Lab Assignment Report 11

CS202: Software Tools and Techniques for CSE

Roll Number: 22110087
Name: Parth Govale

Contents

1 Introduction

This lab focuses on analyzing C# console game applications using Visual Studio 2022. The objectives are to examine control flow using debugger operations, identify bugs that cause crashes, and fix those bugs. Two console game projects from dotnet-console-games were selected for this assignment.

2 Environment Setup

2.1 System Requirements

- **Operating System:** Windows
- **Software:** Visual Studio 2022 (Community Edition) with .NET SDK
- **Programming Language:** C# (latest stable version)

3 Methodology and Execution

3.1 Project Selection and Debugging Setup

- Two console games were chosen from the `dotnet-console-games` repository.

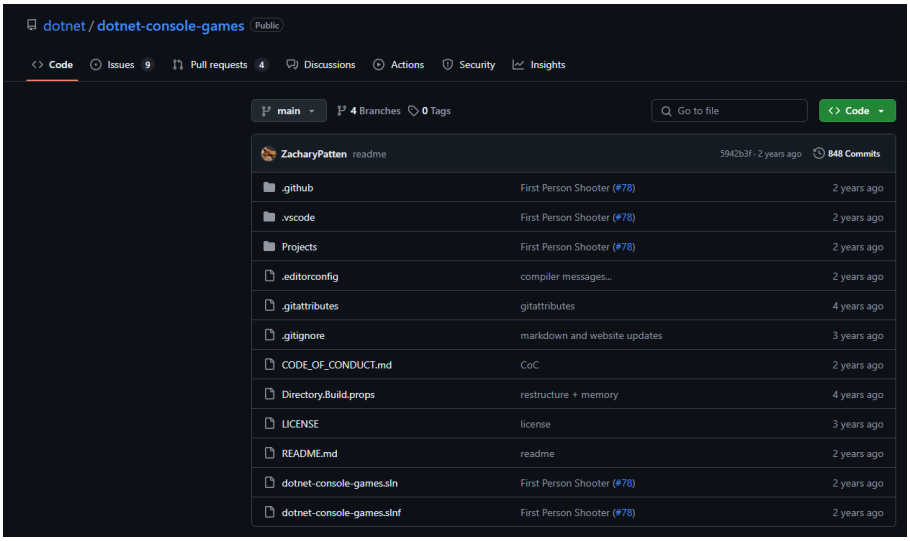


Figure 1: Dotnet Console Games GitHub Repository

- Projects were opened in Visual Studio 2022.
- Breakpoints were inserted at key locations to capture control flow.
- Debugger operations including `Step Over` (F10), `Step In` (F11), and `Step Out` (Shift+F11) were used to trace execution and identify bugs.

3.2 Bug Hunting and Fixing

Bug #1: Fixing Crash on Window Resize

Code Before Bug

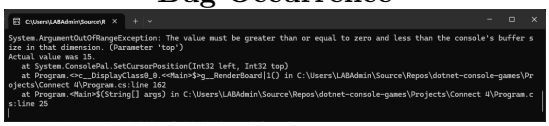
```
while (true)
{
    (int I, int J) move = default;
    if (player1Turn)
    {
        RenderBoard();
        int i = 0;
        Console.SetCursorPosition(moveMinI,
        ↪  moveJ);
        Console.Write('v');
```

Code After Fix

```
while (true)
{
if (Console.WindowWidth < minWidth ||
↪  Console.WindowHeight < minHeight)
{
    Console.Clear();
    Console.WriteLine("Window size too
    ↪  small!");
    Console.WriteLine("Resize and press [enter]
    ↪  to continue, or [escape] to exit.");

    while (true)
    {
        switch (Console.ReadKey(true).Key)
        {
            case ConsoleKey.Enter:
                if (Console.WindowWidth >=
                ↪  minWidth &&
                ↪  Console.WindowHeight >=
                ↪  minHeight)
                {
                    goto ContinueGame;
                }
                break;
            case ConsoleKey.Escape:
                return;
        }
    }
}
ContinueGame:
```

Bug Occurrence



Bug Fixed Output

Figure 2: Bug #1: Added Continuous Checking Before every Move

Bug #2: No Collision Checking Before Spawing the Bullet

Code Before Bug

```
if (tank.IsShooting)
{
    tank.Bullet = new Bullet()
    {
        X = tank.Direction switch
        {
            Direction.Left => tank.X - 3,
            Direction.Right => tank.X + 3,
            _ => tank.X,
        },
        Y = tank.Direction switch
        {
            Direction.Up => tank.Y - 2,
            Direction.Down => tank.Y + 2,
            _ => tank.Y,
        },
        Direction = tank.Direction,
    };
    tank.IsShooting = false;
    continue;
}
```

