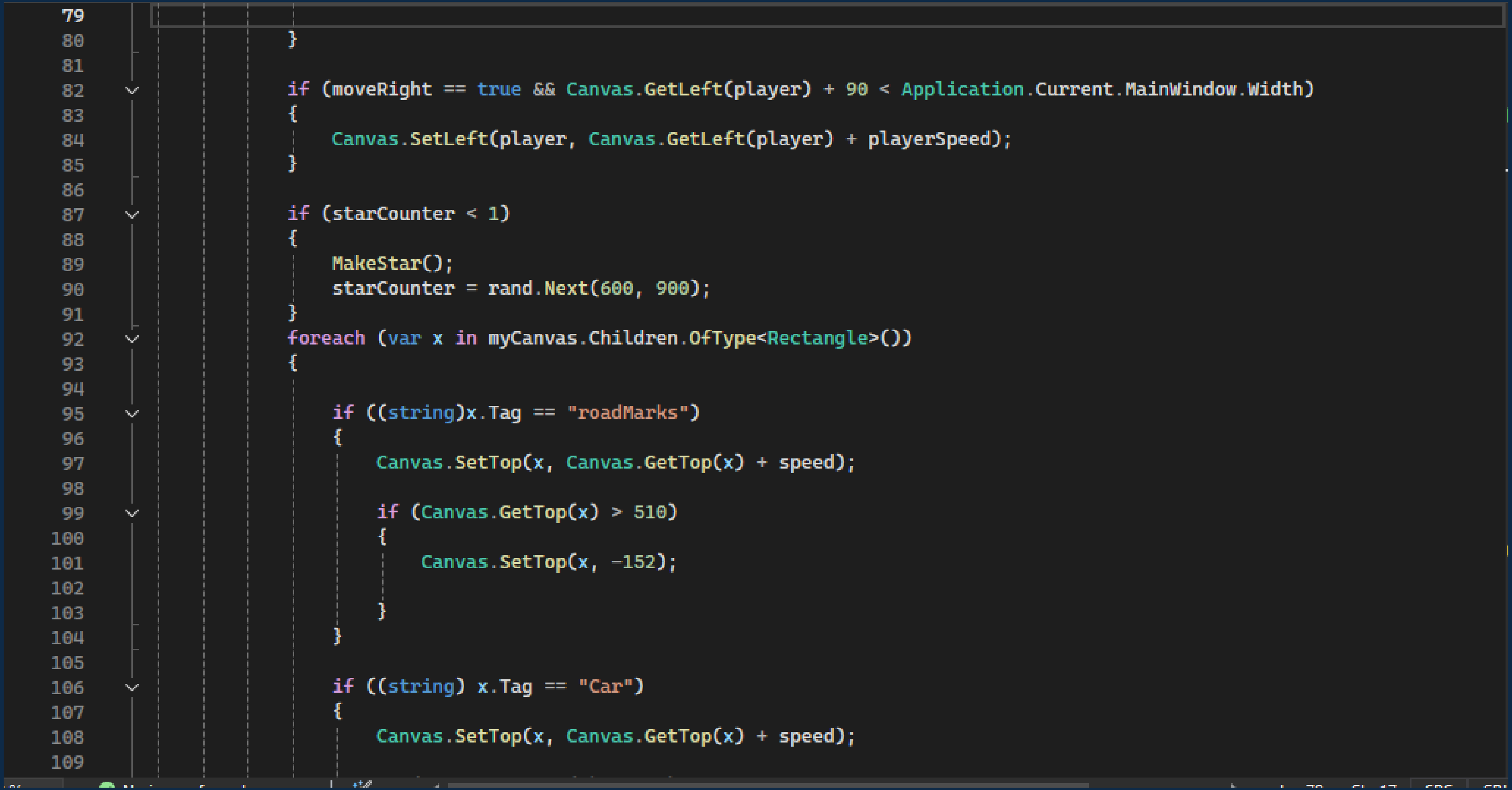
```
using System.Windows.Threading;

namespace car_race_game
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {

        DispatcherTimer gameTimer = new DispatcherTimer();
        List<Rectangle> itemRemover = new List<Rectangle>();

        Random rand = new Random();

        ImageBrush playerImage = new ImageBrush();
        ImageBrush starImage = new ImageBrush();

        Rect playerHitBox;

        int speed = 15;
        int playerSpeed = 10;
        int carNum;
        int starCounter = 30;
        int powerModeCounter = 200;

        double score;
        double i;

        bool moveLeft;
        bool moveRight;
        bool gameOver;
        bool powerMode;

```



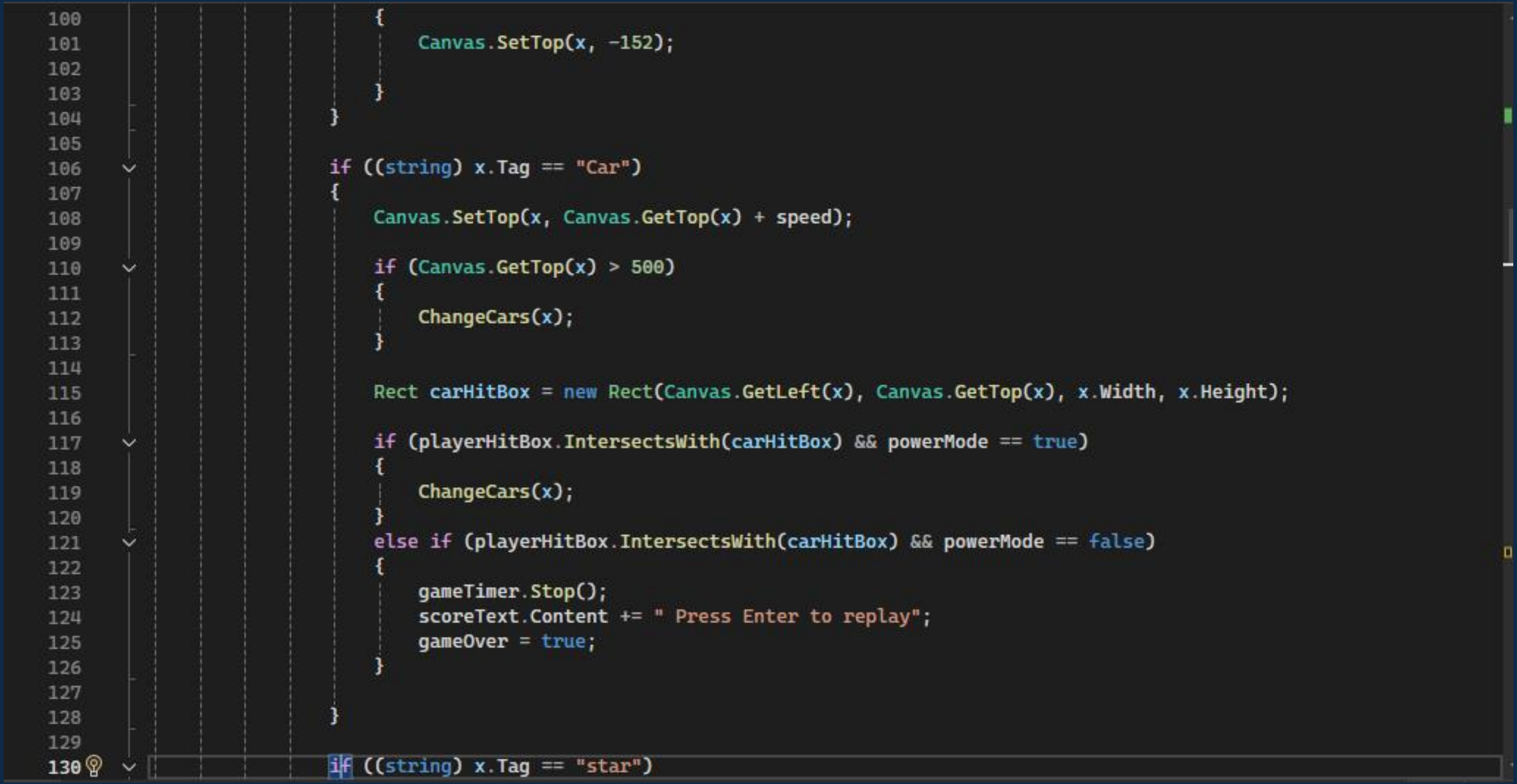
```
        bool powerMode;



        0 references
        public MainWindow()
        {
            InitializeComponent();

            myCanvas.Focus();

            gameTimer.Tick += GameLoop;
            gameTimer.Interval = TimeSpan.FromMilliseconds(20);

            StartGame();
        }

        1 reference
        private void GameLoop(object sender, EventArgs e)
        {
            score += .05;
```



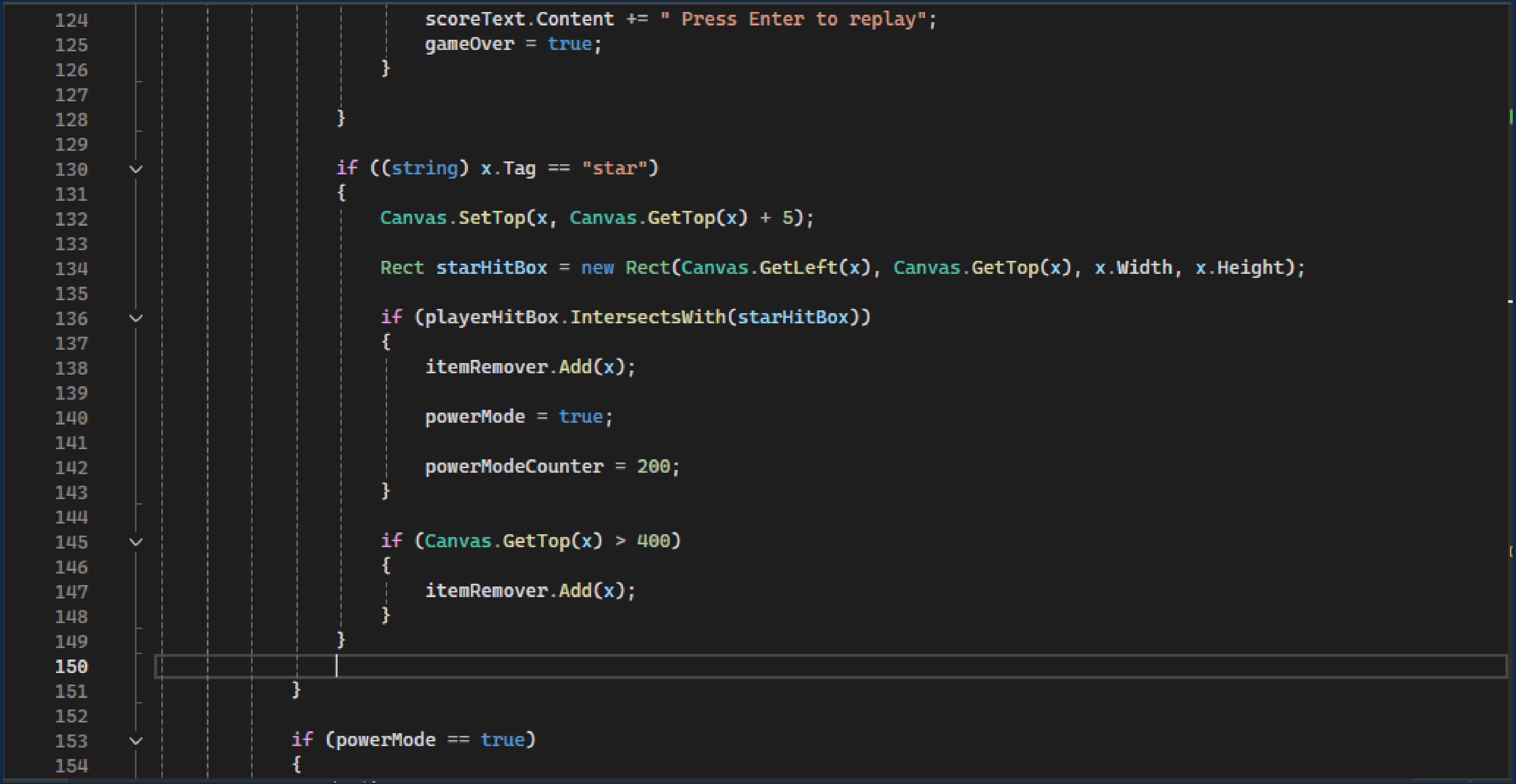
```
            StartGame();
        }

        1 reference
        private void GameLoop(object sender, EventArgs e)
        {
            score += .05;

            starCounter -= 1;

            scoreText.Content = "survived " + score.ToString("#.#") + " Seconds";

            playerHitBox = new Rect(Canvas.GetLeft(player), Canvas.GetTop(player), player.Width, player.Height);

            if (moveLeft == true && Canvas.GetLeft(player) > 0)
            {
                Canvas.SetLeft(player, Canvas.GetLeft(player) - playerSpeed);

            }

            if (moveRight == true && Canvas.GetLeft(player) + 90 < Application.Current.MainWindow.Width)
            {
                Canvas.SetLeft(player, Canvas.GetLeft(player) + playerSpeed);
            }

            if (starCounter < 1)
            {
                MakeStar();
                starCounter = rand.Next(600, 900);
            }
            foreach (var x in myCanvas.Children.OfType<Rectangle>())
```



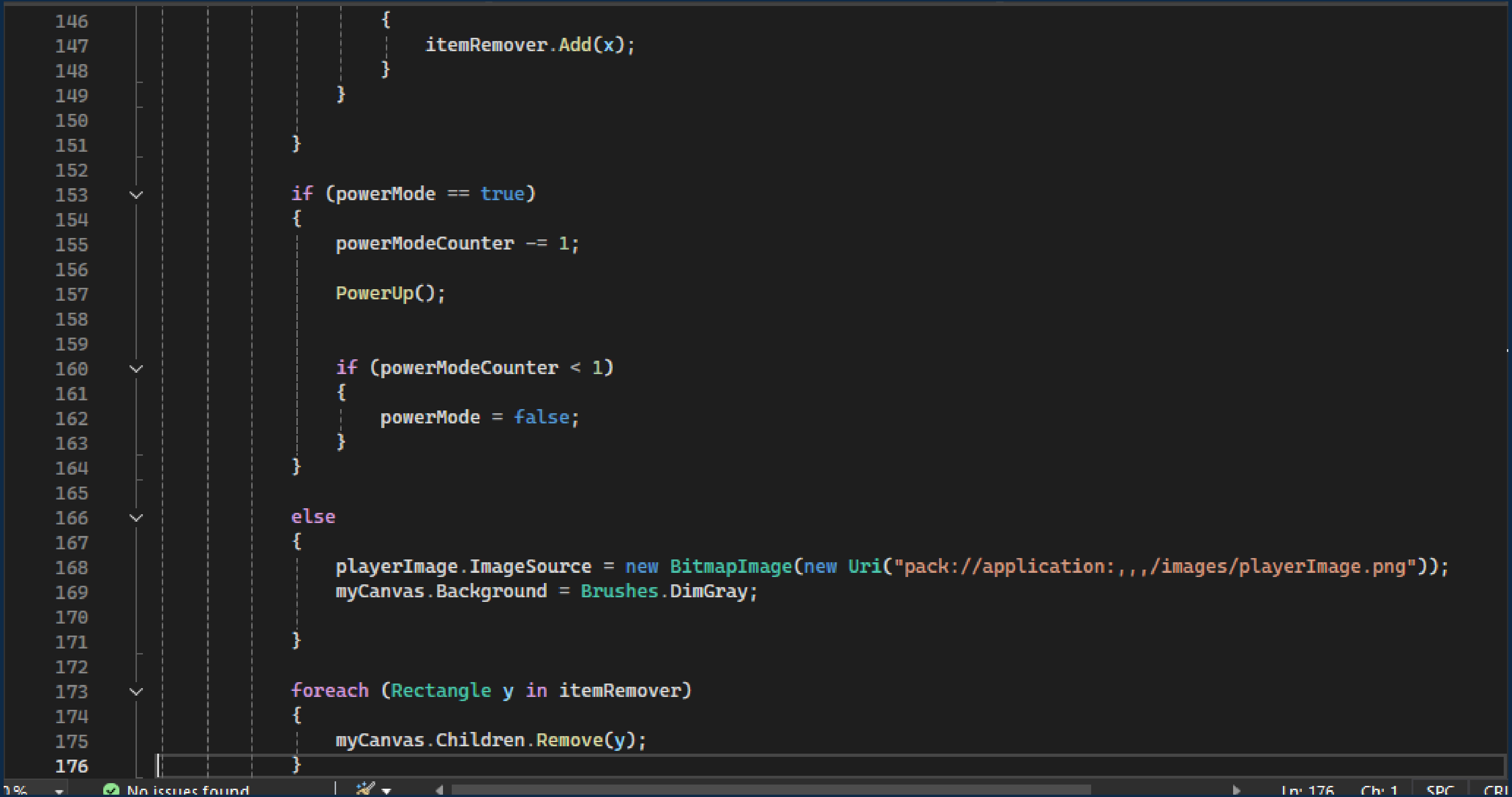
```
            }

            if (moveRight == true && Canvas.GetLeft(player) + 90 < Application.Current.MainWindow.Width)
            {
                Canvas.SetLeft(player, Canvas.GetLeft(player) + playerSpeed);
            }

            if (starCounter < 1)
            {
                MakeStar();
                starCounter = rand.Next(600, 900);
            }
            foreach (var x in myCanvas.Children.OfType<Rectangle>())
            {

                if ((string)x.Tag == "roadMarks")
                {
                    Canvas.SetTop(x, Canvas.GetTop(x) + speed);

                    if (Canvas.GetTop(x) > 510)
                    {
                        Canvas.SetTop(x, -152);

                    }
                }

                if ((string) x.Tag == "Car")
                {
                    Canvas.SetTop(x, Canvas.GetTop(x) + speed);

```



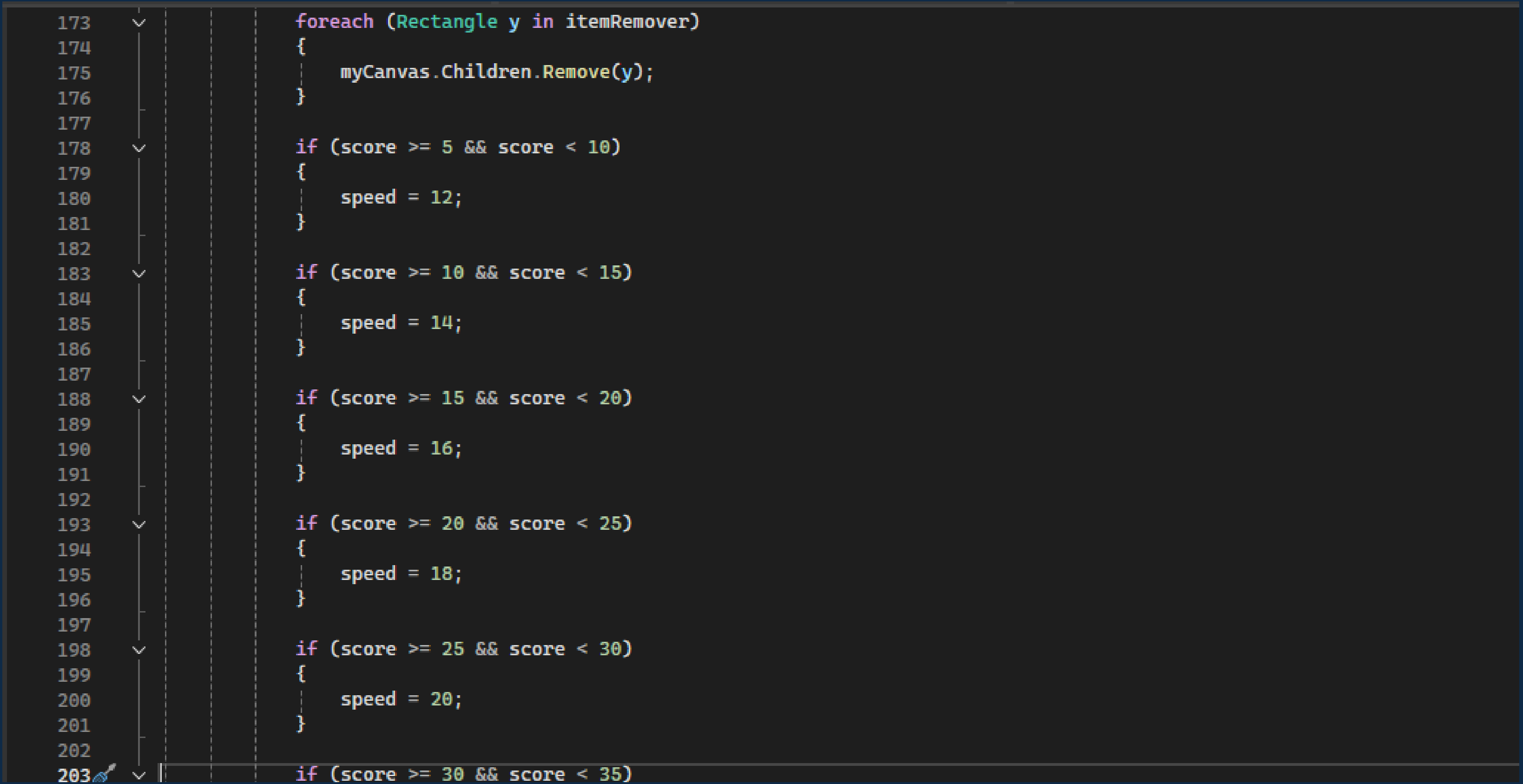
```
                {
                    Canvas.SetTop(x, -152);

                }
            }

            if ((string) x.Tag == "Car")
            {
                Canvas.SetTop(x, Canvas.GetTop(x) + speed);

                if (Canvas.GetTop(x) > 500)
                {
                    ChangeCars(x);
                }

                Rect carHitBox = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);

                if (playerHitBox.IntersectsWith(carHitBox) && powerMode == true)
                {
                    ChangeCars(x);
                }
                else if (playerHitBox.IntersectsWith(carHitBox) && powerMode == false)
                {
                    gameTimer.Stop();
                    scoreText.Content += " Press Enter to replay";
                    gameOver = true;
                }

            }

            if ((string) x.Tag == "star")
```



