CS202: Software Tools and Techniques

Lab 11 Report: Analyzing C# Console Games for Bugs

Jiya Bhavin Desai | Roll No. 22110107 | 03.04.2025

---

1. Introduction

This report presents the work done in lab 11, an analysis of standard C# console games, utilizing Visual Studio's debugging tools to uncover and resolve issues that lead to program crashes.
The debugging approach included selecting games from the dotnet-console-games repository, exploring their control flow with the Visual Studio Debugger, and detecting major bugs through structured techniques such as setting breakpoints and using step-in, step-over, and step-out operations.
This study highlights how to follow execution paths, observe variable values during runtime, and pinpoint the underlying causes of software failures. By understanding the timing, location, and reasons behind these bugs, we can create stronger error-handling mechanisms and enhance the overall stability of the software.

1.1 Setup

- Since I didn't have enough space on my laptop for installing Visual Studio 2022 (needed for building the C# console applications), I worked with one of the PCs present in the computer lab, which already had Visual Studio 2022 installed, with support for .NET framework.

1.2 Tools

- **Operating System:** Windows 10
- **IDE:** Visual Studio 2022 (Community Edition)

- **Target Framework:** .NET *(version 6 wasn't available in the lab PC, so I used an earlier version)*
- **Programming Language:** C# (latest stable version as available in the lab PC)

2. Methodology and Execution

2.1 Bug Identification Process

The methodology for identifying bugs in the .NET Console Games collection followed a systematic approach. The following were done after first familiarising myself with the game and its console interface.

1. Code Review: Each game's source code was carefully reviewed to identify potential issues that could lead to crashes. This involved examining:
    - Array access without bounds checking
    - Division operations without zero checks
    - Null reference usage without validation
    - Collection modifications during iteration
    - Input handling without proper validation

2. Dynamic Analysis: Each game was executed and tested with various inputs and edge cases to trigger potential crashes. This included:
    - Testing boundary conditions (e.g., game borders, maximum/minimum values)
    - Providing unexpected user inputs
    - Creating race conditions where possible
    - Forcing resource constraints

3. Crash Analysis: When crashes occurred, the following information was collected:
    - Stack trace and exception details
    - Program state at the time of crash
    - Sequence of actions that led to the crash
    - Environmental factors that might have contributed

4. Root Cause Analysis: For each identified bug, a thorough analysis was conducted to determine:
    - The underlying cause of the bug
    - Why the bug wasn't caught during development
    - How the bug affects gameplay
    - Potential impact on other parts of the code

2.2 Debugging Tools and Techniques

The following tools and techniques were employed during the debugging process:

1. Visual Studio Debugger: Used to set breakpoints, step through code, and inspect variables at runtime.

2. Breakpoints: Strategic breakpoints were placed at:
    - Suspected problematic code sections
    - Exception-throwing locations
    - Input handling routines
    - Complex algorithmic implementations

3. Debugging Commands:
    - Step Into: Used to follow code execution into method calls
    - Step Over: Used to execute a line of code without diving into methods
    - Step Out: Used to complete the current method and return to the caller

4. Watch Windows: Used to monitor the values of critical variables during execution.

5. Immediate Window: Used to evaluate expressions and test potential fixes during debugging sessions.

6. Exception Settings: Configured to break when exceptions were thrown, even if caught, to identify all potential issues.

2.3 Bug Fixing Process

For each identified bug, the following process was followed:

1. Reproduction: The bug was consistently reproduced to understand its exact triggers and behavior.

2. Isolation: The specific code section responsible for the bug was isolated and analyzed.

3. Fix Design: A fix was designed that addressed the root cause while maintaining the game's functionality.

4. Implementation: The fix was implemented in a separate directory to preserve the original code for reference.

5. Verification: The fixed code was tested to ensure:
- The bug was completely resolved
- No new bugs were introduced
- Game functionality remained intact
- Edge cases were handled properly

6. Documentation: Comprehensive documentation was created for each bug, including:
- Bug context (what, when, why, where)
- Reproduction steps
- Debugging guidance (where to set breakpoints)
- Fix explanation and code

3. Results and Analysis

After a lot of hunting, I found the following 5 bugs across all the games:

3.1 Bug 1: Duck Hunt Game - Index Out of Range Bug

When I started the Duck Hunt game and fired multiple bullets in succession, the game crashed. On inspecting the source code and setting breakpoints and following the abovfe mentioned debugging methodology, I found the following.