Bug Occurrence

Code After Fix

```
if (tank.IsShooting)
{
    int spawnX = tank.Direction switch
    {
        Direction.Left => tank.X - 3,
        Direction.Right => tank.X + 3,
        _ => tank.X,
    };
    int spawnY = tank.Direction switch
    {
        Direction.Up => tank.Y - 2,
        Direction.Down => tank.Y + 2,
        _ => tank.Y,
    };

    bool blocked =
        spawnX <= 0 || spawnX >= 74 ||
        spawnY <= 0 || spawnY >= 27
        || (5 < spawnX && spawnX < 11 && spawnY
        ↪  == 13)
        || (spawnX == 37 && 3 < spawnY &&
        ↪  spawnY < 7)
        || (spawnX == 37 && 20 < spawnY &&
        ↪  spawnY < 24)
        || (63 < spawnX && spawnX < 69 &&
        ↪  spawnY == 13);

    if (!blocked)
    {
        tank.Bullet = new Bullet()
        {
            X = spawnX,
            Y = spawnY,
            Direction = tank.Direction
        };
    }
```

Bug Fixed Output

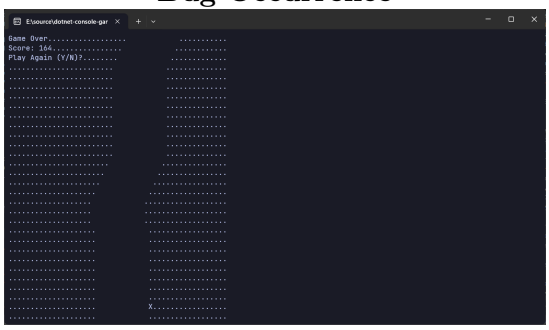
Figure 3: Bug #2: Checking Spawning Location of the Bullet Before Shooting

Bug #3: Previous Game's Score being carried Over

Code Before Bug

```
const int roadWidth = 10;
gameRunning = true;
carPosition = width / 2;
carVelocity = 0;
int leftEdge = (width - roadWidth) / 2;
int rightEdge = leftEdge + roadWidth + 1;
scene = new char[height, width];
```

Bug Occurrence



Code After Fix

```
const int roadWidth = 10;
gameRunning = true;
carPosition = width / 2;
carVelocity = 0;
score = 0;
int leftEdge = (width - roadWidth) / 2;
int rightEdge = leftEdge + roadWidth + 1;
scene = new char[height, width];
```

Bug Fixed Output

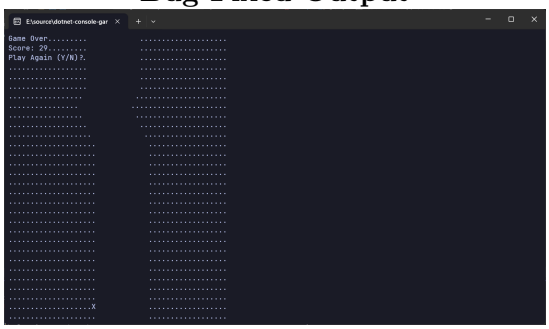


Figure 4: Bug #3: Initialization of Score Before every Game

Bug #4: Game Continues even When same Key is pressed in a row

Code Before Bug

```
if (index < keyCount && key != previous)
{
    previous = key;
    clicks += index > 1 ? BigInteger.Pow(10,
    ↪  index - 1) / (index - 1) : 1;
}
break;
```

Code After Fix

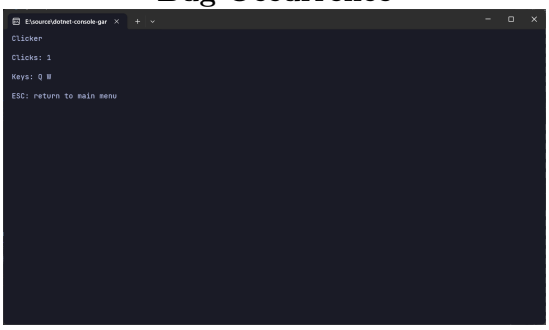
```
if (index < keyCount)
{
    if (key == previous)
    {
        Console.Clear();
        Console.WriteLine(
        $"""
         Clicker

         You Lost!

         You pressed [{(char)key}] twice in a
         ↪  row.
         Final score: {clicksString}
         Time: {DateTime.Now - start}

         ESC: return to main menu
        """);
        GameOverInput:
        Console.CursorVisible = false;
        switch (Console.ReadKey(true).Key)
        {
            case ConsoleKey.Escape: goto
            ↪  MainMenu;
            default: goto GameOverInput;
        }
    }
    previous = key;
    clicks += index > 1 ? BigInteger.Pow(10,
    ↪  index - 1) / (index - 1) : 1;
}
break;
```