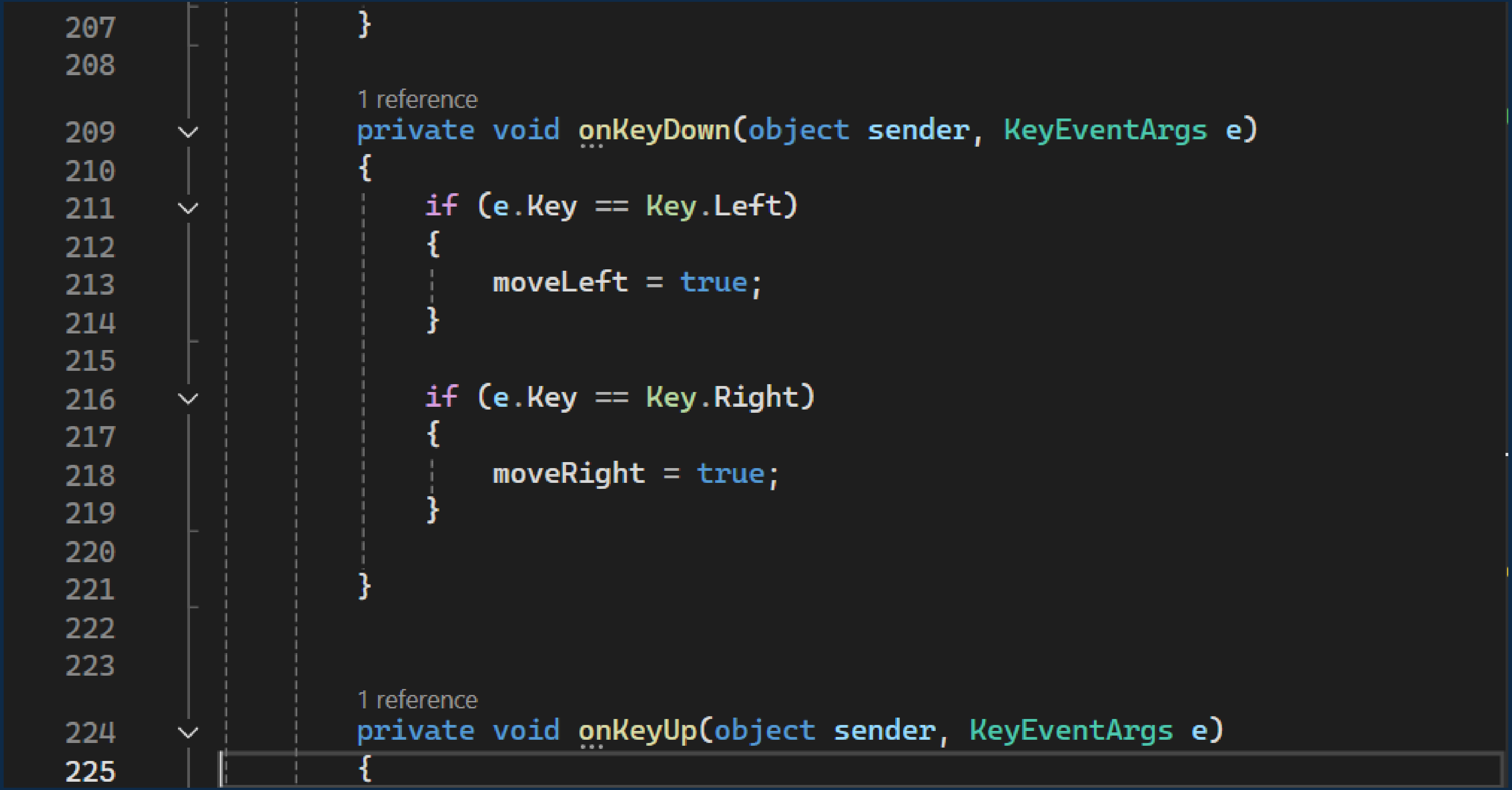
```
                        scoreText.Content += " Press Enter to replay";
                        gameOver = true;
                    }

                }

                if ((string) x.Tag == "star")
                {
                    Canvas.SetTop(x, Canvas.GetTop(x) + 5);

                    Rect starHitBox = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);

                    if (playerHitBox.IntersectsWith(starHitBox))
                    {
                        itemRemover.Add(x);

                        powerMode = true;

                        powerModeCounter = 200;
                    }

                    if (Canvas.GetTop(x) > 400)
                    {
                        itemRemover.Add(x);
                    }
                }

            }

            if (powerMode == true)
            {
```



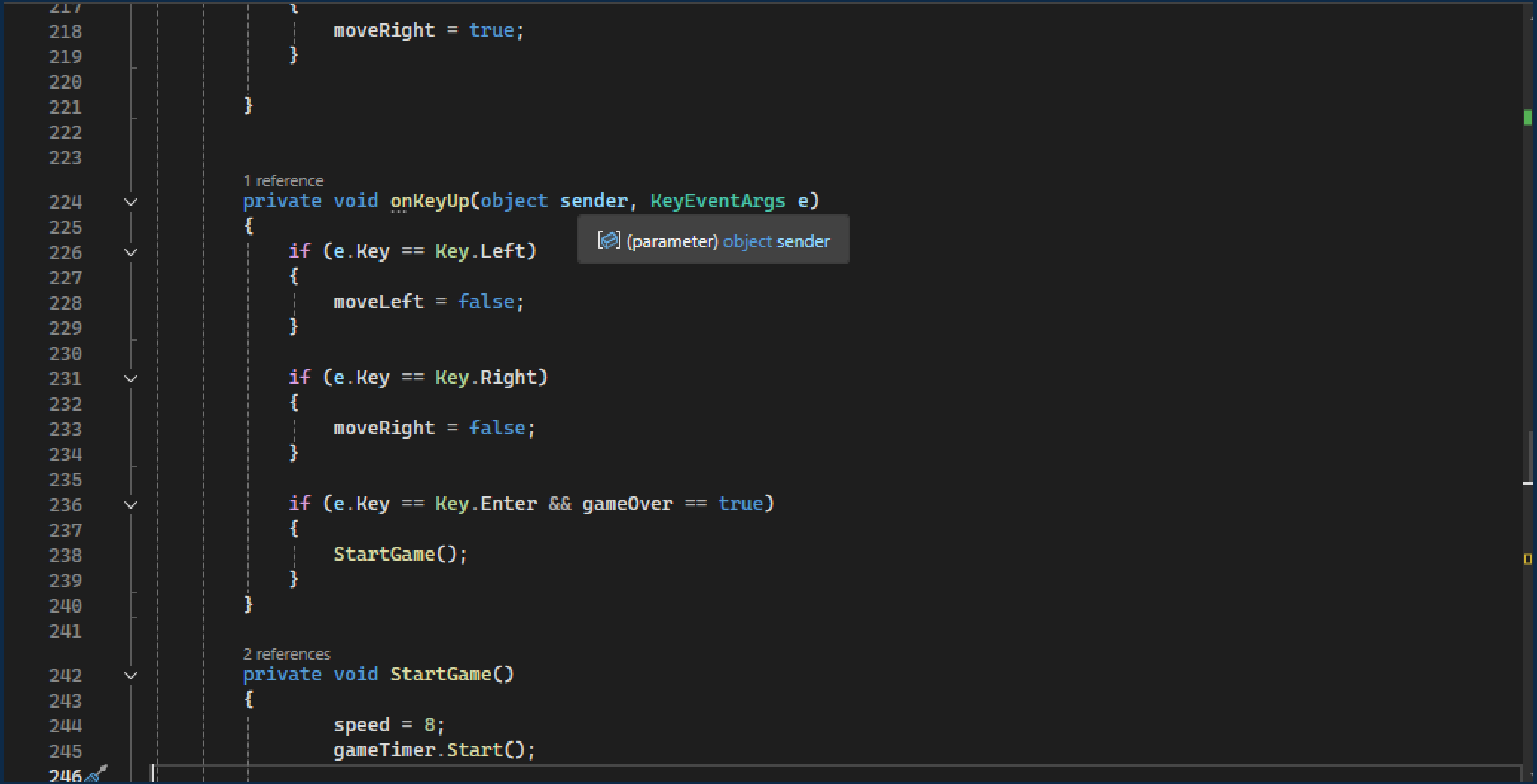
```
                    {
                        itemRemover.Add(x);
                    }
                }

            }

            if (powerMode == true)
            {
                powerModeCounter -= 1;

                PowerUp();


                if (powerModeCounter < 1)
                {
                    powerMode = false;
                }
            }

            else
            {
                playerImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/playerImage.png"));
                myCanvas.Background = Brushes.DimGray;

            }

            foreach (Rectangle y in itemRemover)
            {
                myCanvas.Children.Remove(y);
            }
```



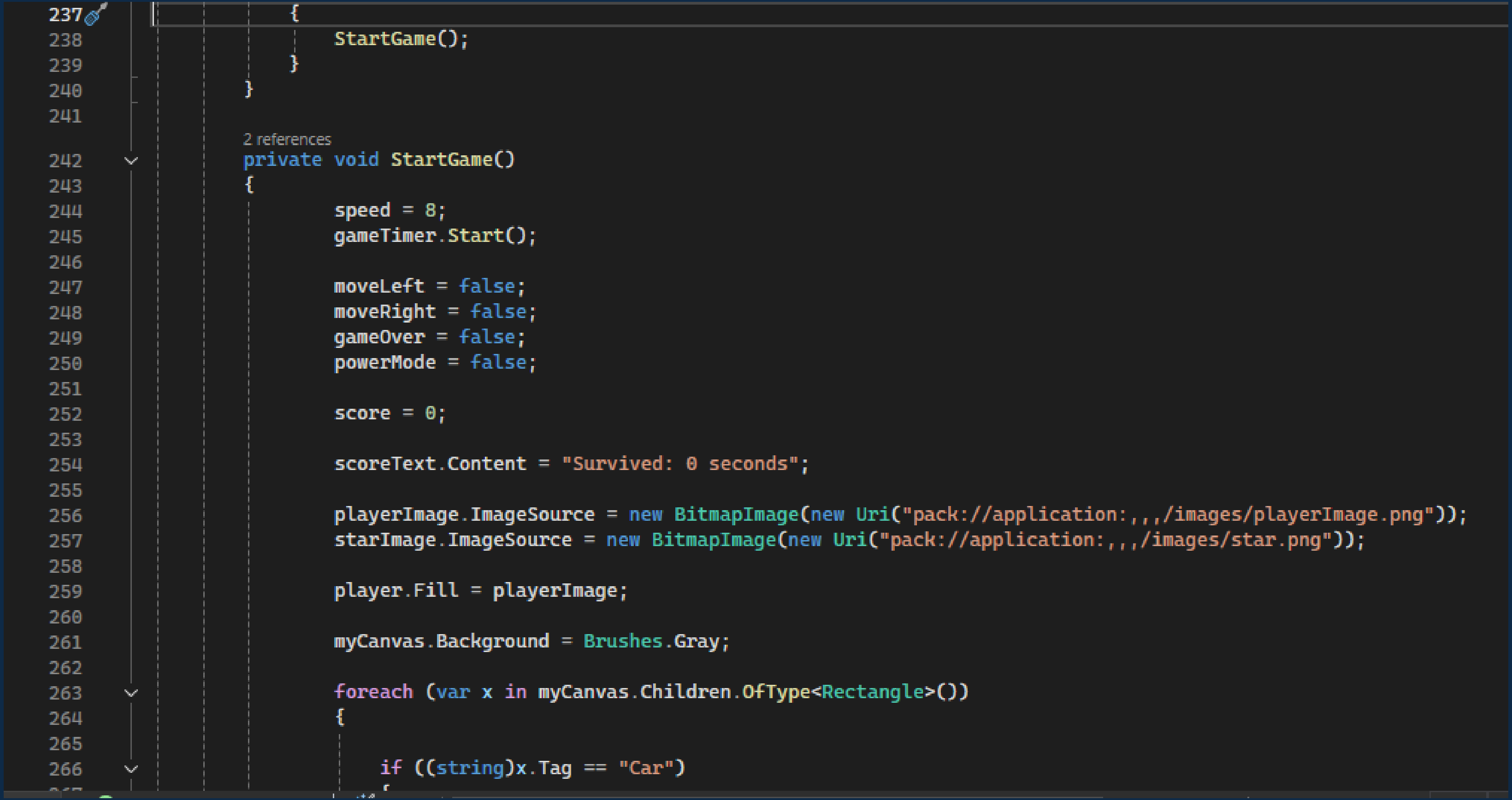
```
            foreach (Rectangle y in itemRemover)
            {
                myCanvas.Children.Remove(y);
            }

            if (score >= 5 && score < 10)
            {
                speed = 12;
            }

            if (score >= 10 && score < 15)
            {
                speed = 14;
            }

            if (score >= 15 && score < 20)
            {
                speed = 16;
            }

            if (score >= 20 && score < 25)
            {
                speed = 18;
            }

            if (score >= 25 && score < 30)
            {
                speed = 20;
            }

            if (score >= 30 && score < 35)
```



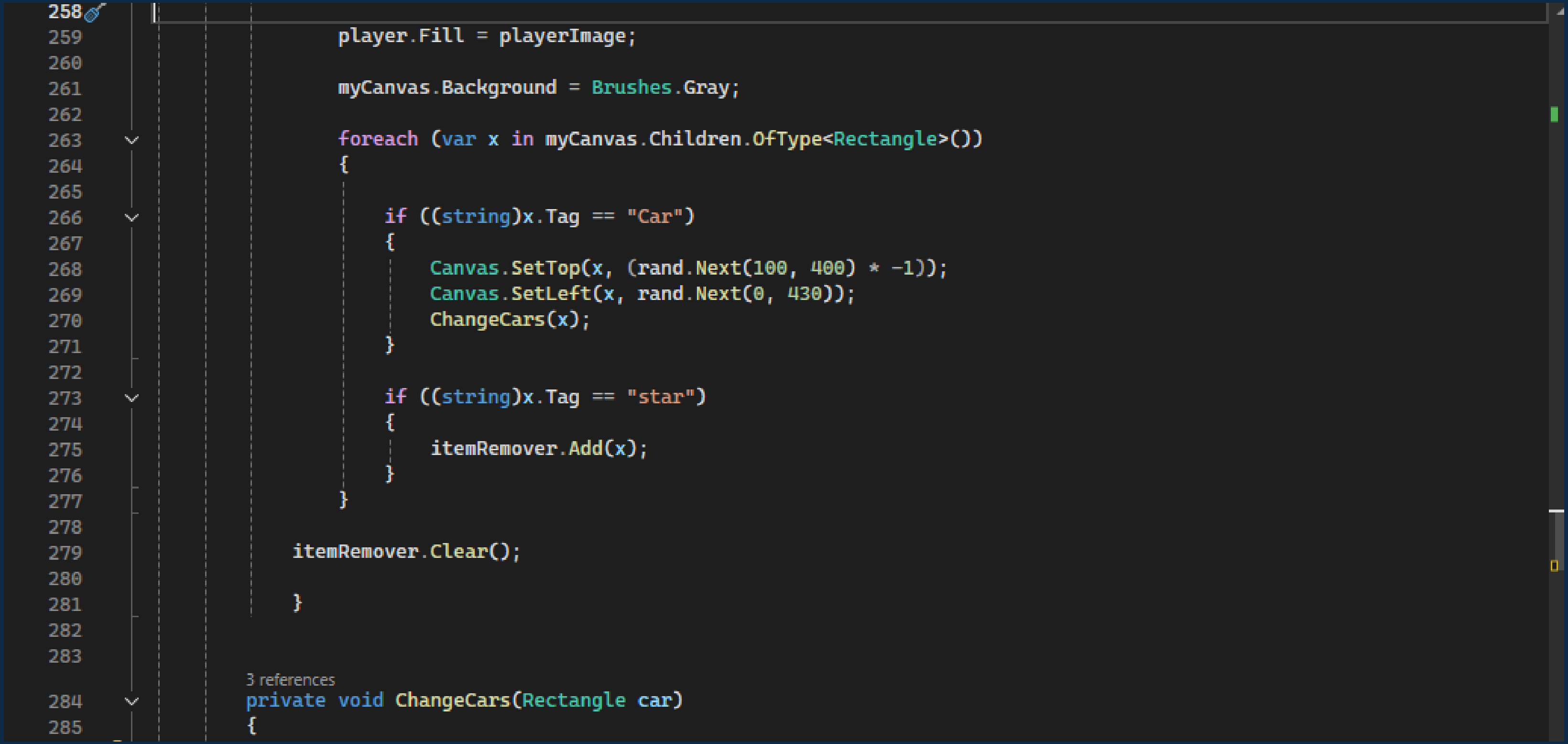
```
        }

        1 reference
        private void onKeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Left)
            {
                moveLeft = true;
            }

            if (e.Key == Key.Right)
            {
                moveRight = true;
            }

        }


        1 reference
        private void onKeyUp(object sender, KeyEventArgs e)
        {
```



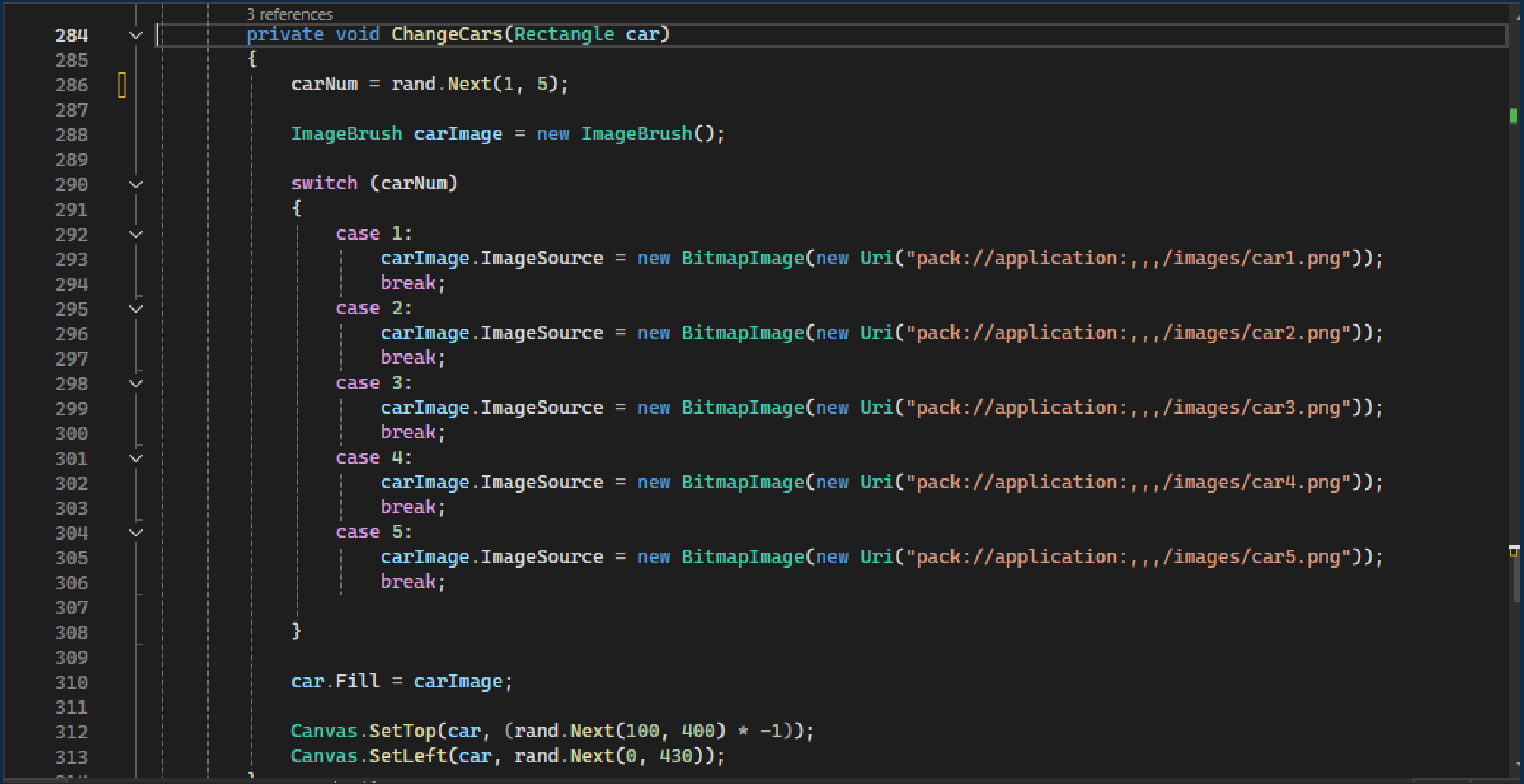
```
            moveRight = true;
        }

    }


    1 reference
    private void onKeyUp(object sender, KeyEventArgs e)
    {
        if (e.Key == Key.Left)
        {
            moveLeft = false;
        }

        if (e.Key == Key.Right)
        {
            moveRight = false;
        }

        if (e.Key == Key.Enter && gameOver == true)
        {
            StartGame();
        }
    }

    2 references
    private void StartGame()
    {
            speed = 8;
            gameTimer.Start();
```