Context

- What: The game can crash due to an index out of range exception in the bullet update logic.
- When: This occurs when updating and removing bullets.
- Why: The code removes bullets from the list while iterating through it, but doesn't adjust the index properly.
- Where: The bug is in the bullet update section of Projects/Duck Hunt/Program.cs.

Reproduction Steps

- Start the Duck Hunt game
- Fire multiple bullets in quick succession
- The game may crash due to an index out of range exception

Debugging

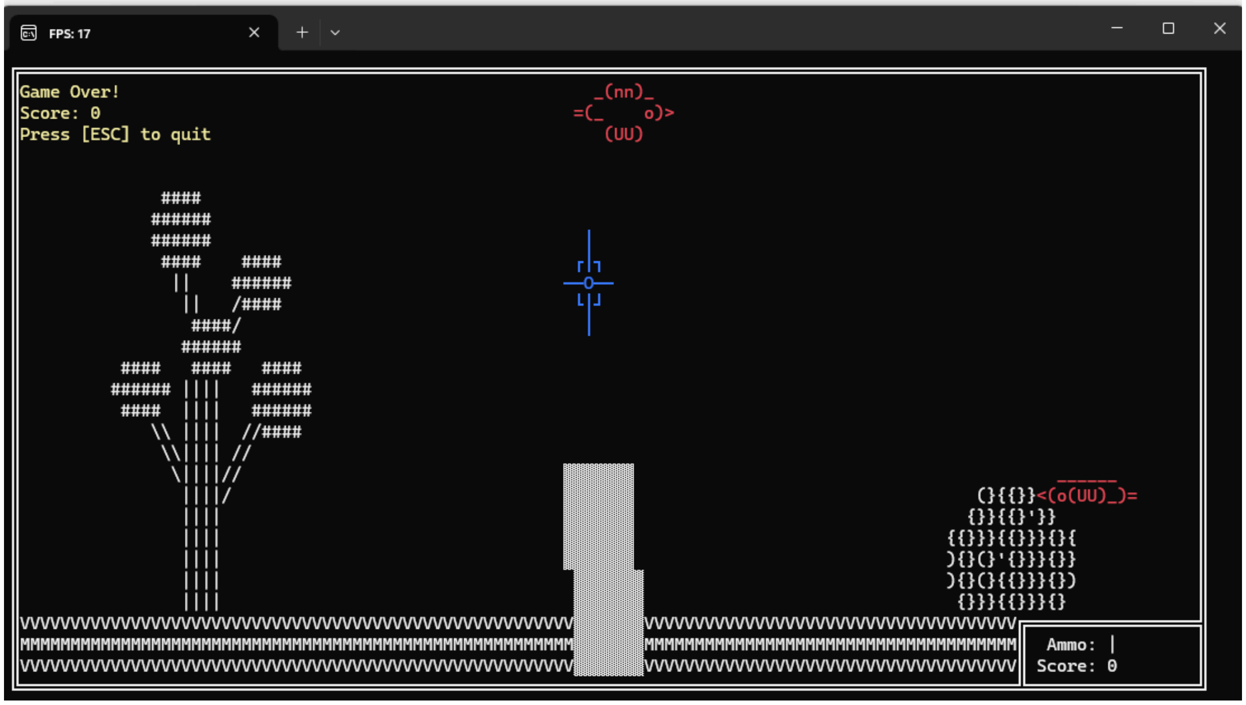
I set breakpoints at multiple places in the main game loop and in the bullet list update part specifically in Projects/Duck Hunt/Program.cs (in the bullet update section) to observe the potential index out of range access.