Bug Occurrence



Bug Fixed Output

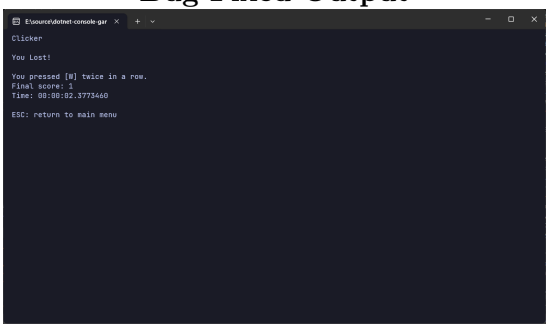


Figure 5: Bug #4: Adding Losing Condition when same Key is pressed in a row

Bug #5: Tank Not Exploding on Continuous Shooting

Code Before Bug

```
if (collision)
{
    if (collisionTank is not null &&
    ↪  --collisionTank.Health <= 0)
    {
        collisionTank.ExplodingFrame = 1;
    }
    tank.Bullet = null;
}
```

Code After Fix

```
if (collision)
{
    if (collisionTank is not null &&
    ↪  collisionTank.ExplodingFrame is 0 &&
    ↪  --collisionTank.Health <= 0)
    {
        collisionTank.ExplodingFrame = 1;
    }
    tank.Bullet = null;
}
```

Bug Occurrence

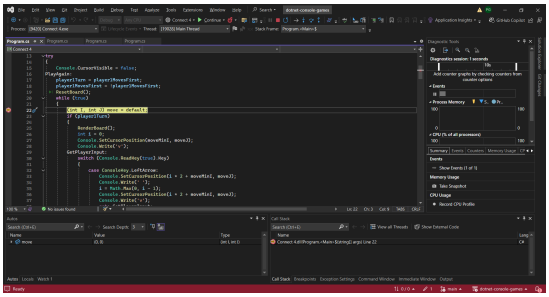
Bug Fixed Output

Figure 6: Bug #5: Added Exploding Frame Checking

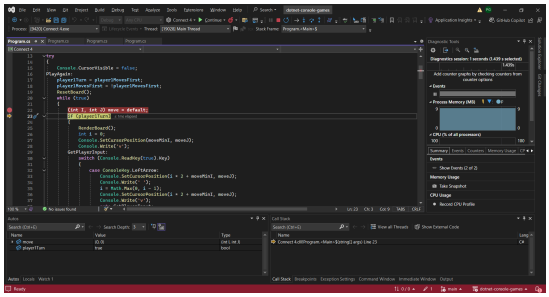
4 Results and Analysis

4.1 Debugging Visual Illustrations

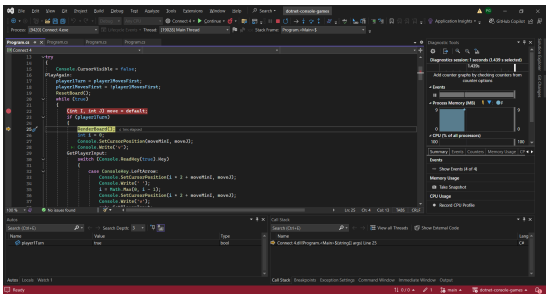
4.1.1 Bug 1 Debugging Illustrations



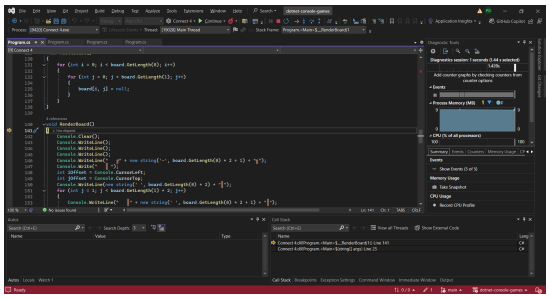
Step Over (Before)



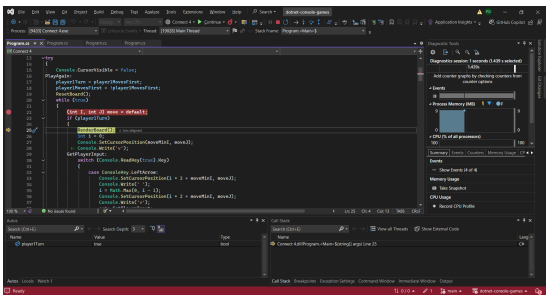
Step Over (After)



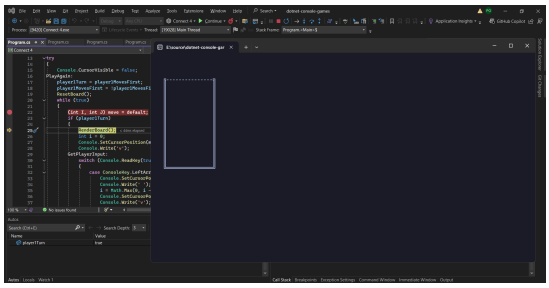
Step In (Before)



Step In (After)