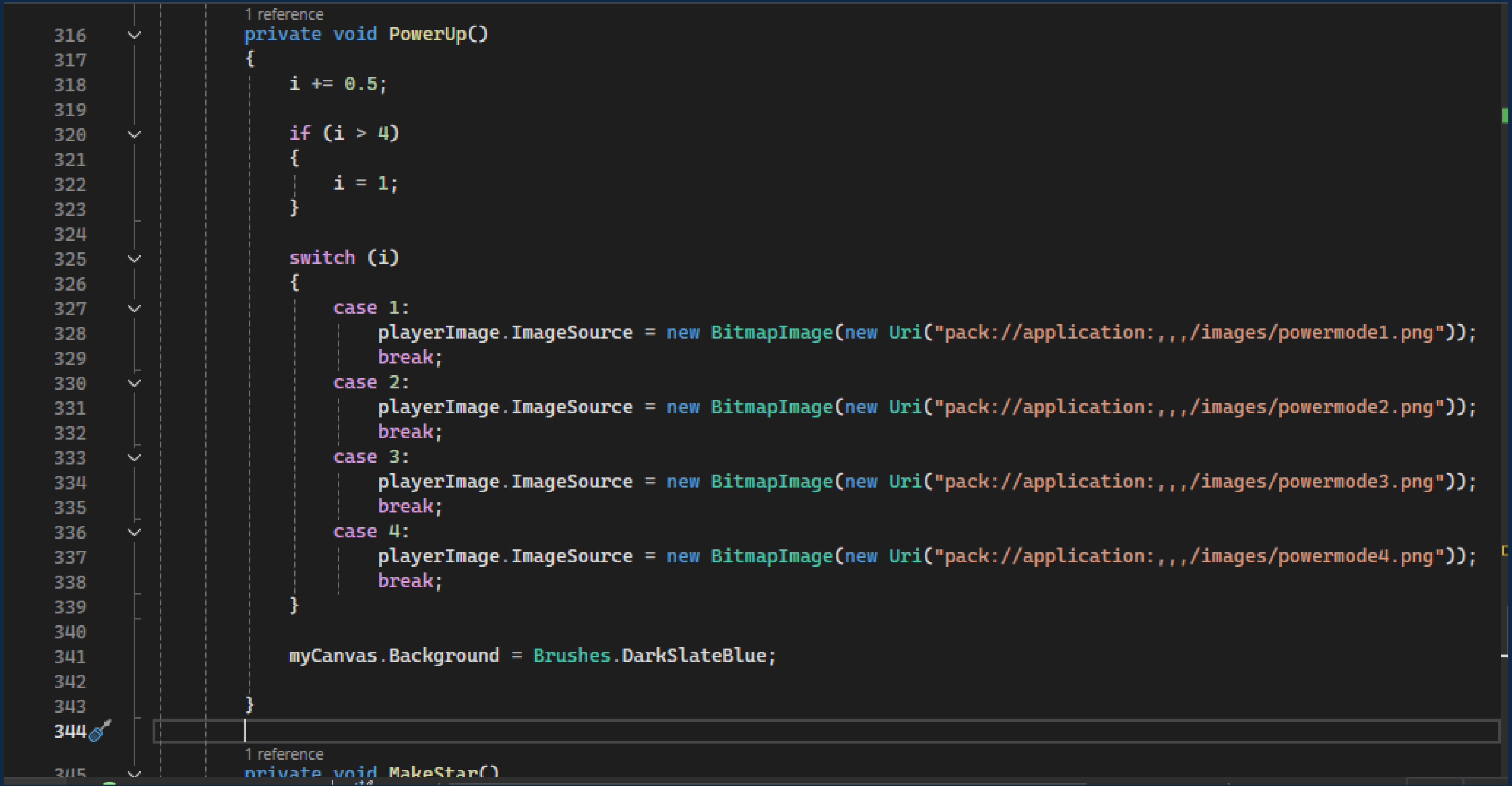
```
            {
                StartGame();
            }
        }

        2 references
        private void StartGame()
        {
                speed = 8;
                gameTimer.Start();

                moveLeft = false;
                moveRight = false;
                gameOver = false;
                powerMode = false;

                score = 0;

                scoreText.Content = "Survived: 0 seconds";

                playerImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/playerImage.png"));
                starImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/star.png"));

                player.Fill = playerImage;

                myCanvas.Background = Brushes.Gray;

                foreach (var x in myCanvas.Children.OfType<Rectangle>())
                {

                    if ((string)x.Tag == "Car")
```



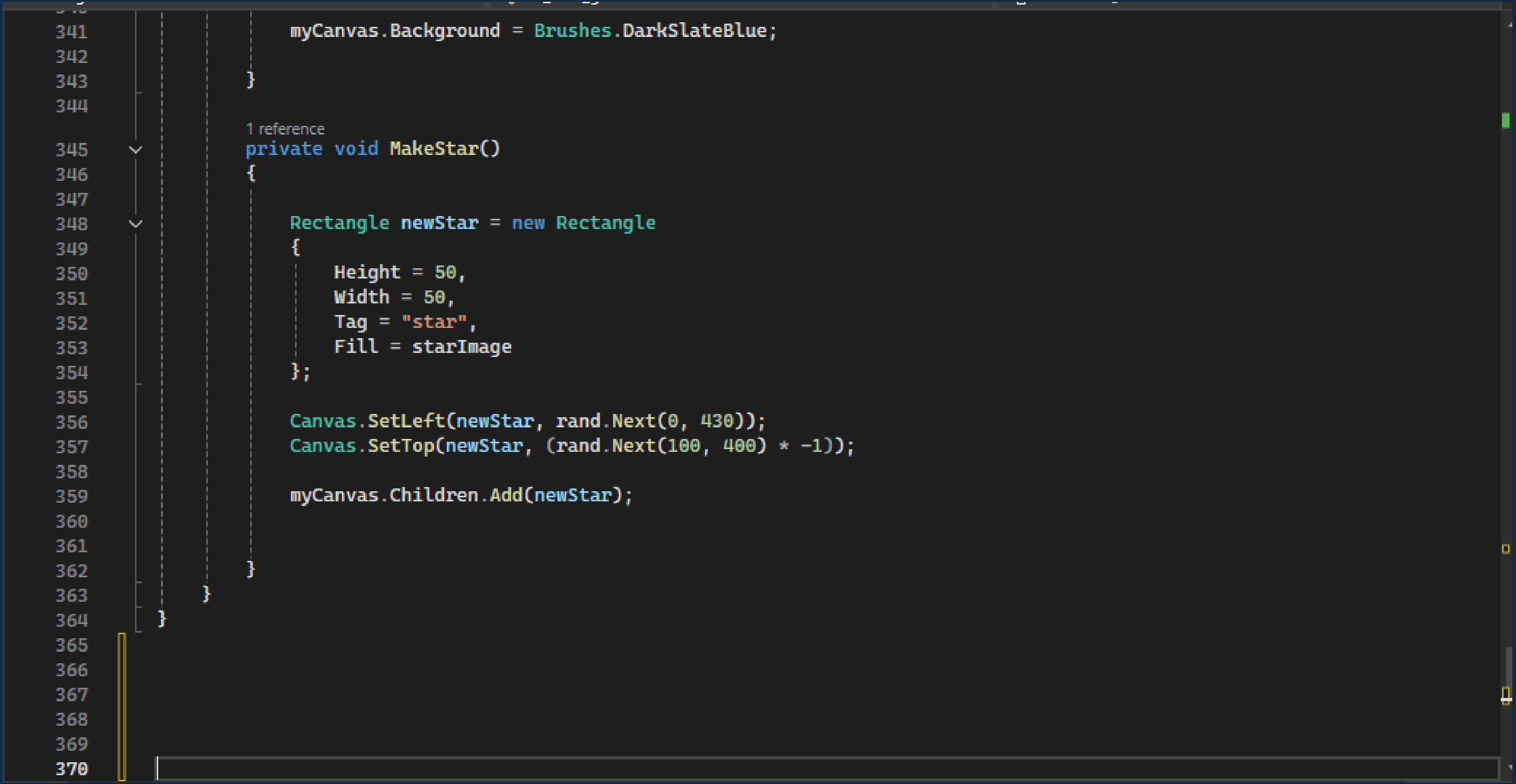
```
                player.Fill = playerImage;

                myCanvas.Background = Brushes.Gray;

                foreach (var x in myCanvas.Children.OfType<Rectangle>())
                {

                    if ((string)x.Tag == "Car")
                    {
                        Canvas.SetTop(x, (rand.Next(100, 400) * -1));
                        Canvas.SetLeft(x, rand.Next(0, 430));
                        ChangeCars(x);
                    }

                    if ((string)x.Tag == "star")
                    {
                        itemRemover.Add(x);
                    }
                }

            itemRemover.Clear();

            }


        3 references
        private void ChangeCars(Rectangle car)
        {
```



```
3 references
private void ChangeCars(Rectangle car)
{
    carNum = rand.Next(1, 5);

    ImageBrush carImage = new ImageBrush();

    switch (carNum)
    {
        case 1:
            carImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/car1.png"));
            break;
        case 2:
            carImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/car2.png"));
            break;
        case 3:
            carImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/car3.png"));
            break;
        case 4:
            carImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/car4.png"));
            break;
        case 5:
            carImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/car5.png"));
            break;

    }

    car.Fill = carImage;

    Canvas.SetTop(car, (rand.Next(100, 400) * -1));
    Canvas.SetLeft(car, rand.Next(0, 430));
```



```
1 reference
private void PowerUp()
{
    i += 0.5;

    if (i > 4)
    {
        i = 1;
    }

    switch (i)
    {
        case 1:
            playerImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/powermode1.png"));
            break;
        case 2:
            playerImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/powermode2.png"));
            break;
        case 3:
            playerImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/powermode3.png"));
            break;
        case 4:
            playerImage.ImageSource = new BitmapImage(new Uri("pack://application:,,,/images/powermode4.png"));
            break;
    }

    myCanvas.Background = Brushes.DarkSlateBlue;

}

1 reference
private void MakeStar()
```

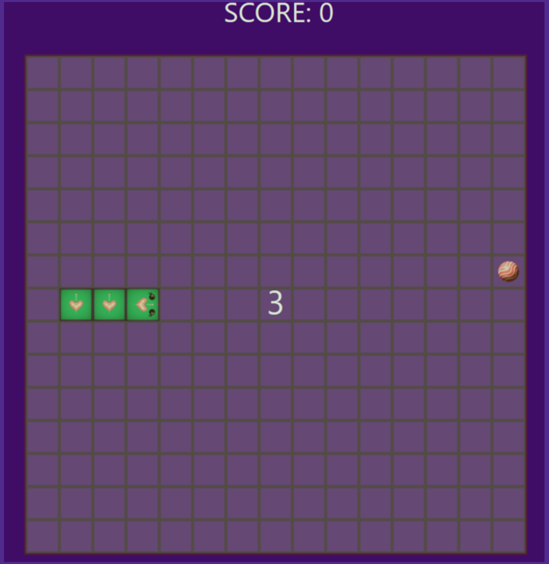


```
            myCanvas.Background = Brushes.DarkSlateBlue;

        }

        1 reference
        private void MakeStar()
        {

            Rectangle newStar = new Rectangle
            {
                Height = 50,
                Width = 50,
                Tag = "star",
                Fill = starImage
            };

            Canvas.SetLeft(newStar, rand.Next(0, 430));
            Canvas.SetTop(newStar, (rand.Next(100, 400) * -1));

            myCanvas.Children.Add(newStar);


        }
    }
}
```

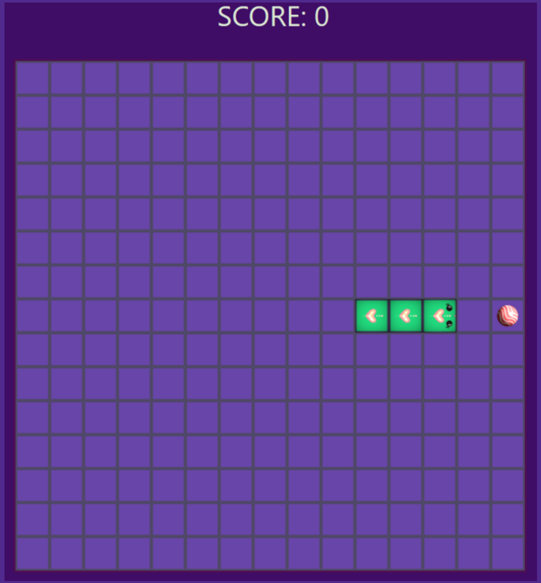
03.

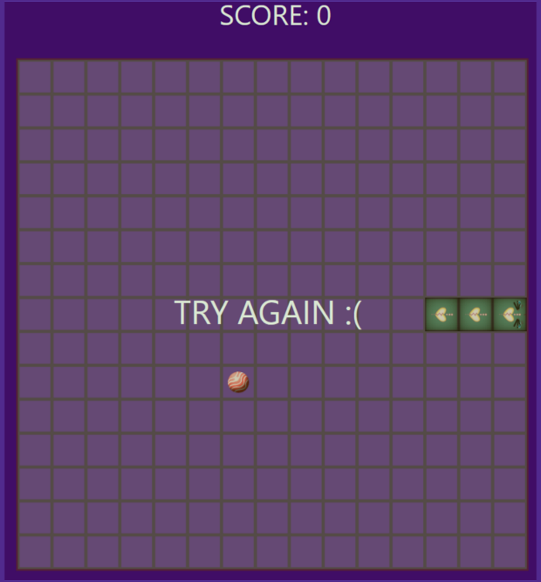
Snake Game

The **Snake** Game is a classic **arcade-style** game recreated in C# using WPF, where the player **controls** a snake that moves around the screen. The objective is to eat as many "food" items as possible without colliding with the walls or the snake's own body. With each item consumed, the snake grows longer, **increasing** the game's **difficulty**. This project showcases essential **programming skills** like managing dynamic object **movement**, handling collision detection, and **updating** real-time game **states**, making for a fun yet **challenging** game experience that tests a player’s **strategy** and precision.