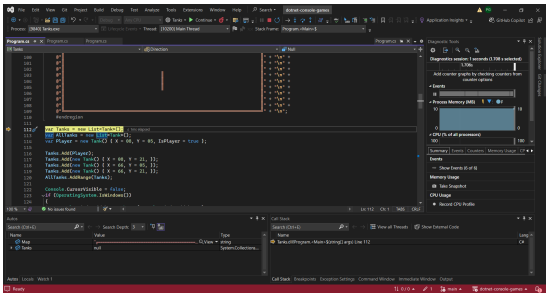
Step Out (Before)



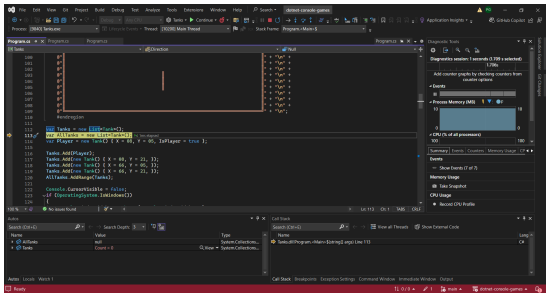
Step Out (After)

Figure 7: Debugging Visual Illustrations for Game 1

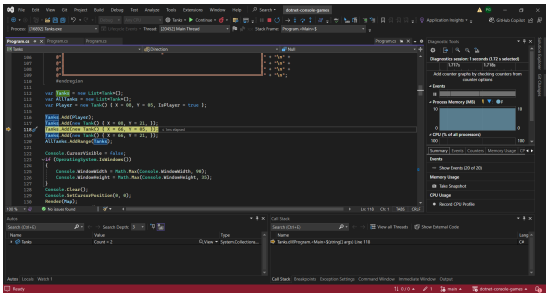
4.1.2 Bug 2 Debugging Illustrations



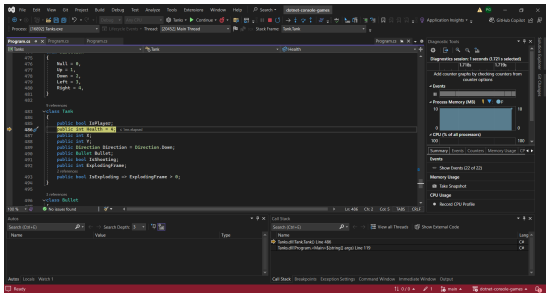
Step Over (Before)



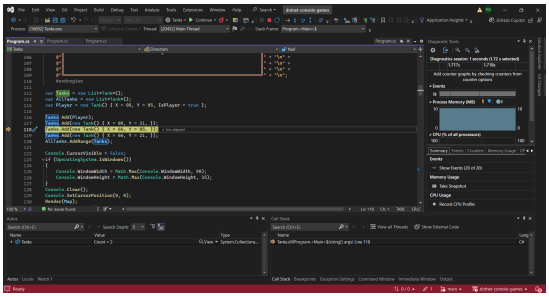
Step Over (After)



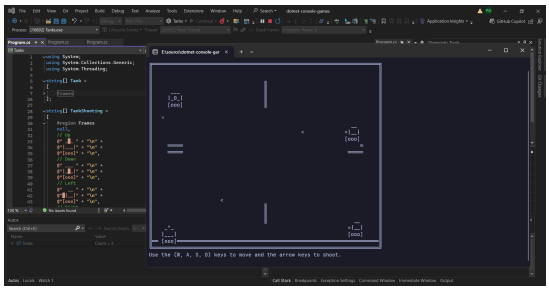
Step In (Before)



Step In (After)



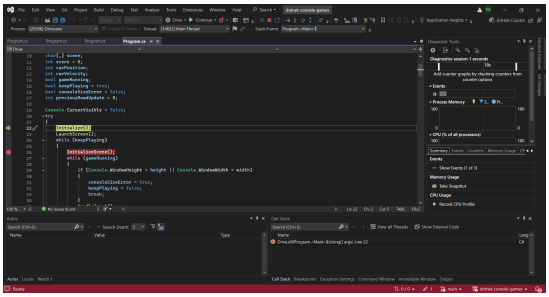
Step Out (Before)



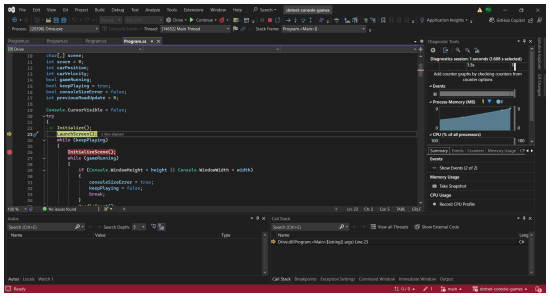
Step Out (After)