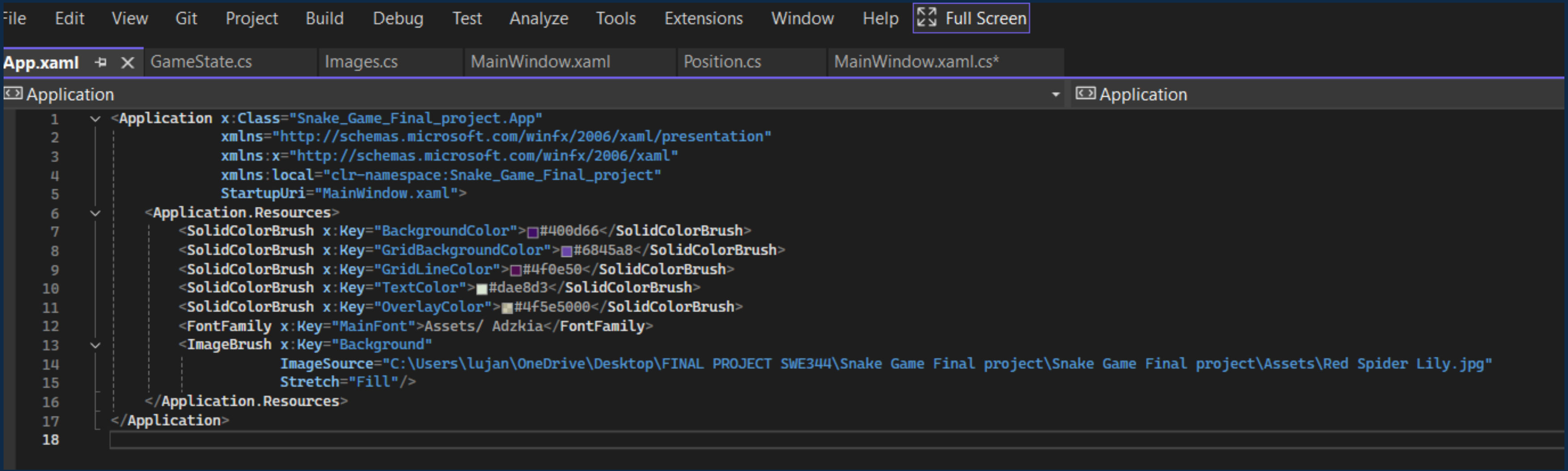

SCORE: 0

3

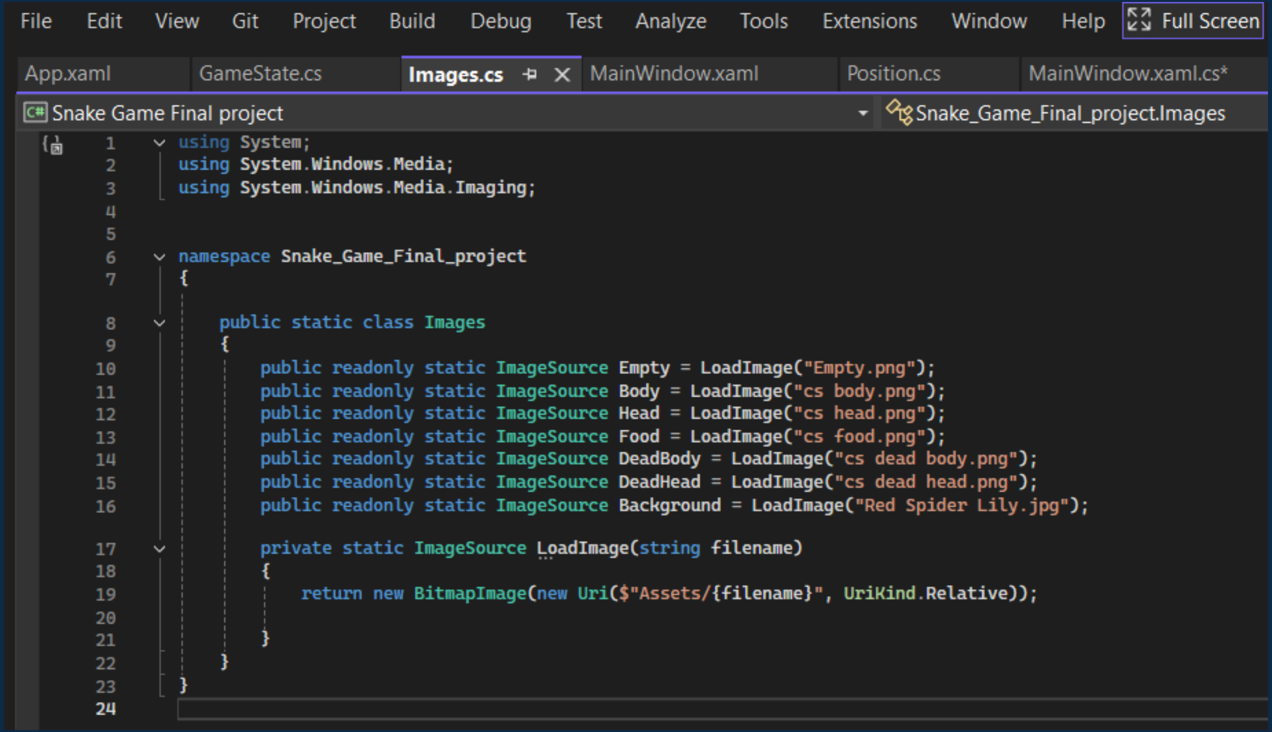

SCORE: 0


SCORE: 0

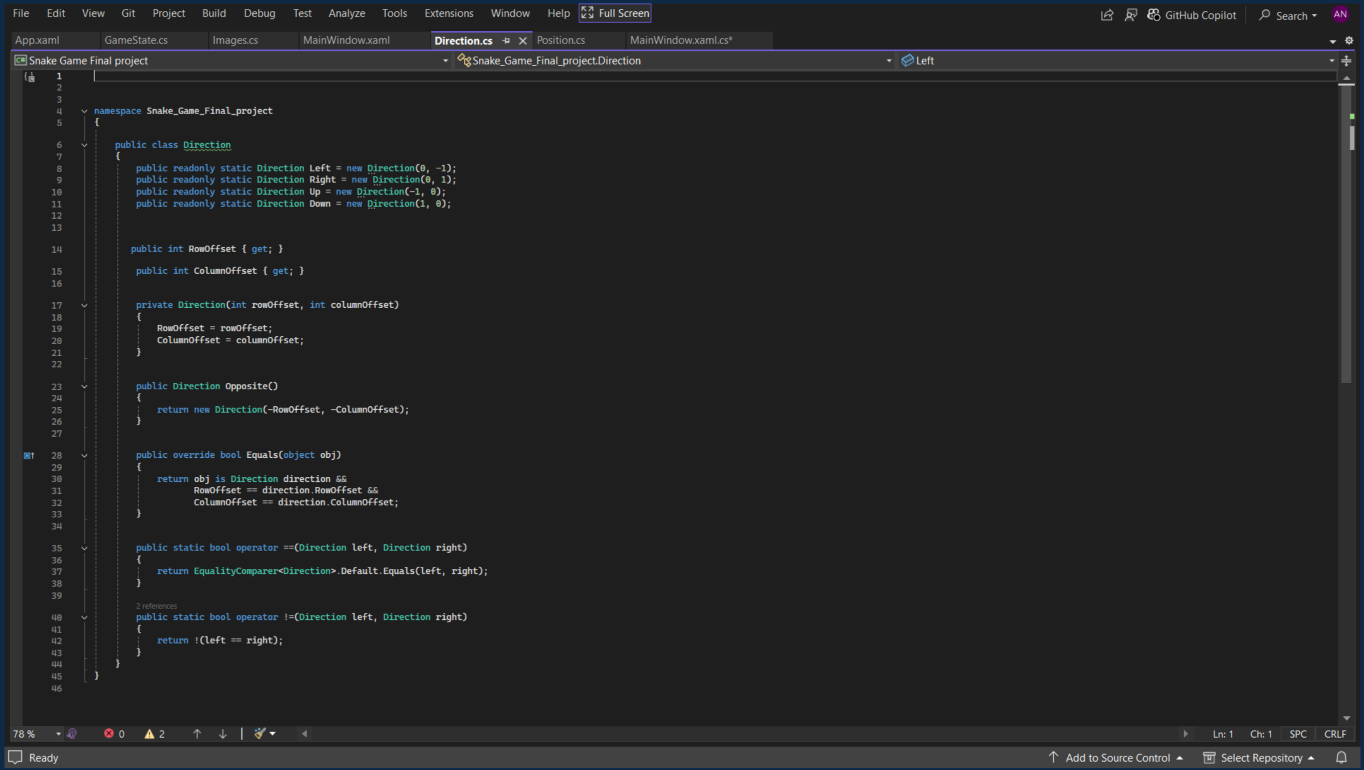
TRY AGAIN :(



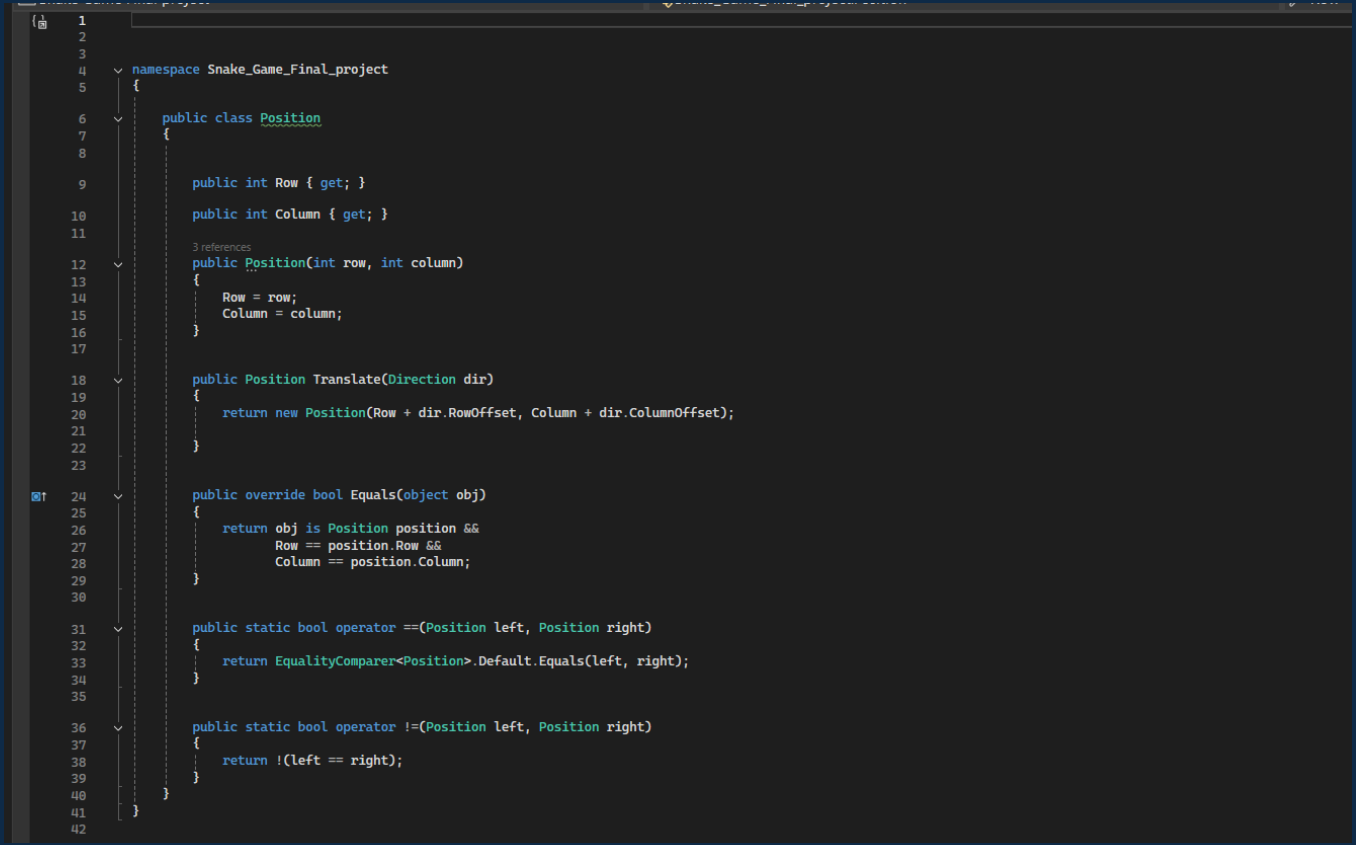
```xml
<Application x:Class="Snake_Game_Final_project.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:Snake_Game_Final_project"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <SolidColorBrush x:Key="BackgroundColor">#400d66</SolidColorBrush>
        <SolidColorBrush x:Key="GridBackgroundColor">#6845a8</SolidColorBrush>
        <SolidColorBrush x:Key="GridLineColor">#4f0e50</SolidColorBrush>
        <SolidColorBrush x:Key="TextColor">#dae8d3</SolidColorBrush>
        <SolidColorBrush x:Key="OverlayColor">#4f5e5000</SolidColorBrush>
        <FontFamily x:Key="MainFont">Assets/ Adzkia</FontFamily>
        <ImageBrush x:Key="Background"
                    ImageSource="C:\Users\lujan\OneDrive\Desktop\FINAL PROJECT SWE344\Snake Game Final project\Snake Game Final project\Assets\Red Spider Lily.jpg"
                    Stretch="Fill"/>
    </Application.Resources>
</Application>
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
App.xaml GameState.cs Images.cs MainWindow.xaml Position.cs MainWindow.xaml.cs*
Snake Game Final project
Snake_Game_Final_project.Images

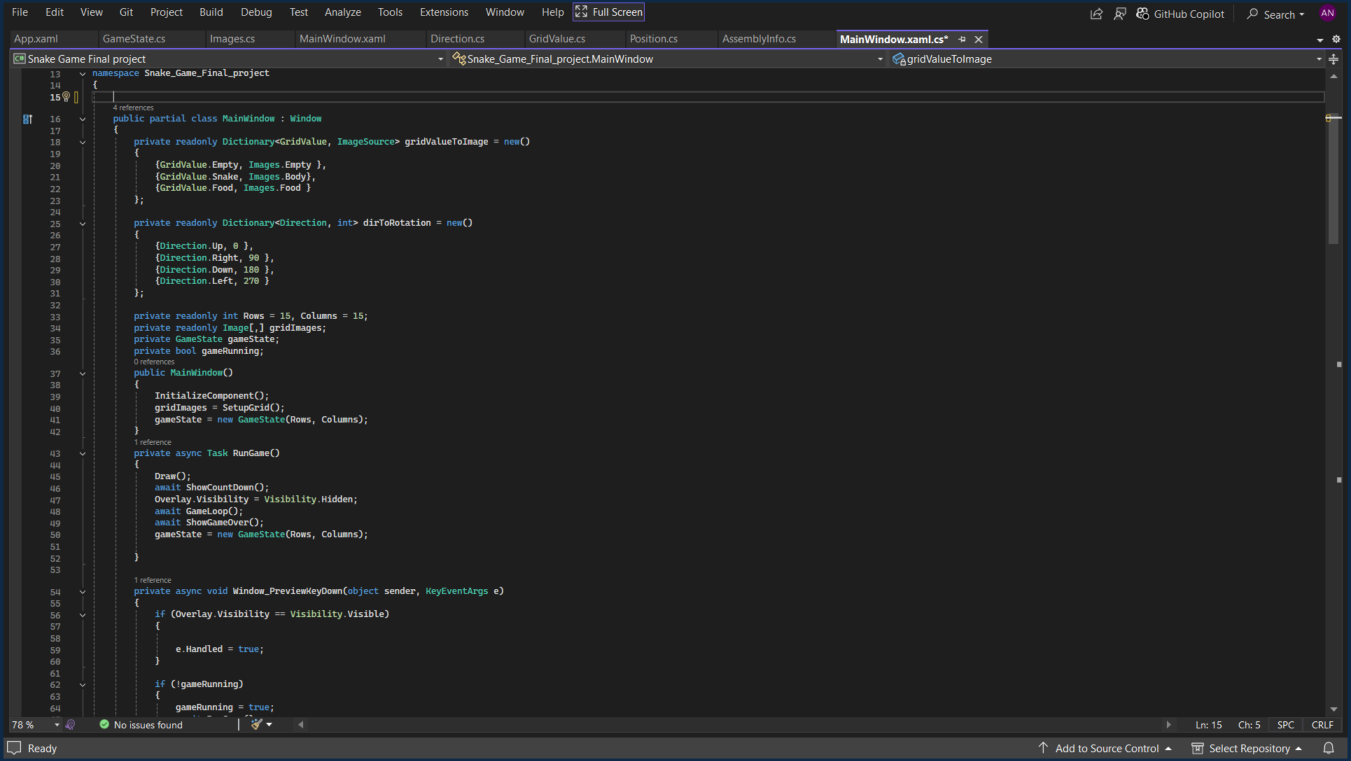
```csharp
using System;
using System.Windows.Media;
using System.Windows.Media.Imaging;


namespace Snake_Game_Final_project
{

    public static class Images
    {
        public readonly static ImageSource Empty = LoadImage("Empty.png");
        public readonly static ImageSource Body = LoadImage("cs body.png");
        public readonly static ImageSource Head = LoadImage("cs head.png");
        public readonly static ImageSource Food = LoadImage("cs food.png");
        public readonly static ImageSource DeadBody = LoadImage("cs dead body.png");
        public readonly static ImageSource DeadHead = LoadImage("cs dead head.png");
        public readonly static ImageSource Background = LoadImage("Red Spider Lily.jpg");

        private static ImageSource LoadImage(string filename)
        {
            return new BitmapImage(new Uri($"Assets/{filename}", UriKind.Relative));

        }
    }
}
```



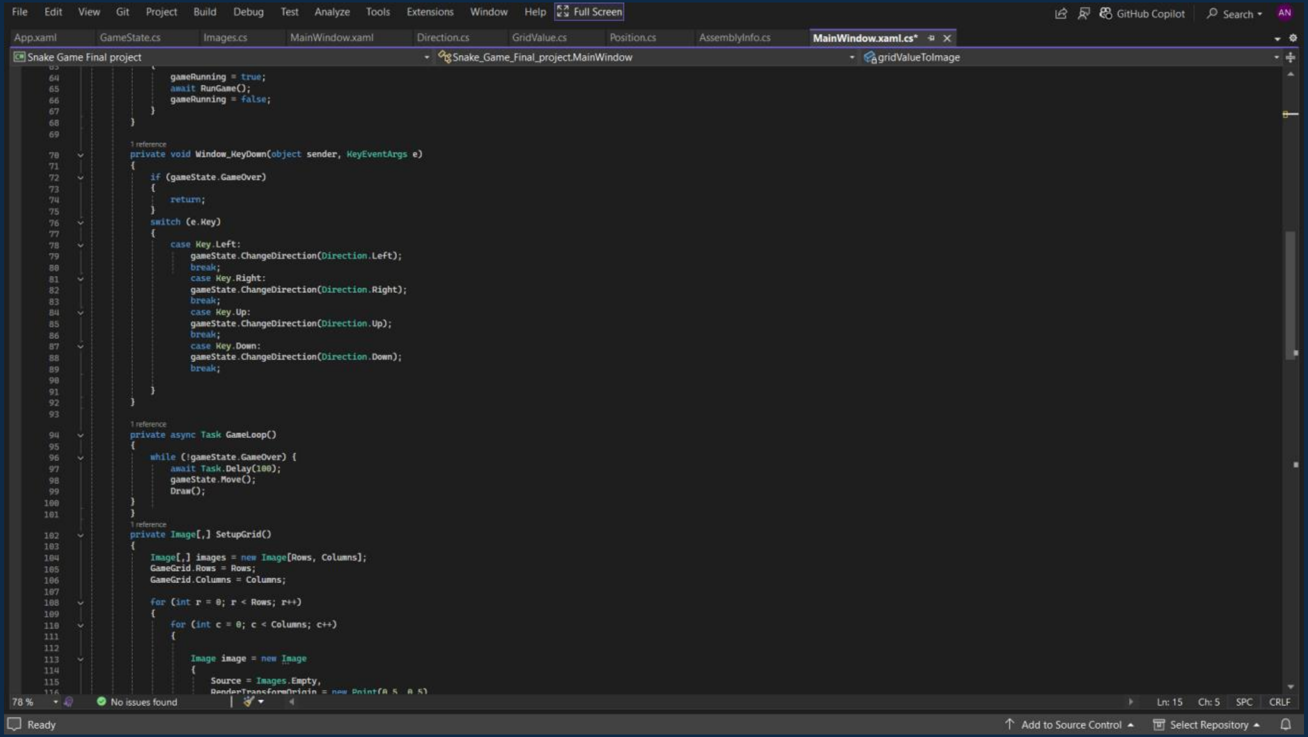
```
namespace Snake_Game_Final_project
{
    public class Direction
    {
        public readonly static Direction Left = new Direction(0, -1);
        public readonly static Direction Right = new Direction(0, 1);
        public readonly static Direction Up = new Direction(-1, 0);
        public readonly 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 static bool operator ==(Direction left, Direction right)
        {
            return EqualityComparer<Direction>.Default.Equals(left, right);
        }

        2 references
        public static bool operator !=(Direction left, Direction right)
        {
            return !(left == right);
        }
    }
}
```



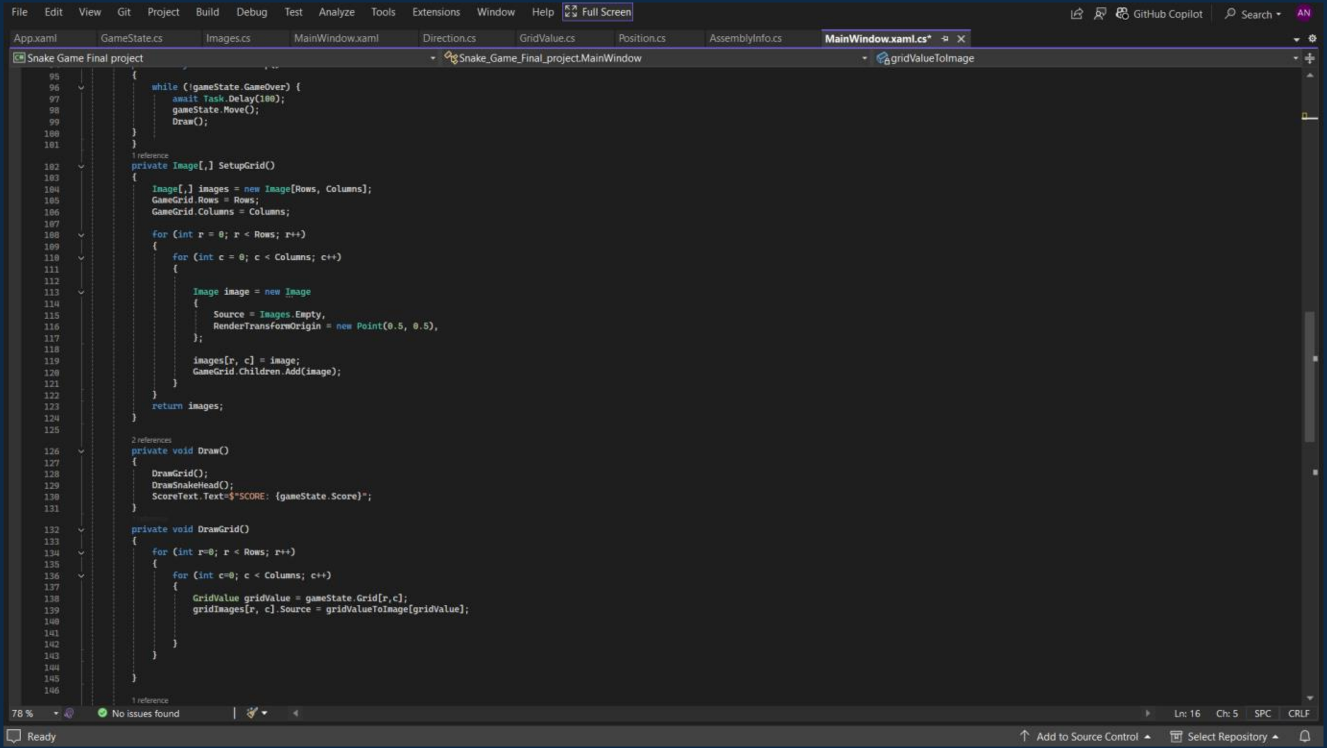
```csharp


namespace Snake_Game_Final_project
{
    public class Position
    {

        public int Row { get; }
        public int Column { get; }

        3 references
        public Position(int row, int column)
        {
            Row = row;
            Column = column;
        }

        public Position Translate(Direction dir)
        {
            return new Position(Row + dir.RowOffset, Column + dir.ColumnOffset);

        }

        public override bool Equals(object obj)
        {
            return obj is Position position &&
                   Row == position.Row &&
                   Column == position.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);
        }
    }
}
```



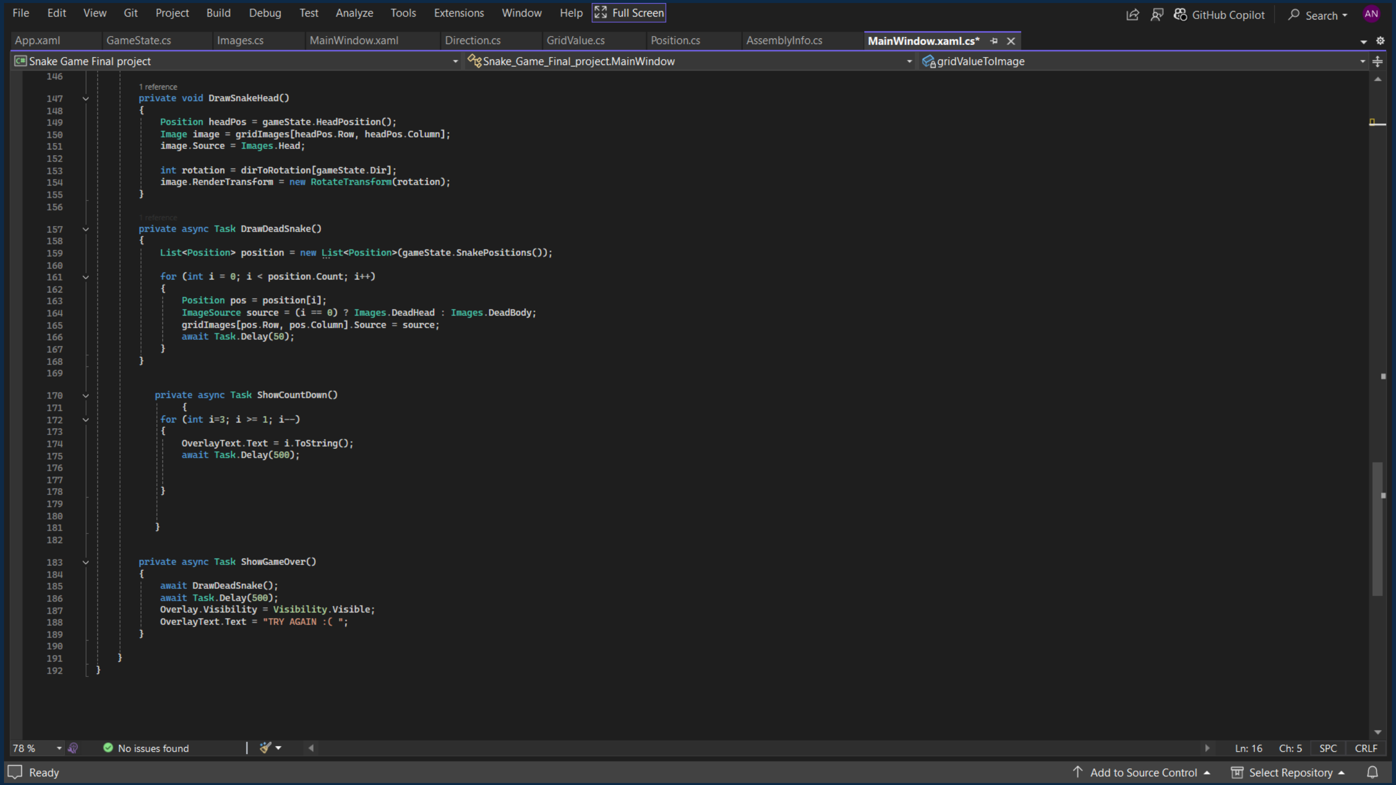
```csharp
namespace Snake_Game_Final_project
{

    4 references
    public partial class MainWindow : Window
    {
        private readonly Dictionary<GridValue, ImageSource> gridValueToImage = 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, Columns = 15;
        private readonly Image[,] gridImages;
        private GameState gameState;
        private bool gameRunning;
        0 references
        public MainWindow()
        {
            InitializeComponent();
            gridImages = SetupGrid();
            gameState = new GameState(Rows, Columns);
        }
        1 reference
        private async Task RunGame()
        {
            Draw();
            await ShowCountDown();
            Overlay.Visibility = Visibility.Hidden;
            await GameLoop();
            await ShowGameOver();
            gameState = new GameState(Rows, Columns);

        }

        1 reference
        private async void Window_PreviewKeyDown(object sender, KeyEventArgs e)
        {
            if (Overlay.Visibility == Visibility.Visible)
            {

                e.Handled = true;
            }

            if (!gameRunning)
            {
                gameRunning = true;
```



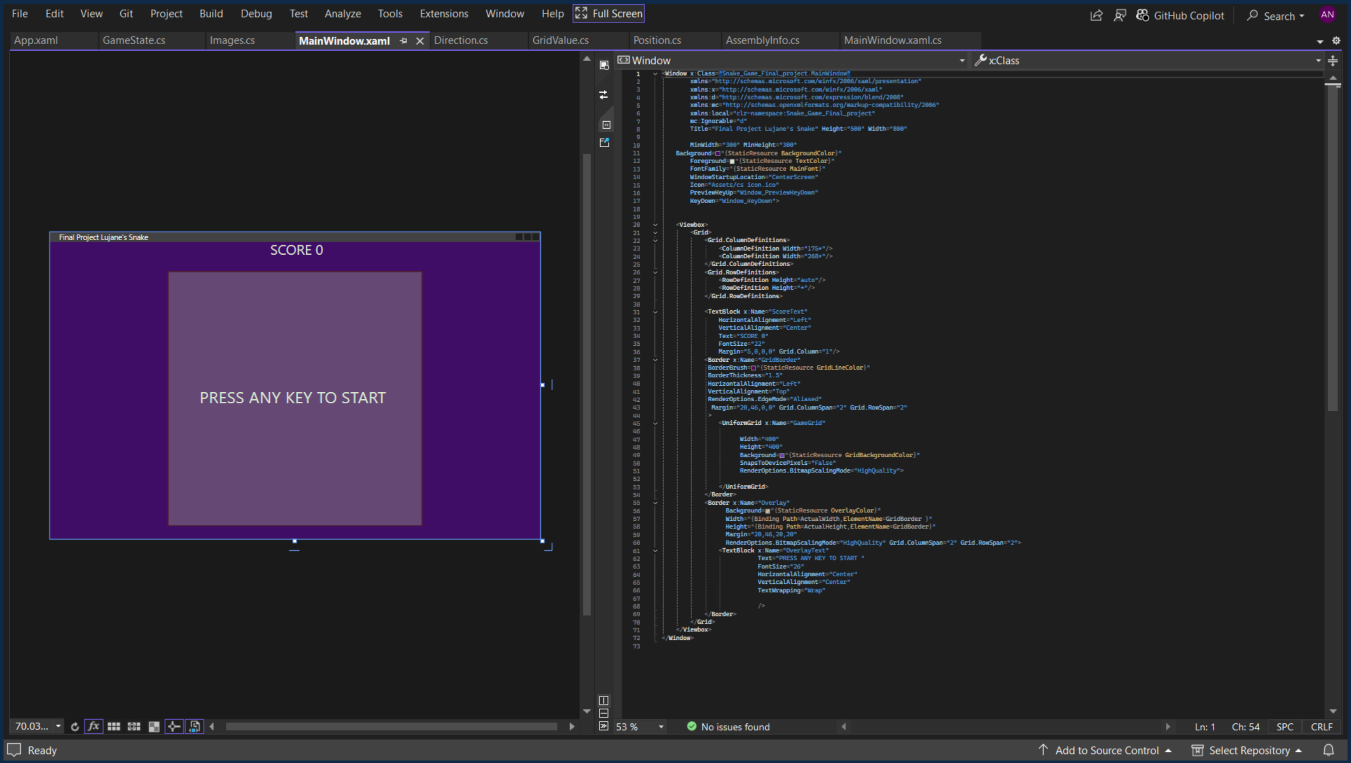
```
        gameRunning = true;
        await RunGame();
        gameRunning = false;
    }
}

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, Columns];
    GameGrid.Rows = Rows;
    GameGrid.Columns = Columns;

    for (int r = 0; r < Rows; r++)
    {
        for (int c = 0; c < Columns; c++)
        {

            Image image = new Image
            {
                Source = Images.Empty,
                RenderTransformOrigin = new Point(0.5, 0.5)
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
GitHub Copilot Search
App.xaml GameState.cs Images.cs MainWindow.xaml Direction.cs GridValue.cs Position.cs AssemblyInfo.cs MainWindow.xaml.cs*
Snake Game Final project
Snake_Game_Final_project.MainWindow
gridValueToImage
78 %
No issues found
Ln: 16 Ch: 5 SPC CRLF
Ready
Add to Source Control
Select Repository

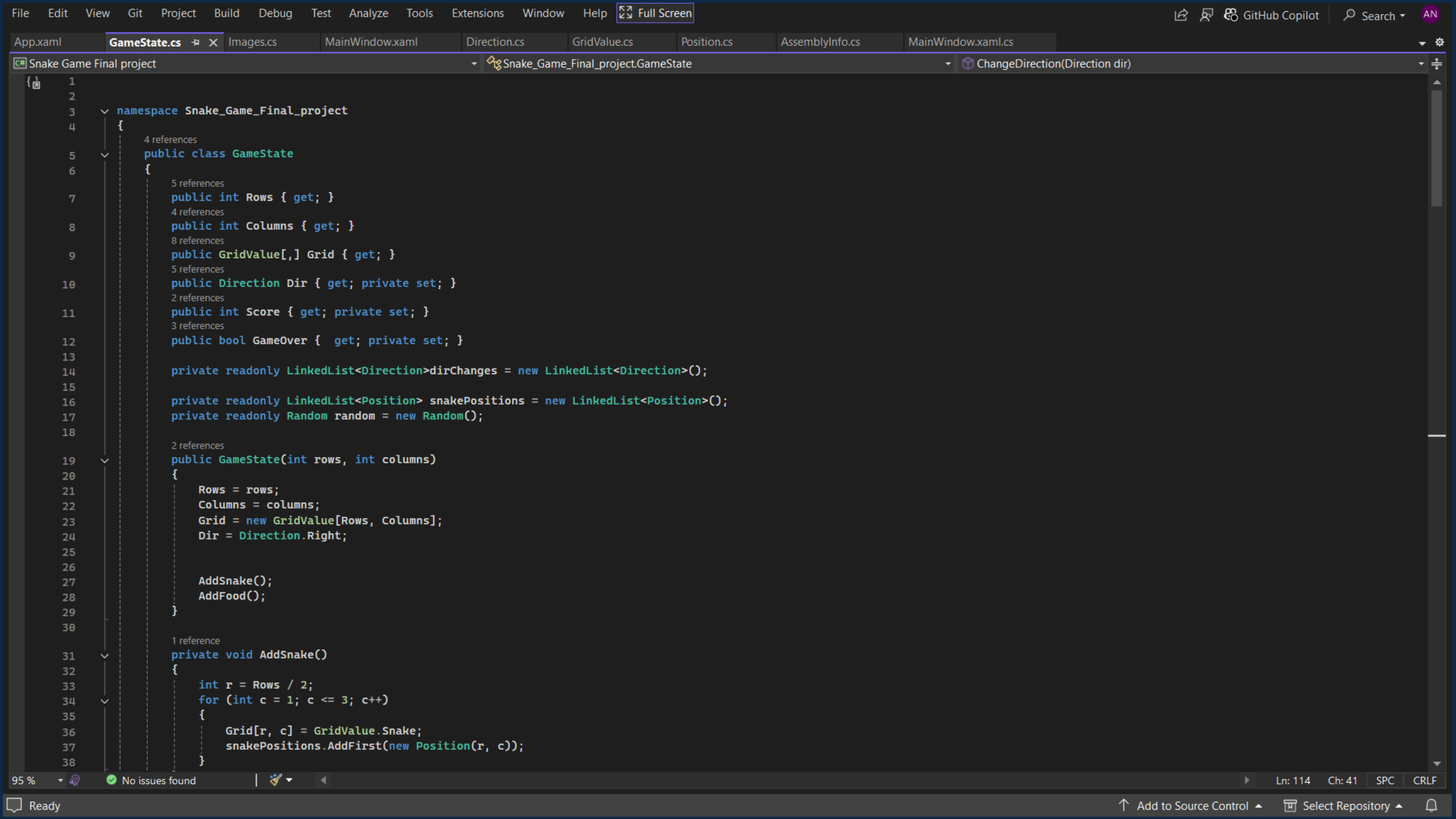
```
    {
        while (!gameState.GameOver) {
            await Task.Delay(100);
            gameState.Move();
            Draw();
        }
    }
    1 reference
    private Image[,] SetupGrid()
    {
        Image[,] images = new Image[Rows, Columns];
        GameGrid.Rows = Rows;
        GameGrid.Columns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

        for (int r = 0; r < Rows; r++)
        {
            for (int c = 0; c < Columns; 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;
    }

    2 references
    private void Draw()
    {
        DrawGrid();
        DrawSnakeHead();
        ScoreText.Text=$"SCORE: {gameState.Score}";
    }

    private void DrawGrid()
    {
        for (int r=0; r < Rows; r++)
        {
            for (int c=0; c < Columns; c++)
            {
                GridValue gridValue = gameState.Grid[r,c];
                gridImages[r, c].Source = gridValueToImage[gridValue];


            }
        }

    }

    1 reference
```



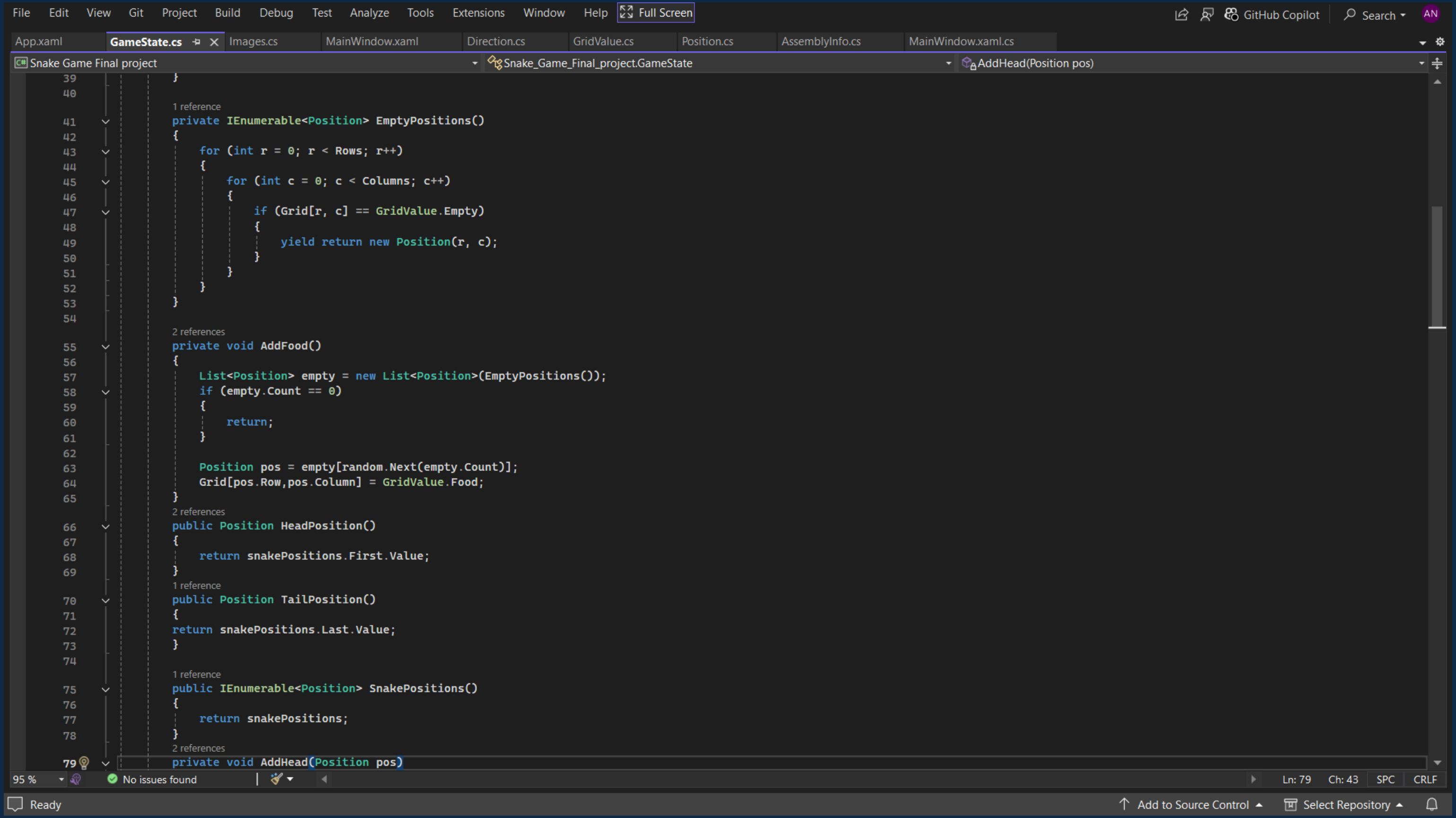
```
        1 reference
        private void DrawSnakeHead()
        {
            Position headPos = gameState.HeadPosition();
            Image image = gridImages[headPos.Row, headPos.Column];
            image.Source = Images.Head;

            int rotation = dirToRotation[gameState.Dir];
            image.RenderTransform = new RotateTransform(rotation);
        }

        1 reference
        private async Task DrawDeadSnake()
        {
            List<Position> position = new List<Position>(gameState.SnakePositions());

            for (int i = 0; i < position.Count; i++)
            {
                Position pos = position[i];
                ImageSource source = (i == 0) ? Images.DeadHead : Images.DeadBody;
                gridImages[pos.Row, pos.Column].Source = source;
                await Task.Delay(50);
            }
        }


           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(500);
            Overlay.Visibility = Visibility.Visible;
            OverlayText.Text = "TRY AGAIN :( ";
        }

    }
}
```