Figure 8: Debugging Visual Illustrations for Game 2

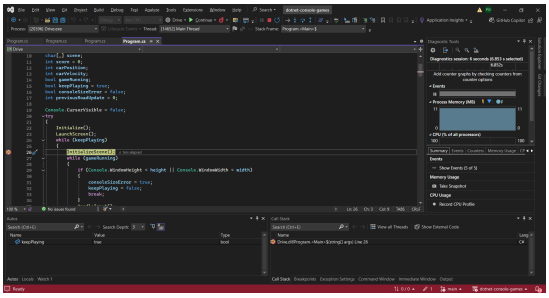
4.1.3 Bug 3 Debugging Illustrations



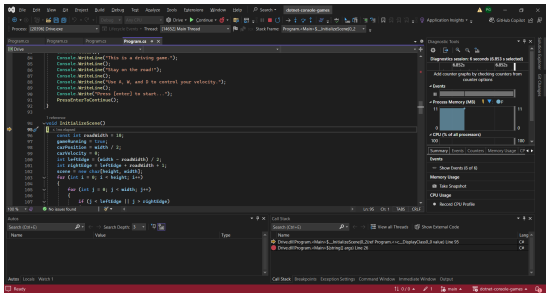
Step Over (Before)



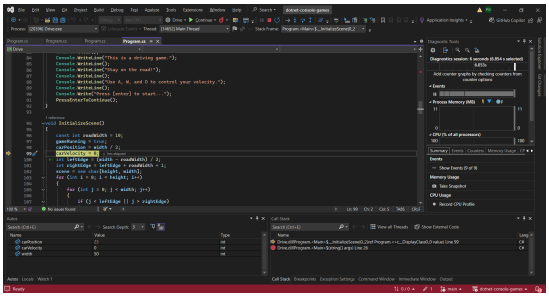
Step Over (After)



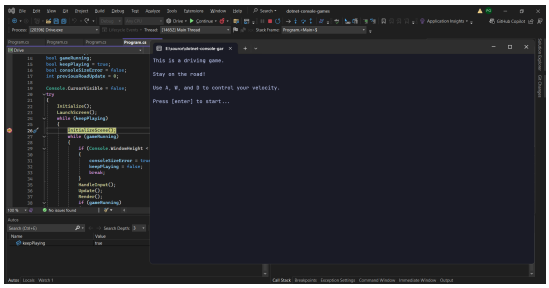
Step In (Before)



Step In (After)



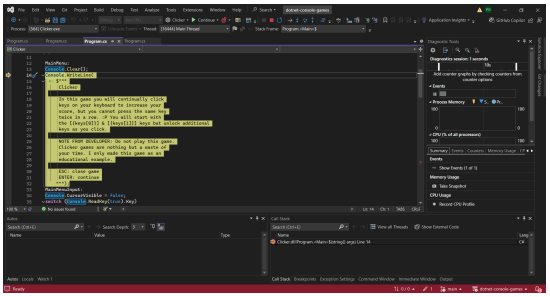
Step Out (Before)



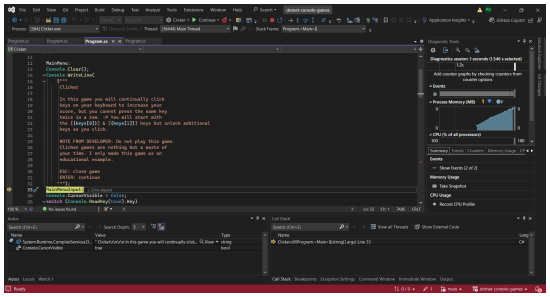
Step Out (After)

Figure 9: Debugging Visual Illustrations for Game 3

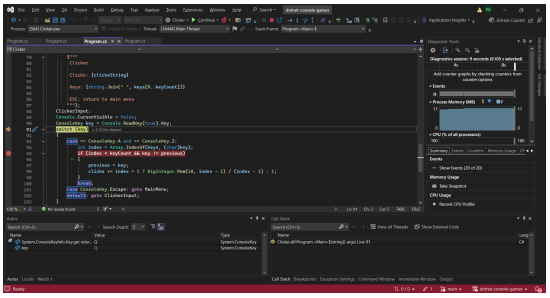
4.1.4 Bug 4 Debugging Illustrations



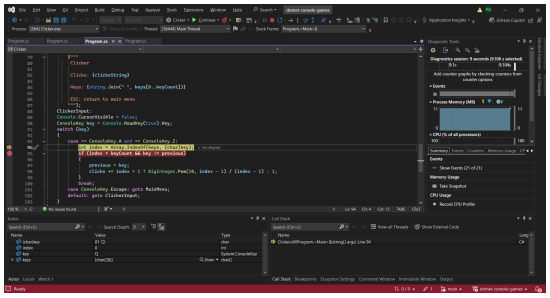
Step Over (Before)



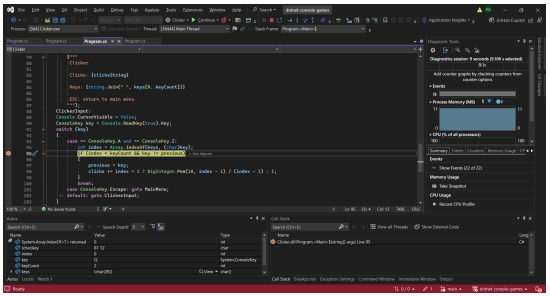
Step Over (After)