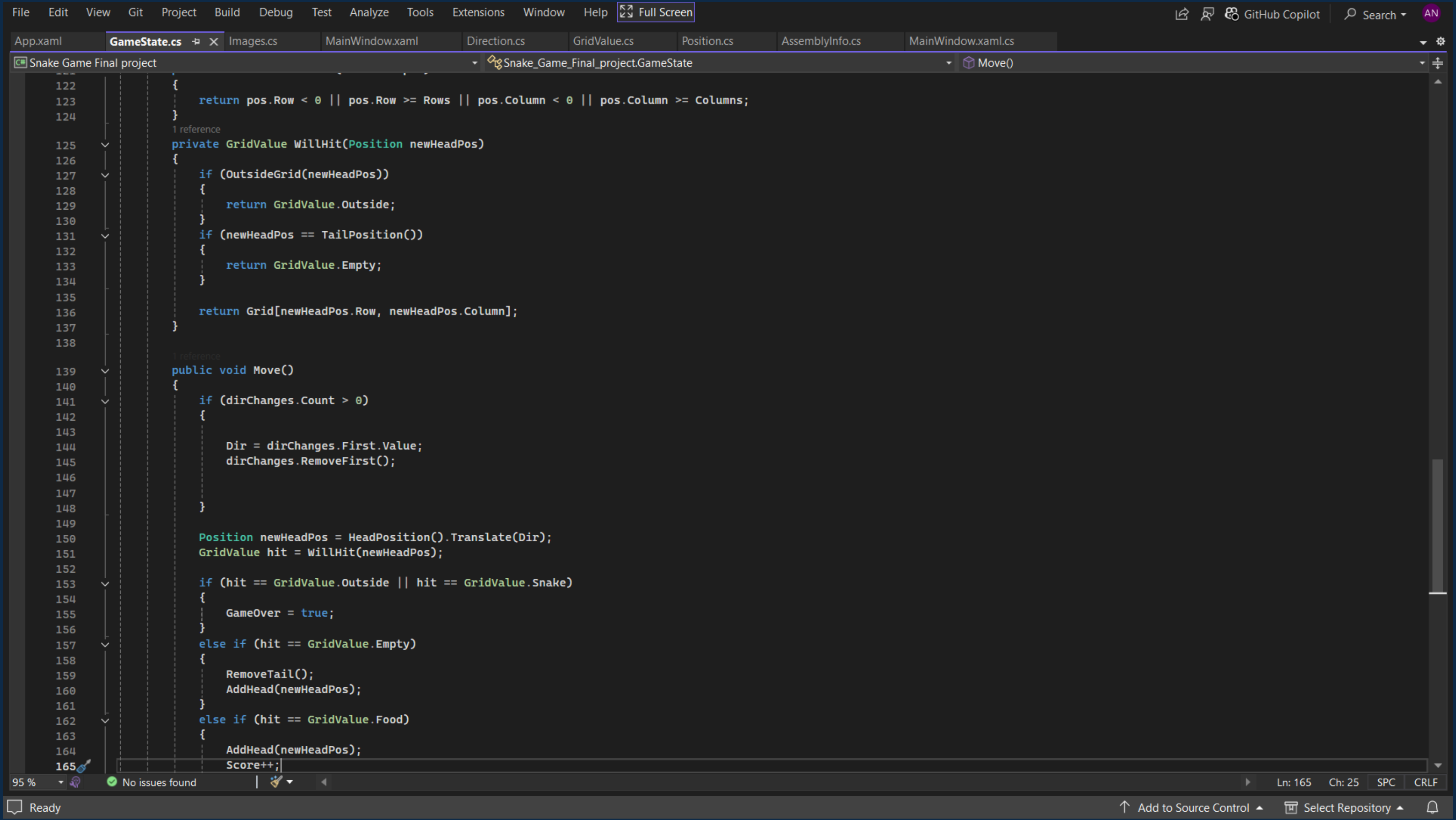
```xml
<Window x:Class="Snake_Game_Final_project.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Snake_Game_Final_project"
        mc:Ignorable="d"
        Title="Final Project Lujane's Snake" Height="500" Width="800"

        MinWidth="300" MinHeight="300"
    Background="{StaticResource BackgroundColor}"
        Foreground="{StaticResource TextColor}"
        FontFamily="{StaticResource MainFont}"
        WindowStartupLocation="CenterScreen"
        Icon="Assets/cs icon.ico"
        PreviewKeyUp="Window_PreviewKeyDown"
        KeyDown="Window_KeyDown">


    <Viewbox>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="175*"/>
                <ColumnDefinition Width="268*"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="auto"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <TextBlock x:Name="ScoreText"
                HorizontalAlignment="Left"
                VerticalAlignment="Center"
                Text="SCORE 0"
                FontSize="22"
                Margin="5,0,0,0" Grid.Column="1"/>
            <Border x:Name="GridBorder"
            BorderBrush="{StaticResource GridLineColor}"
            BorderThickness="1.5"
            HorizontalAlignment="Left"
            VerticalAlignment="Top"
            RenderOptions.EdgeMode="Aliased"
             Margin="20,46,0,0" Grid.ColumnSpan="2" Grid.RowSpan="2"
            >
                <UniformGrid x:Name="GameGrid"

                     Width="400"
                     Height="400"
                     Background="{StaticResource GridBackgroundColor}"
                     SnapsToDevicePixels="False"
                     RenderOptions.BitmapScalingMode="HighQuality">

                </UniformGrid>
            </Border>
            <Border x:Name="Overlay"
                  Background="{StaticResource OverlayColor}"
                  Width="{Binding Path=ActualWidth,ElementName=GridBorder }"
                  Height="{Binding Path=ActualHeight,ElementName=GridBorder}"
                  Margin="20,46,20,20"
                  RenderOptions.BitmapScalingMode="HighQuality" Grid.ColumnSpan="2" Grid.RowSpan="2">
                <TextBlock x:Name="OverlayText"
                         Text="PRESS ANY KEY TO START "
                         FontSize="26"
                         HorizontalAlignment="Center"
                         VerticalAlignment="Center"
                         TextWrapping="Wrap"

                         />
            </Border>
        </Grid>
    </Viewbox>
</Window>
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
GitHub Copilot Search
App.xaml GameState.cs Images.cs MainWindow.xaml Direction.cs GridValue.cs Position.cs AssemblyInfo.cs MainWindow.xaml.cs
Snake Game Final project
Snake_Game_Final_project.GameState
ChangeDirection(Direction dir)
95 % No issues found Ln: 114 Ch: 41 SPC CRLF
Ready Add to Source Control Select Repository

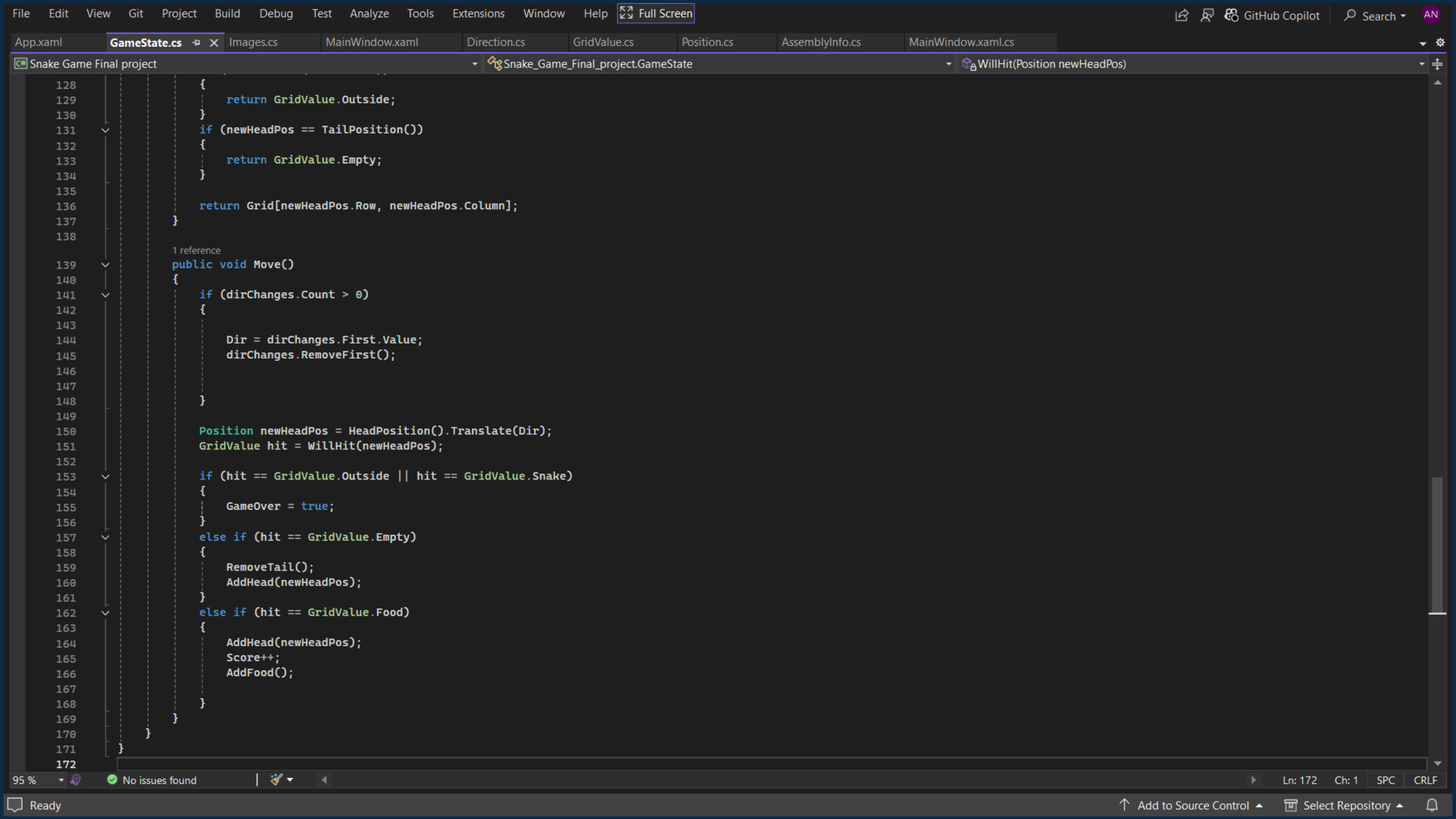
```


namespace Snake_Game_Final_project
{
    4 references
    public class GameState
    {
        5 references
        public int Rows { get; }
        4 references
        public int Columns { get; }
        8 references
        public GridValue[,] Grid { get; }
        5 references
        public Direction Dir { get; private set; }
        2 references
        public int Score { get; private set; }
        3 references
        public bool GameOver {  get; private set; }

        private readonly LinkedList<Direction>dirChanges = new LinkedList<Direction>();

        private readonly LinkedList<Position> snakePositions = new LinkedList<Position>();
        private readonly Random random = new Random();

        2 references
        public GameState(int rows, int columns)
        {
            Rows = rows;
            Columns = columns;
            Grid = new GridValue[Rows, Columns];
            Dir = Direction.Right;


            AddSnake();
            AddFood();
        }

        1 reference
        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));
            }
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
GitHub Copilot
Search
App.xaml GameState.cs Images.cs MainWindow.xaml Direction.cs GridValue.cs Position.cs AssemblyInfo.cs MainWindow.xaml.cs
Snake Game Final project
Snake_Game_Final_project.GameState
AddHead(Position pos)
95 %
No issues found
Ln: 79 Ch: 43 SPC CRLF
Ready
Add to Source Control
Select Repository

```
        }

        1 reference
        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);
                    }
                }
            }
        }

        2 references
        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;
        }
        2 references
        public Position HeadPosition()
        {
            return snakePositions.First.Value;
        }
        1 reference
        public Position TailPosition()
        {
        return snakePositions.Last.Value;
        }

        1 reference
        public IEnumerable<Position> SnakePositions()
        {
            return snakePositions;
        }
        2 references
        private void AddHead(Position pos)
```


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
GitHub Copilot Search AN
App.xaml GameState.cs Images.cs MainWindow.xaml Direction.cs GridValue.cs Position.cs AssemblyInfo.cs MainWindow.xaml.cs
Snake Game Final project
Snake_Game_Final_project.GameState
Move()
95 % No issues found Ln: 165 Ch: 25 SPC CRLF
Ready Add to Source Control Select Repository

```
        {
            return pos.Row < 0 || pos.Row >= Rows || pos.Column < 0 || pos.Column >= Columns;
        }
        1 reference
        private GridValue WillHit(Position newHeadPos)
        {
            if (OutsideGrid(newHeadPos))
            {
                return GridValue.Outside;
            }
            if (newHeadPos == TailPosition())
            {
                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);
            }
            else if (hit == GridValue.Food)
            {
                AddHead(newHeadPos);
                Score++;
```

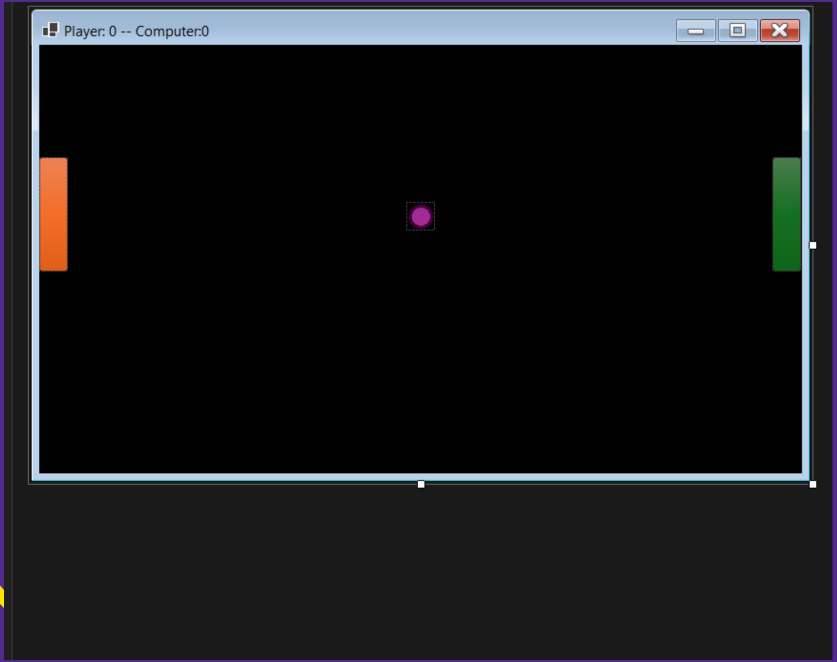

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Full Screen
GitHub Copilot
Search
App.xaml GameState.cs Images.cs MainWindow.xaml Direction.cs GridValue.cs Position.cs AssemblyInfo.cs MainWindow.xaml.cs
Snake Game Final project
Snake_Game_Final_project.GameState
WillHit(Position newHeadPos)
95 %
No issues found
Ln: 172 Ch: 1 SPC CRLF
Ready
Add to Source Control
Select Repository

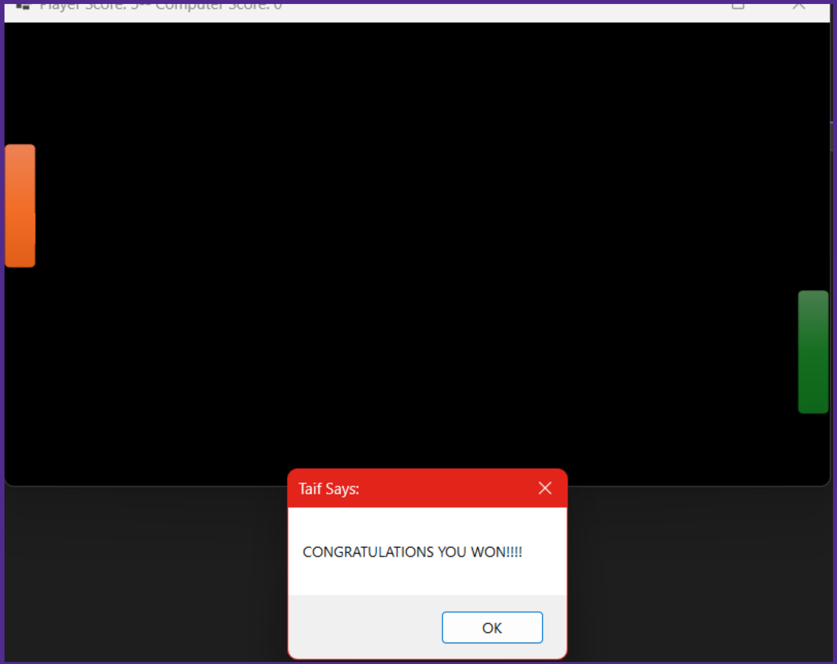
```
            {
                return GridValue.Outside;
            }
            if (newHeadPos == TailPosition())
            {
                return GridValue.Empty;
            }

            return Grid[newHeadPos.Row, newHeadPos.Column];
        }

        1 reference
        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);
            }
            else if (hit == GridValue.Food)
            {
                AddHead(newHeadPos);
                Score++;
                AddFood();

            }
        }
    }
}
```

04.

Ping Pong Game

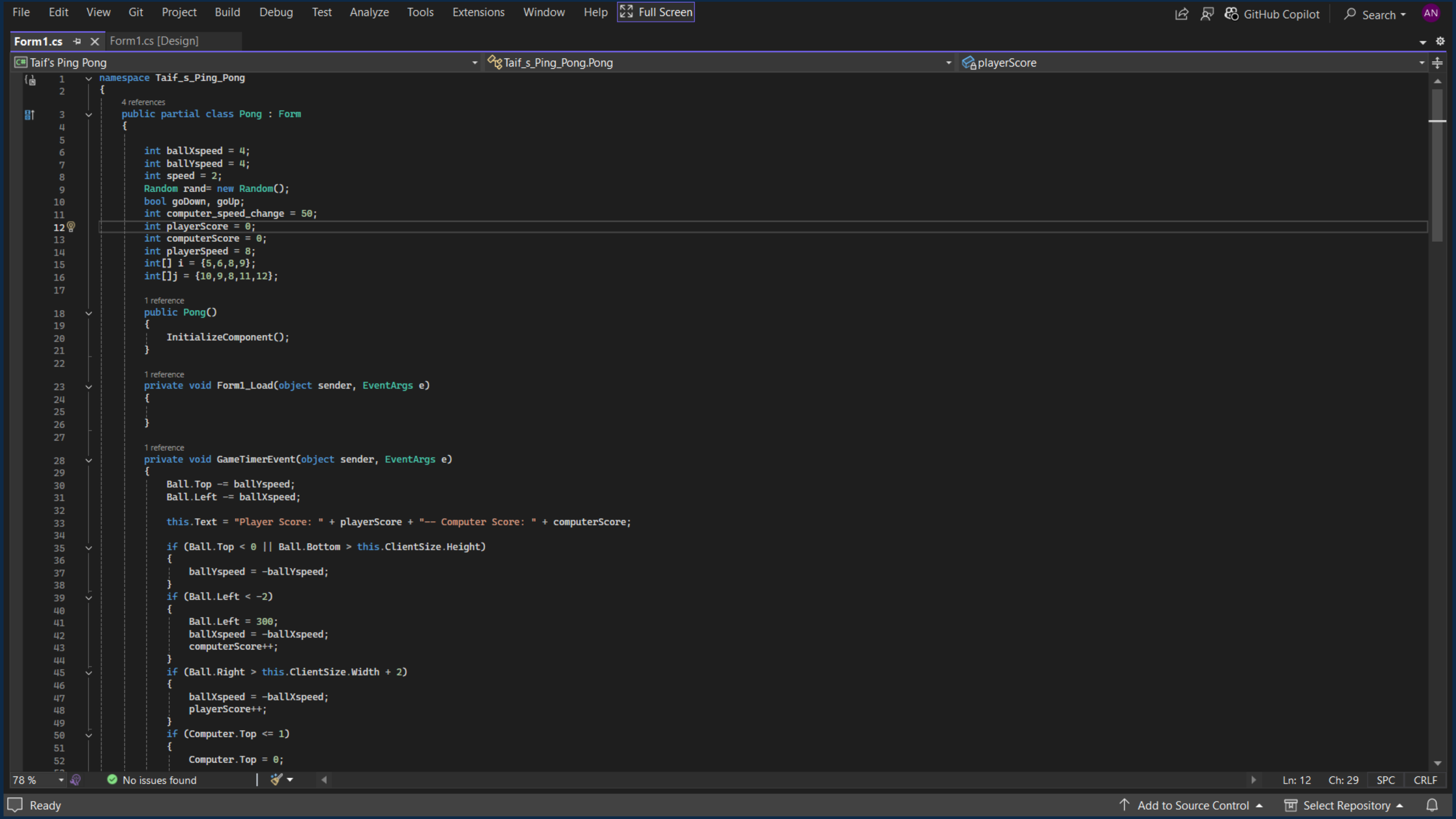
recreated in **C#** using WPF, where the player **controls** a **paddle** to **bounce** a ball back and forth against an opposing paddle. The objective is to **prevent** the ball from passing the **player's** paddle while attempting to score points by making the ball pass the opponent's paddle. This project demonstrates **core programming skills**, such as real-time object **movement**, collision detection, and **handling** game **logic** for scoring and paddle control. With its fast-paced gameplay, it offers an **exciting** and challenging experience that tests a player’s reflexes, timing, and **strategy**.



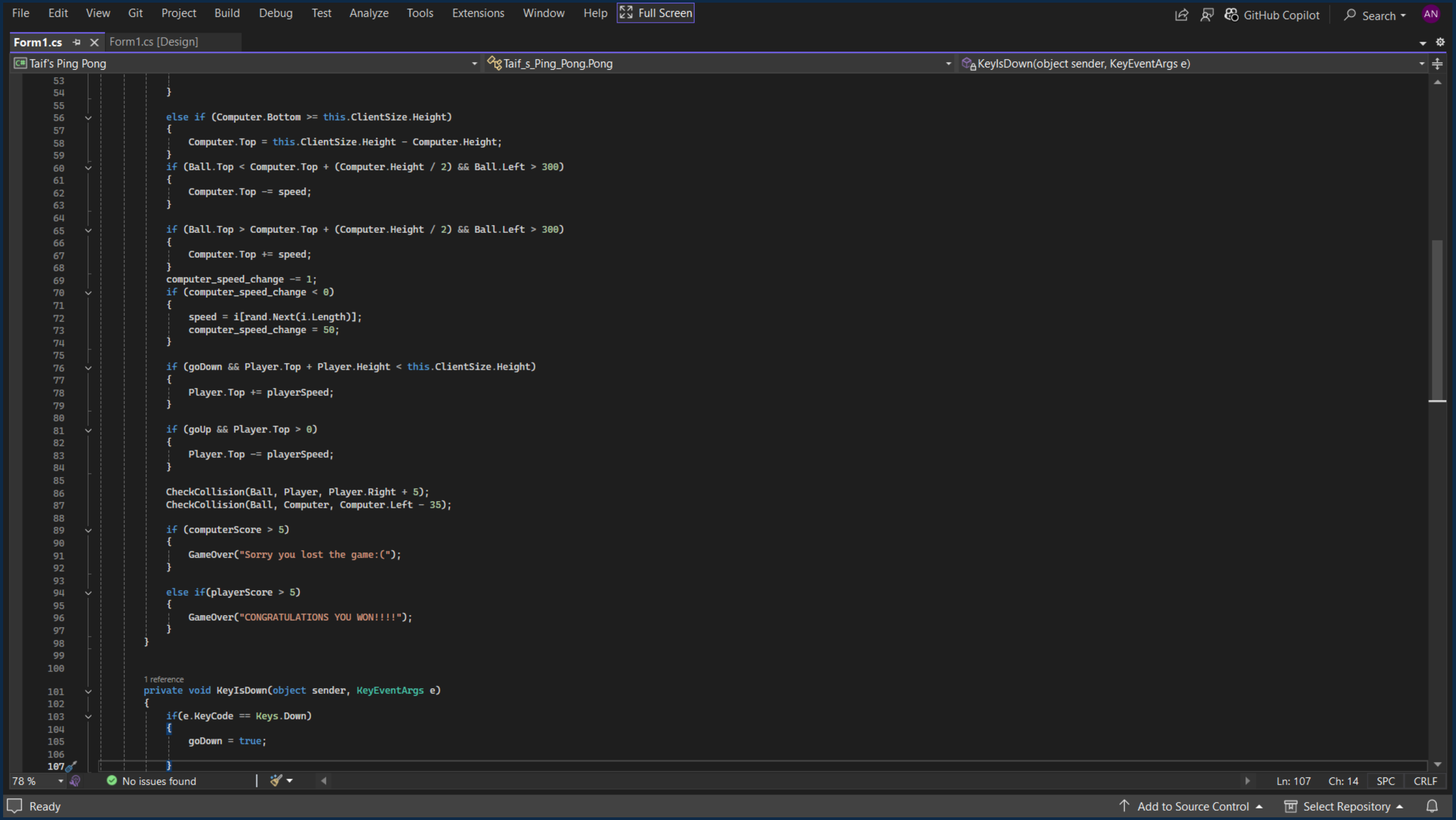


Taif Says:

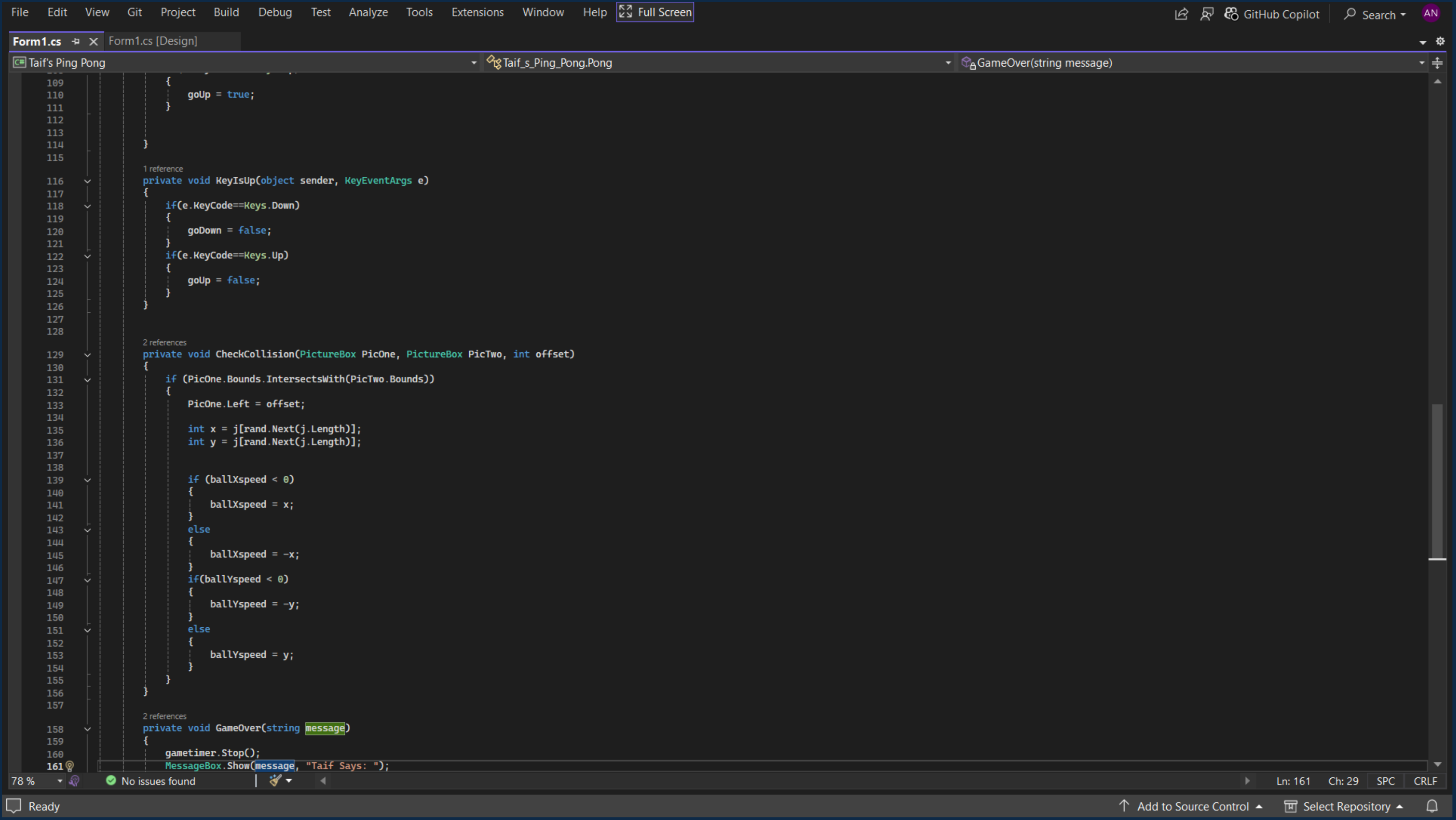
CONGRATULATIONS YOU WON!!!!



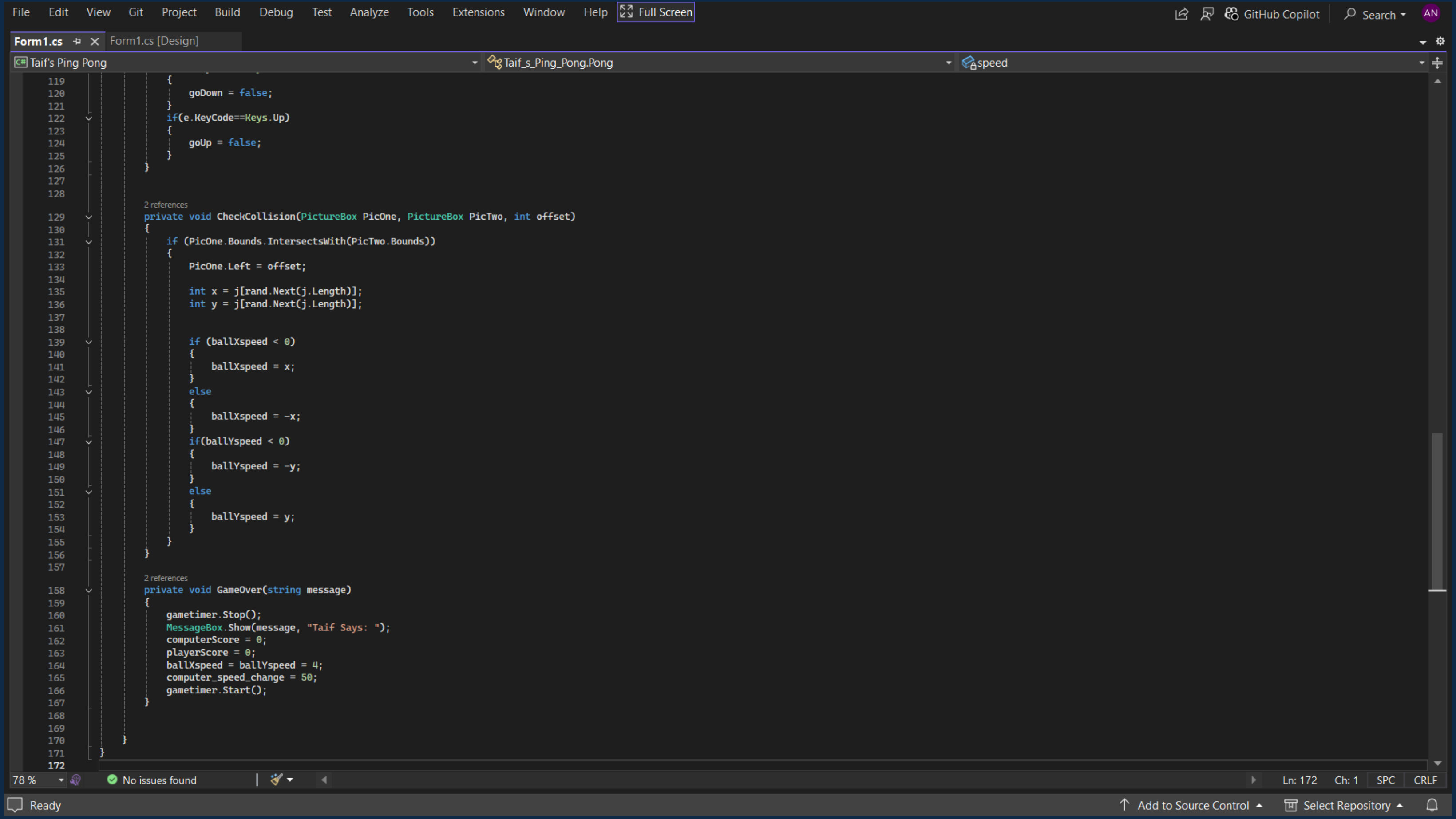
```
namespace Taif_s_Ping_Pong
{
    4 references
    public partial class Pong : Form
    {

        int ballXspeed = 4;
        int ballYspeed = 4;
        int speed = 2;
        Random rand= new Random();
        bool goDown, goUp;
        int computer_speed_change = 50;
        int playerScore = 0;
        int computerScore = 0;
        int playerSpeed = 8;
        int[] i = {5,6,8,9};
        int[]j = {10,9,8,11,12};

        1 reference
        public Pong()
        {
            InitializeComponent();
        }

        1 reference
        private void Form1_Load(object sender, EventArgs e)
        {

        }

        1 reference
        private void GameTimerEvent(object sender, EventArgs e)
        {
            Ball.Top -= ballYspeed;
            Ball.Left -= ballXspeed;

            this.Text = "Player Score: " + playerScore + "-- Computer Score: " + computerScore;

            if (Ball.Top < 0 || Ball.Bottom > this.ClientSize.Height)
            {
                ballYspeed = -ballYspeed;
            }
            if (Ball.Left < -2)
            {
                Ball.Left = 300;
                ballXspeed = -ballXspeed;
                computerScore++;
            }
            if (Ball.Right > this.ClientSize.Width + 2)
            {
                ballXspeed = -ballXspeed;
                playerScore++;
            }
            if (Computer.Top <= 1)
            {
                Computer.Top = 0;
```



```
            }

            else if (Computer.Bottom >= this.ClientSize.Height)
            {
                Computer.Top = this.ClientSize.Height - Computer.Height;
            }
            if (Ball.Top < Computer.Top + (Computer.Height / 2) && Ball.Left > 300)
            {
                Computer.Top -= speed;
            }

            if (Ball.Top > Computer.Top + (Computer.Height / 2) && Ball.Left > 300)
            {
                Computer.Top += speed;
            }
            computer_speed_change -= 1;
            if (computer_speed_change < 0)
            {
                speed = i[rand.Next(i.Length)];
                computer_speed_change = 50;
            }

            if (goDown && Player.Top + Player.Height < this.ClientSize.Height)
            {
                Player.Top += playerSpeed;
            }

            if (goUp && Player.Top > 0)
            {
                Player.Top -= playerSpeed;
            }

            CheckCollision(Ball, Player, Player.Right + 5);
            CheckCollision(Ball, Computer, Computer.Left - 35);

            if (computerScore > 5)
            {
                GameOver("Sorry you lost the game:(");
            }

            else if(playerScore > 5)
            {
                GameOver("CONGRATULATIONS YOU WON!!!!");
            }
        }


        1 reference
        private void KeyIsDown(object sender, KeyEventArgs e)
        {
            if(e.KeyCode == Keys.Down)
            {
                goDown = true;

            }
```



```
            {
                goUp = true;
            }


        }

        private void KeyIsUp(object sender, KeyEventArgs e)
        {
            if(e.KeyCode==Keys.Down)
            {
                goDown = false;
            }
            if(e.KeyCode==Keys.Up)
            {
                goUp = false;
            }
        }


        private void CheckCollision(PictureBox PicOne, PictureBox PicTwo, int offset)
        {
            if (PicOne.Bounds.IntersectsWith(PicTwo.Bounds))
            {
                PicOne.Left = offset;

                int x = j[rand.Next(j.Length)];
                int y = j[rand.Next(j.Length)];


                if (ballXspeed < 0)
                {
                    ballXspeed = x;
                }
                else
                {
                    ballXspeed = -x;
                }
                if(ballYspeed < 0)
                {
                    ballYspeed = -y;
                }
                else
                {
                    ballYspeed = y;
                }
            }
        }

        private void GameOver(string message)
        {
            gametimer.Stop();
            MessageBox.Show(message, "Taif Says: ");
```



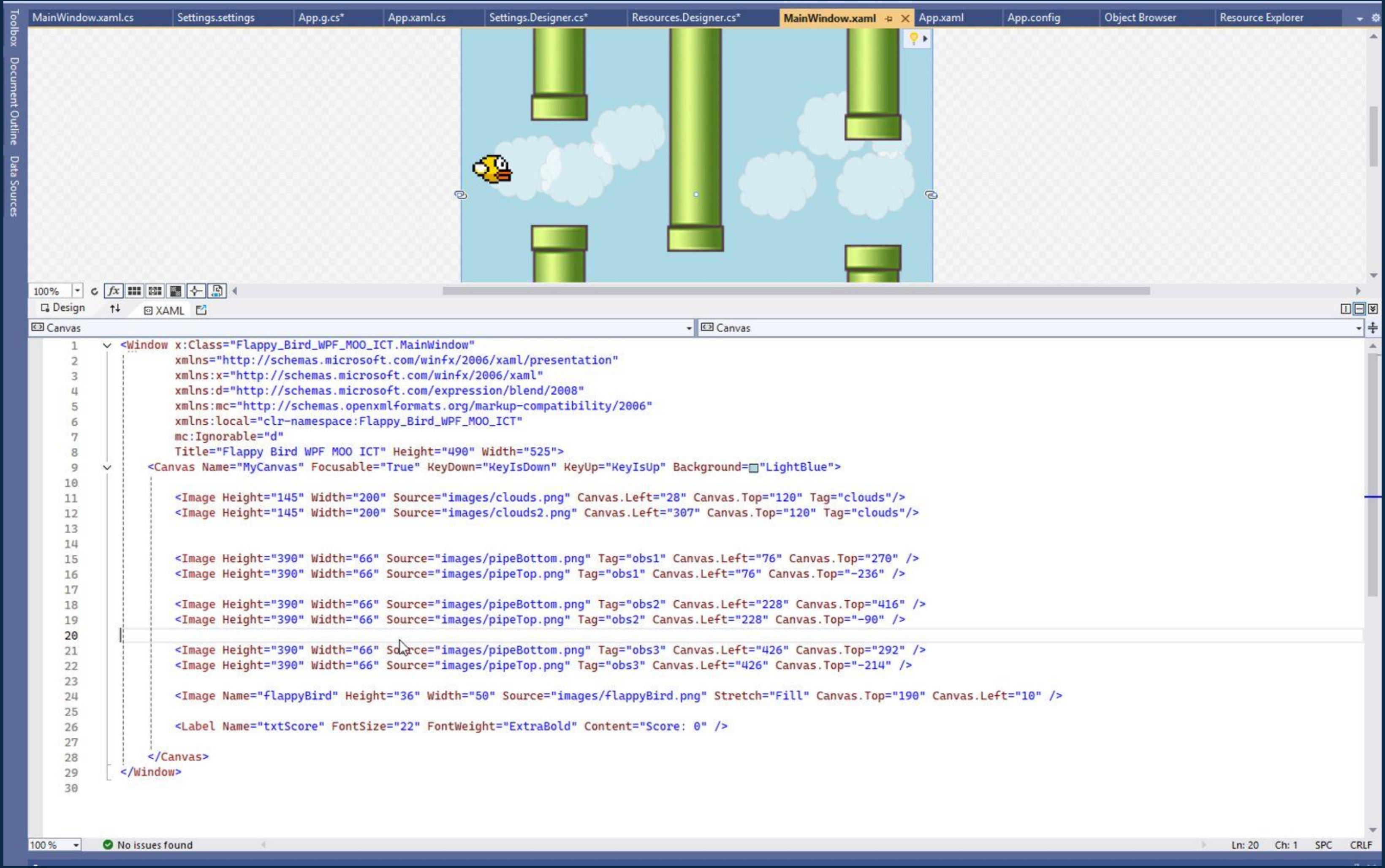
```
            {
                goDown = false;
            }
            if(e.KeyCode==Keys.Up)
            {
                goUp = false;
            }
        }


        2 references
        private void CheckCollision(PictureBox PicOne, PictureBox PicTwo, int offset)
        {
            if (PicOne.Bounds.IntersectsWith(PicTwo.Bounds))
            {
                PicOne.Left = offset;

                int x = j[rand.Next(j.Length)];
                int y = j[rand.Next(j.Length)];


                if (ballXspeed < 0)
                {
                    ballXspeed = x;
                }
                else
                {
                    ballXspeed = -x;
                }
                if(ballYspeed < 0)
                {
                    ballYspeed = -y;
                }
                else
                {
                    ballYspeed = y;
                }
            }
        }

        2 references
        private void GameOver(string message)
        {
            gametimer.Stop();
            MessageBox.Show(message, "Taif Says: ");
            computerScore = 0;
            playerScore = 0;
            ballXspeed = ballYspeed = 4;
            computer_speed_change = 50;
            gametimer.Start();
        }


    }
}
```

.05

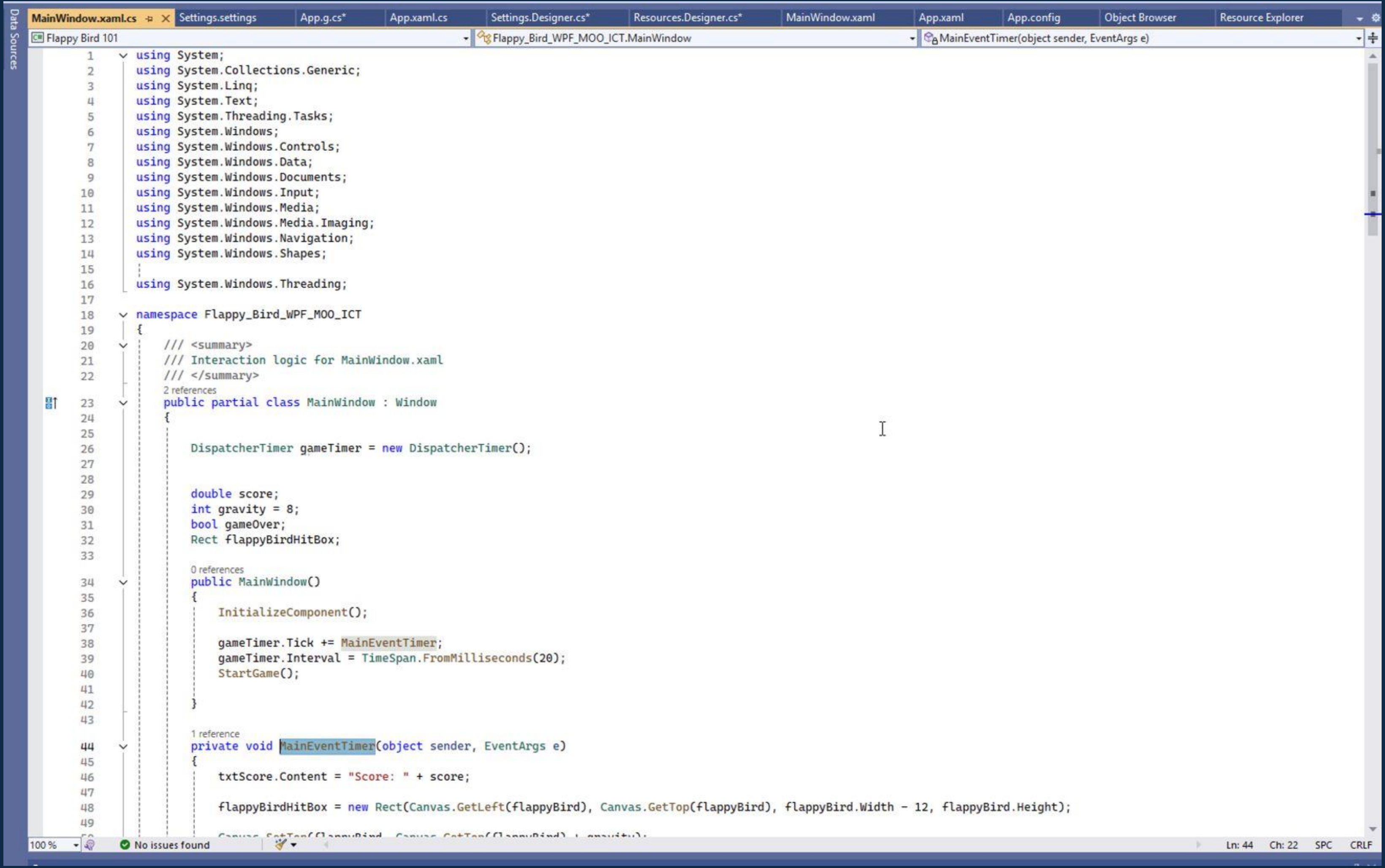
Floppy Bird Game

The **Flappy Bird game** is a popular arcade-style game recreated using C# and WPF, where the player controls a bird that must **navigate** through a series of **pipes**. The bird is **continuously** falling due to gravity, and the player must tap the screen (or press a key) to make the bird "flap" and rise **temporarily**. The goal is to fly through the gaps in the pipes without **colliding** with them, while **avoiding** falling to the ground. This game demonstrates key concepts in game development, such as gravity simulation, collision detection, and **continuous** gameplay, offering an engaging and fast-paced challenge.