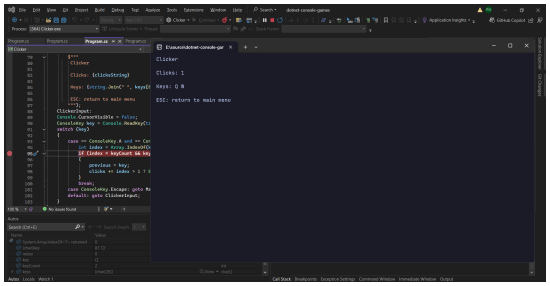
Step In (Before)



Step In (After)



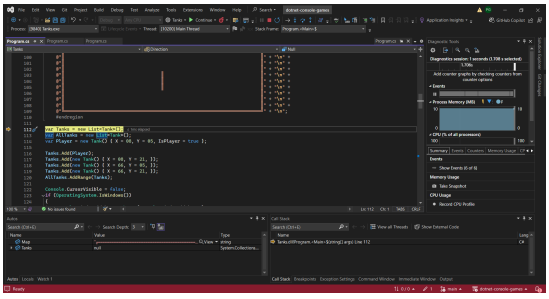
Step Out (Before)



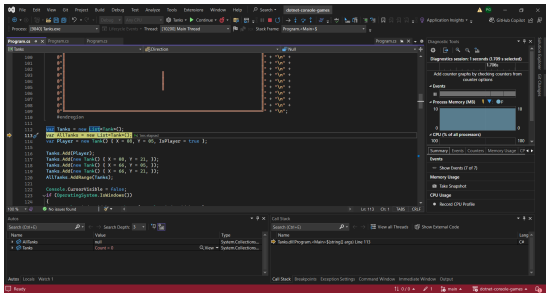
Step Out (After)

Figure 10: Debugging Visual Illustrations for Game 4

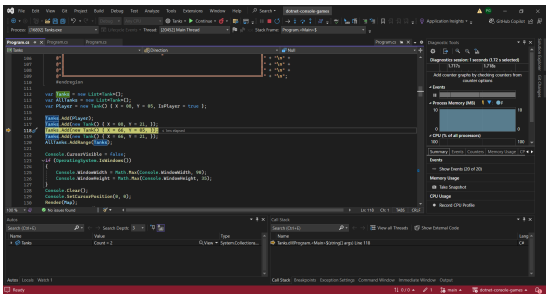
4.1.5 Bug 5 Debugging Illustrations



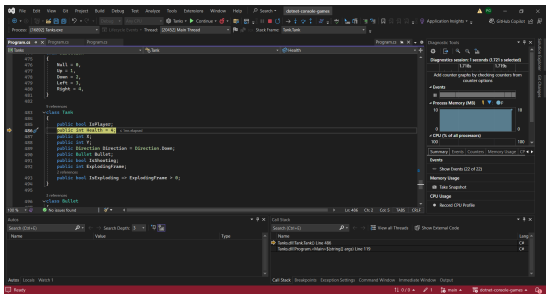
Step Over (Before)



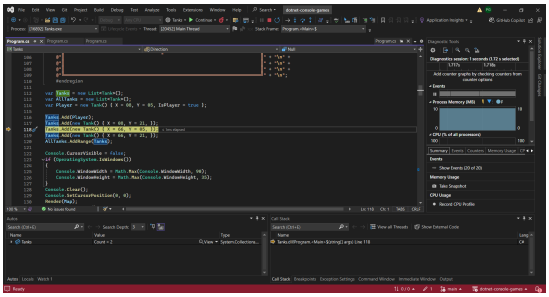
Step Over (After)



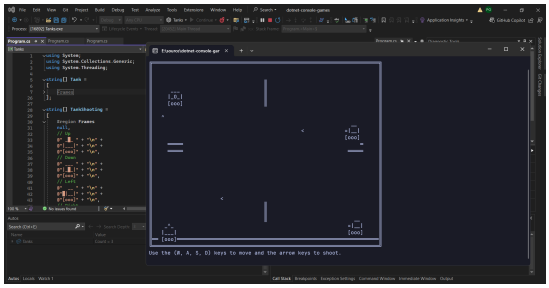
Step In (Before)



Step In (After)



Step Out (Before)



Step Out (After)

Figure 11: Debugging Visual Illustrations for Game 5

4.2 Bug Context Analysis

Example Bug 1:

- **What:** When the window is resized and the player presses Enter.
- **When:** Occurred during the game loop.
- **Why:** No checking for window size change before updating the board.
- **Where:** At the start of gameplay loop.
- **How:** Adding window size checking before every player input.

Example Bug 2:

- **What:** Firing Bullets through the Wall.
- **When:** The tank is very close to the wall and fires a bullet in that direction.
- **Why:** An object was not properly instantiated.

- **Where:** Spawning the bullet without checking for initial collision.
- **How:** Adding collision checking before spawning the bullet.

Example Bug 3:

- **What:** Previous game scored being carried over to the next game.
- **When:** Restarting to next game.
- **Why:** Score isn't initialized before every game.
- **Where:** In the `InitializeScene()` method.
- **How:** Adding score = 0 initialization when new game starts.

Example Bug 4:

- **What:** Game continues even when the same key is pressed twice a row.
- **When:** Pressing the same key repeatedly.
- **Why:** No penalty or losing condition added for this case.
- **Where:** In the gameloop, user input is checked.
- **How:** Using a previous variable to keep track of the same key not being pressed twice.

Example Bug 5:

- **What:** Hitting the tank continuously keeps that tank in lasting exploding state.
- **When:** Repeatedly shooting the tank even after exploding.
- **Why:** Not checking for exploding frame.
- **Where:** In the condition where the exploding tanks' health is checked.
- **How:** Only set the exploding frame condition when it is 0.

5 Discussion and Conclusion

5.1 Challenges and Reflections

- Locating bugs in fast-moving game loops required precise breakpoint placement.
- Debugging interactive console game code was challenging due to its dynamic nature.
- Verifying bug fixes involved careful observation of debugger output before and after changes.

5.2 Lessons Learned

- The Visual Studio Debugger is essential for tracking down subtle bugs.
- Even minor code changes, such as altering operators or checking for null values, can significantly improve stability.
- Detailed bug documentation (what, when, why, where, and how) is valuable for learning and maintaining code.

5.3 Summary

This lab provided hands-on experience with debugging C# console game applications. By using the Visual Studio Debugger (step over, step in, step out) to trace control flow, bugs were identified and fixed. These activities underscored the importance of rigorous debugging techniques in ensuring stable game performance.

Lab Assignment Report 12

CS202: Software Tools and Techniques for CSE

Roll Number: 22110087
Name: Parth Govale

Contents

1 Introduction, Setup, and Tools

1.1 Overview

This lab introduces event-driven programming using C# for Windows Forms applications. The main objectives are to understand the event-driven paradigm, implement a time-based alarm system, and demonstrate how user interactions and application state changes control program flow.

2 Environment Setup

2.1 System Requirements

- **Operating System:** Windows
- **Software:** Visual Studio 2022 (Community Edition) with .NET SDK
- **Programming Language:** C# (latest stable version)

3 Methodology and Execution

3.1 Activity 1: Console Application for Time-based Alarm

- **Design:** Develop a console application that:
  - Prompts the user for a target time in `HH:MM:SS` format.
  - Continuously checks the current system time.
  - When the current time matches the target, triggers a user-defined event `raiseAlarm` following the publisher/subscriber model.
  - The `raiseAlarm` event calls the function `Ring_alarm()` to display a message.
- **Implementation:**

```
// Console Alarm Application
using System;
using System.Timers;

namespace ConsoleAlarmApp
{
    class Program
    {
        // Declare a delegate for the alarm event
        public delegate void AlarmEventHandler();
        public static event AlarmEventHandler raiseAlarm;

        static void Main(string[] args)
        {
            Console.WriteLine("Enter target time in HH:MM:SS format:");
            string targetTime = Console.ReadLine();

            // Subscribe the Ring_alarm method to the raiseAlarm event.
            raiseAlarm += Ring_alarm;

            // Check time every second
            while (true)
```

```
            {
                string currentTime = DateTime.Now.ToString("HH:mm:ss");
                if(currentTime.Equals(targetTime))
                {
                    raiseAlarm(); // Raise the alarm event
                    break;
                }
            }
        }

        static void Ring_alarm()
        {
            Console.WriteLine("Alarm! Target time reached.");
        }
    }
}
```

- **Expected Outcome:**

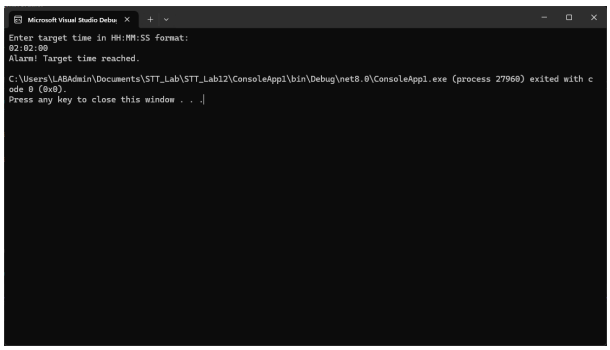


Figure 1: Alarm Ringing on Input Matching

3.2 Activity 2: Windows Forms Application for Event-driven Alarm

- **Design:** Modify the console application so that all user input/output is performed through a Windows Form:
  - The form should contain a textbox for time input (validated for `HH:MM:SS` format) and a start button.
  - On clicking the start button, the application should change the background color of the form every second.
  - When the current system time matches the target time, the background color stops changing and a message box is displayed.