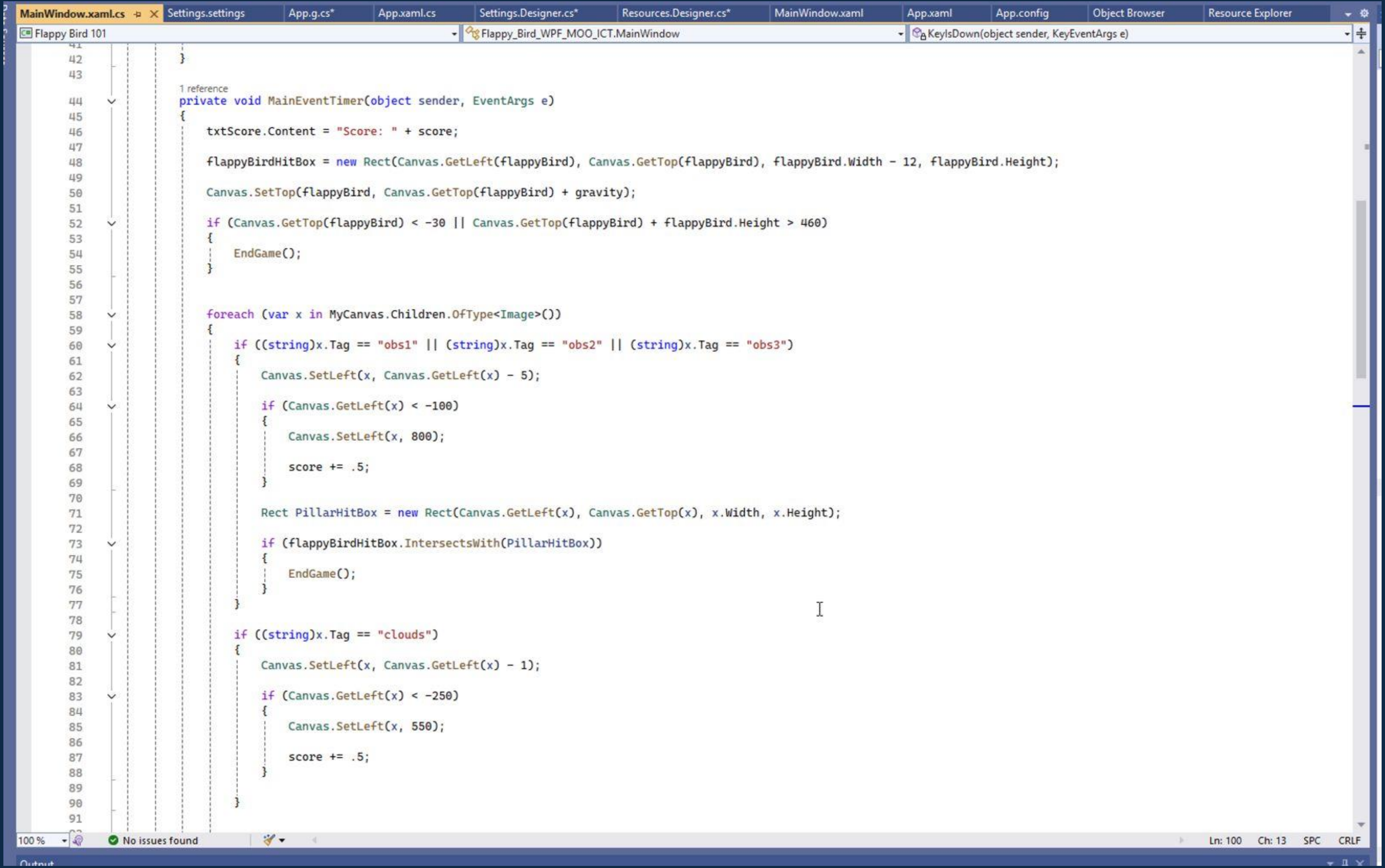
Flappy Bird
CT
Score: 0



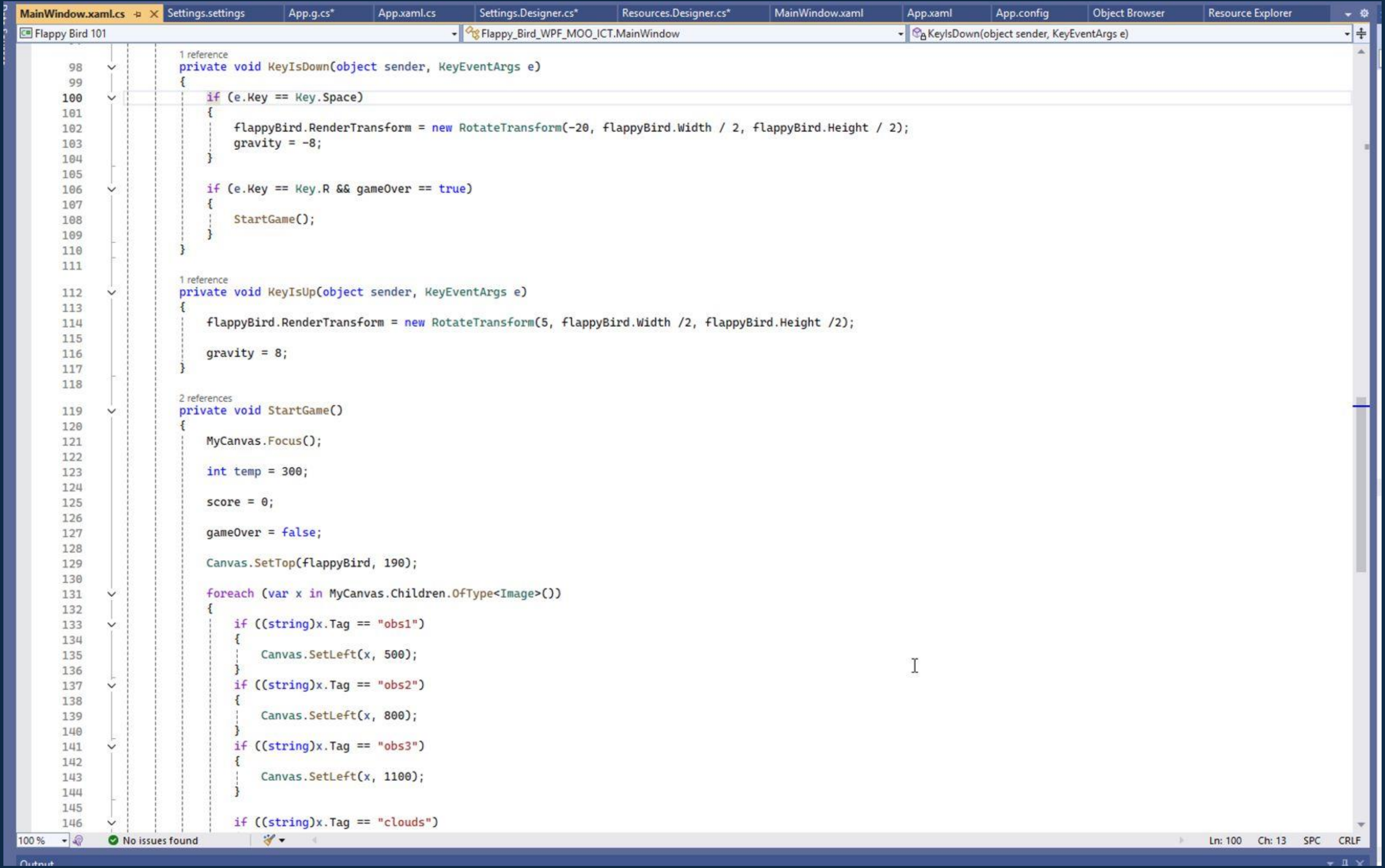
```xml
<Window x:Class="Flappy_Bird_WPF_MOO_ICT.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Flappy_Bird_WPF_MOO_ICT"
        mc:Ignorable="d"
        Title="Flappy Bird WPF MOO ICT" Height="490" Width="525">
    <Canvas Name="MyCanvas" Focusable="True" KeyDown="KeyIsDown" KeyUp="KeyIsUp" Background="LightBlue">

        <Image Height="145" Width="200" Source="images/clouds.png" Canvas.Left="28" Canvas.Top="120" Tag="clouds"/>
        <Image Height="145" Width="200" Source="images/clouds2.png" Canvas.Left="307" Canvas.Top="120" Tag="clouds"/>


        <Image Height="390" Width="66" Source="images/pipeBottom.png" Tag="obs1" Canvas.Left="76" Canvas.Top="270" />
        <Image Height="390" Width="66" Source="images/pipeTop.png" Tag="obs1" Canvas.Left="76" Canvas.Top="-236" />

        <Image Height="390" Width="66" Source="images/pipeBottom.png" Tag="obs2" Canvas.Left="228" Canvas.Top="416" />
        <Image Height="390" Width="66" Source="images/pipeTop.png" Tag="obs2" Canvas.Left="228" Canvas.Top="-90" />

        <Image Height="390" Width="66" Source="images/pipeBottom.png" Tag="obs3" Canvas.Left="426" Canvas.Top="292" />
        <Image Height="390" Width="66" Source="images/pipeTop.png" Tag="obs3" Canvas.Left="426" Canvas.Top="-214" />

        <Image Name="flappyBird" Height="36" Width="50" Source="images/flappyBird.png" Stretch="Fill" Canvas.Top="190" Canvas.Left="10" />

        <Label Name="txtScore" FontSize="22" FontWeight="ExtraBold" Content="Score: 0" />

    </Canvas>
</Window>
```



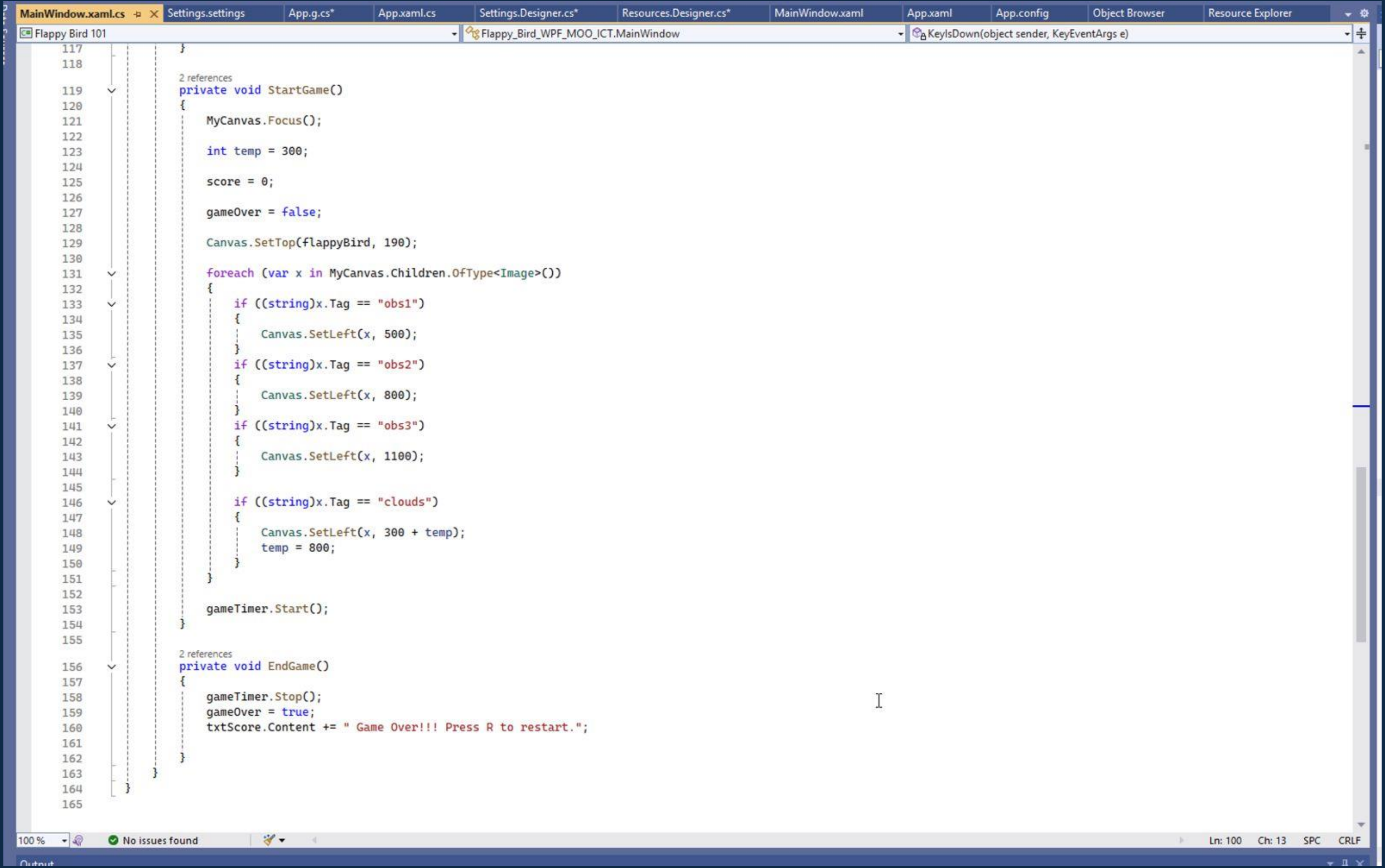
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;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

using System.Windows.Threading;

namespace Flappy_Bird_WPF_MOO_ICT
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {

        DispatcherTimer gameTimer = new DispatcherTimer();


        double score;
        int gravity = 8;
        bool gameOver;
        Rect flappyBirdHitBox;

        0 references
        public MainWindow()
        {
            InitializeComponent();

            gameTimer.Tick += MainEventTimer;
            gameTimer.Interval = TimeSpan.FromMilliseconds(20);
            StartGame();

        }

        1 reference
        private void MainEventTimer(object sender, EventArgs e)
        {
            txtScore.Content = "Score: " + score;

            flappyBirdHitBox = new Rect(Canvas.GetLeft(flappyBird), Canvas.GetTop(flappyBird), flappyBird.Width - 12, flappyBird.Height);

            Canvas.SetTop(flappyBird, Canvas.GetTop(flappyBird) + gravity);
```



```
        }

        1 reference
        private void MainEventTimer(object sender, EventArgs e)
        {
            txtScore.Content = "Score: " + score;

            flappyBirdHitBox = new Rect(Canvas.GetLeft(flappyBird), Canvas.GetTop(flappyBird), flappyBird.Width - 12, flappyBird.Height);

            Canvas.SetTop(flappyBird, Canvas.GetTop(flappyBird) + gravity);

            if (Canvas.GetTop(flappyBird) < -30 || Canvas.GetTop(flappyBird) + flappyBird.Height > 460)
            {
                EndGame();
            }


            foreach (var x in MyCanvas.Children.OfType<Image>())
            {
                if ((string)x.Tag == "obs1" || (string)x.Tag == "obs2" || (string)x.Tag == "obs3")
                {
                    Canvas.SetLeft(x, Canvas.GetLeft(x) - 5);

                    if (Canvas.GetLeft(x) < -100)
                    {
                        Canvas.SetLeft(x, 800);

                        score += .5;
                    }

                    Rect PillarHitBox = new Rect(Canvas.GetLeft(x), Canvas.GetTop(x), x.Width, x.Height);

                    if (flappyBirdHitBox.IntersectsWith(PillarHitBox))
                    {
                        EndGame();
                    }
                }

                if ((string)x.Tag == "clouds")
                {
                    Canvas.SetLeft(x, Canvas.GetLeft(x) - 1);

                    if (Canvas.GetLeft(x) < -250)
                    {
                        Canvas.SetLeft(x, 550);

                        score += .5;
                    }

                }

```



```
        1 reference
        private void KeyIsDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Space)
            {
                flappyBird.RenderTransform = new RotateTransform(-20, flappyBird.Width / 2, flappyBird.Height / 2);
                gravity = -8;
            }

            if (e.Key == Key.R && gameOver == true)
            {
                StartGame();
            }
        }

        1 reference
        private void KeyIsUp(object sender, KeyEventArgs e)
        {
            flappyBird.RenderTransform = new RotateTransform(5, flappyBird.Width /2, flappyBird.Height /2);

            gravity = 8;
        }

        2 references
        private void StartGame()
        {
            MyCanvas.Focus();

            int temp = 300;

            score = 0;

            gameOver = false;

            Canvas.SetTop(flappyBird, 190);

            foreach (var x in MyCanvas.Children.OfType<Image>())
            {
                if ((string)x.Tag == "obs1")
                {
                    Canvas.SetLeft(x, 500);
                }
                if ((string)x.Tag == "obs2")
                {
                    Canvas.SetLeft(x, 800);
                }
                if ((string)x.Tag == "obs3")
                {
                    Canvas.SetLeft(x, 1100);
                }

                if ((string)x.Tag == "clouds")
```


MainWindow.xaml.cs | Settings.settings | App.g.cs* | App.xaml.cs | Settings.Designer.cs* | Resources.Designer.cs* | MainWindow.xaml | App.xaml | App.config | Object Browser | Resource Explorer

Flappy Bird 101 | Flappy_Bird_WPF_MOO_ICT.MainWindow | KeyIsDown(object sender, KeyEventArgs e)

```
        }

        2 references
        private void StartGame()
        {
            MyCanvas.Focus();

            int temp = 300;

            score = 0;

            gameOver = false;

            Canvas.SetTop(flappyBird, 190);

            foreach (var x in MyCanvas.Children.OfType<Image>())
            {
                if ((string)x.Tag == "obs1")
                {
                    Canvas.SetLeft(x, 500);
                }
                if ((string)x.Tag == "obs2")
                {
                    Canvas.SetLeft(x, 800);
                }
                if ((string)x.Tag == "obs3")
                {
                    Canvas.SetLeft(x, 1100);
                }

                if ((string)x.Tag == "clouds")
                {
                    Canvas.SetLeft(x, 300 + temp);
                    temp = 800;
                }
            }

            gameTimer.Start();
        }

        2 references
        private void EndGame()
        {
            gameTimer.Stop();
            gameOver = true;
            txtScore.Content += " Game Over!!! Press R to restart.";

        }
    }
}
```

100 % | No issues found | Ln: 100 Ch: 13 SPC CRLF

Output

.06
Conclusion

In **conclusion**, developing these four games Car Racing, Snake, Ping Pong, and Flappy Bird, using C# and WPF has provided valuable insights into key **programming** concepts such as object-oriented programming, event handling, and game **mechanics**. Each game presents a **unique** set of challenges, from **implementing** smooth **movement** controls in the car racing game to managing the growing snake and detecting collisions in the snake game. The ping pong game highlights the importance of **physics** simulation and player **interaction**, while the Flappy Bird game focuses on **gravity** mechanics and **continuous** gameplay. By working on these games, we gained hands-on **experience** in game design, **problem-solving**, and **creative** thinking. This project not only **enhanced** our technical skills but also deepened our understanding of how to build interactive and engaging **user experiences** using C#.

THANKS!

Do you have any questions?

Daad Awad bin Tuwalah
201806163
Lujane Fawaz
202000637
Taif Ali Alqahtani
202006984
Raghad katb abdullah
202002115