- **Implementation:**

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WindowsFormsAlarmApp
{
    public partial class Form1 : Form
    {
```

```
        private System.Windows.Forms.Timer timer;
        private string targetTime;

        public Form1()
        {
            InitializeComponent();
            // Initialize Timer for 1-second intervals.
            timer = new System.Windows.Forms.Timer();
            timer.Interval = 1000; // 1 second interval.
            timer.Tick += Timer_Tick;
        }

        private void startButton_Click(object sender, EventArgs e)
        {
            // Get target time from textbox in HH:MM:SS format.
            targetTime = timeTextBox.Text;
            timer.Start(); // Start the timer.
        }

        private void Timer_Tick(object sender, EventArgs e)
        {
            string currentTime = DateTime.Now.ToString("HH:mm:ss");
            // Change the background color every second.
            this.BackColor = GetRandomColor();
            if (currentTime.Equals(targetTime))
            {
                timer.Stop();
                // Raise the alarm by displaying a message box.
                MessageBox.Show("Alarm! Target time reached.", "Alarm");
            }
        }

        private Color GetRandomColor()
        {
            Random rnd = new Random();
            return Color.FromArgb(rnd.Next(256), rnd.Next(256), rnd.Next(256));
        }
    }
}
```

- **Expected Outcome:**

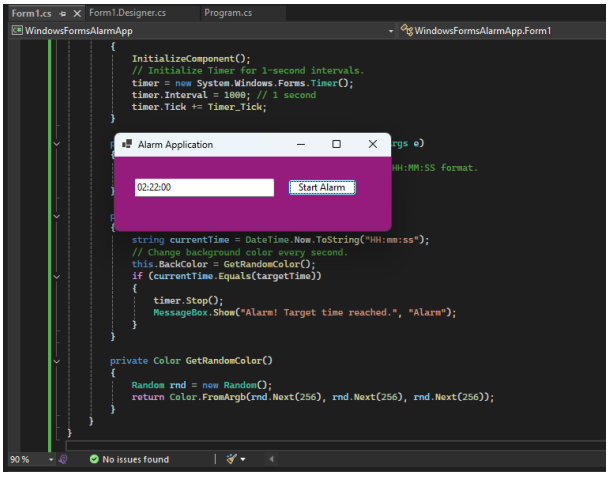


Figure 2: Background Color Changing Form

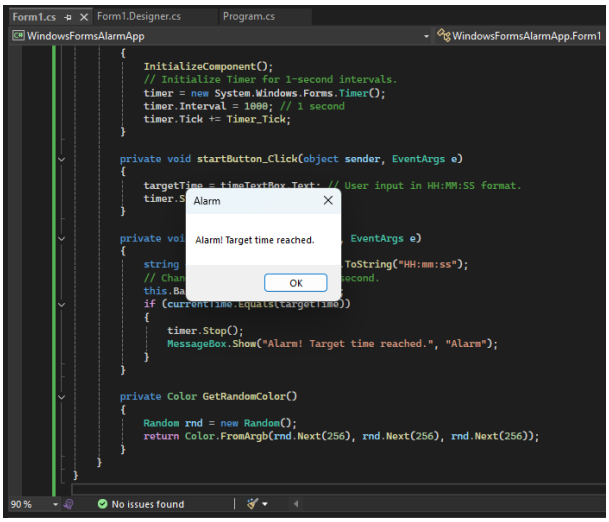


Figure 3: Alarm Ringing on Input Matching

4 Results and Analysis

4.1 Activity 1: Console Application Results

- The console application successfully accepts a target time and continuously compares it with the system time.
- When the target time is reached, the event `raiseAlarm` is triggered and the message from `Ring_alarm()` is displayed.

4.2 Activity 2: Windows Forms Application Results

- The Windows Forms application correctly validates user input from the textbox.
- The background color of the form changes every second after starting the timer.
- Upon reaching the target time, the background color stops changing and a message box is displayed.

5 Discussion and Conclusion

5.1 Challenges and Reflections

- Learning the event-driven model in Windows Forms required understanding how events trigger code execution.
- Integrating timer-based events and handling user input validation on the GUI posed challenges.

5.2 Lessons Learned

- Event-driven programming allows for responsive user interfaces and intuitive control flow.
- Windows Forms provides a rich set of tools for rapid application development and debugging.
- Understanding the publisher/subscriber model is key to implementing events in C#.

5.3 Summary

This lab provided hands-on experience with event-driven programming using C# Windows Forms. The tasks covered building a console-based time alarm and converting it to a GUI-based application. Through these activities, the benefits of a responsive, event-driven design and the debugging techniques in Visual Studio were clearly demonstrated.

6 Appendix

6.1 Tools and Resources

- C# Documentation
- Visual Studio Official Website
- .NET SDK Downloads
- Event-driven Programming - Wikipedia
- What, Why, How of Event-driven Programming