Fix

Adjusted the index after removing a bullet from the list and added bounds checking.

```
for (int i = 0; i < bullets.Count; i++)
{
    bullets[i].UpdatePosition();

    if (bullets[i].OutOfBounds)
    {
        bullets.RemoveAt(i);
        i--; // Adjust index after removal
        continue;
    }

    // Check for valid indices before accessing
    if (i < bullets.Count)
    {
        foreach (Bird bird in birds)
        {
                if (!bird.IsDead &&
                    (bird.Contains((int)bullets[i].X[0], (int)bullets[i].Y[0]) ||
                    bird.Contains((int)bullets[i].X[1], (int)bullets[i].Y[1])))
                {
                    bird.IsDead = true;
                    ammoCount += 2;
                    score += 350;
                }
            }

             DrawToScreenWithColour((int)bullets[i].X[0],  (int)bullets[i].Y[0],
ConsoleColor.DarkGray, '█');
             DrawToScreenWithColour((int)bullets[i].X[1],  (int)bullets[i].Y[1],
ConsoleColor.DarkGray, '█');
        }
}
```

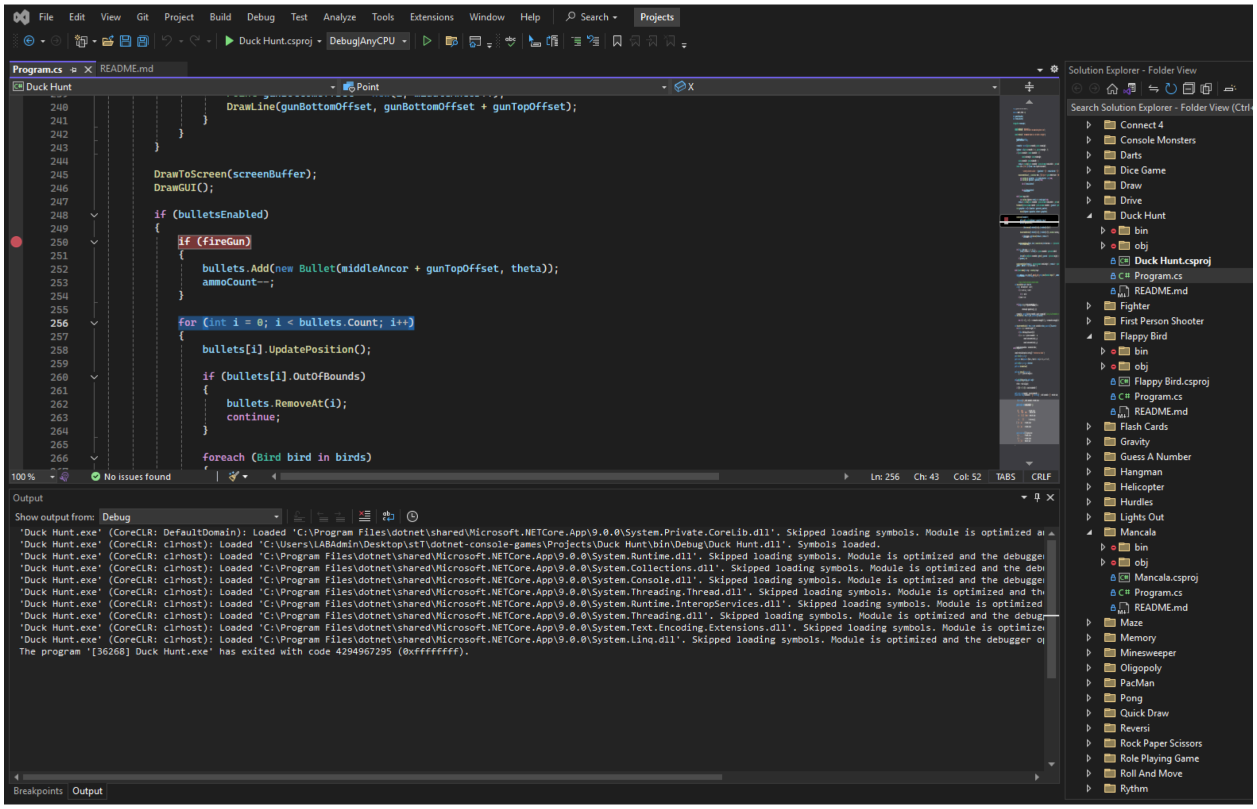
Visual walkthrough of the bug and debugging



```
Game Over!
Score: 0
Press [ESC] to quit
```

```
Ammo: |
Score: 0
```

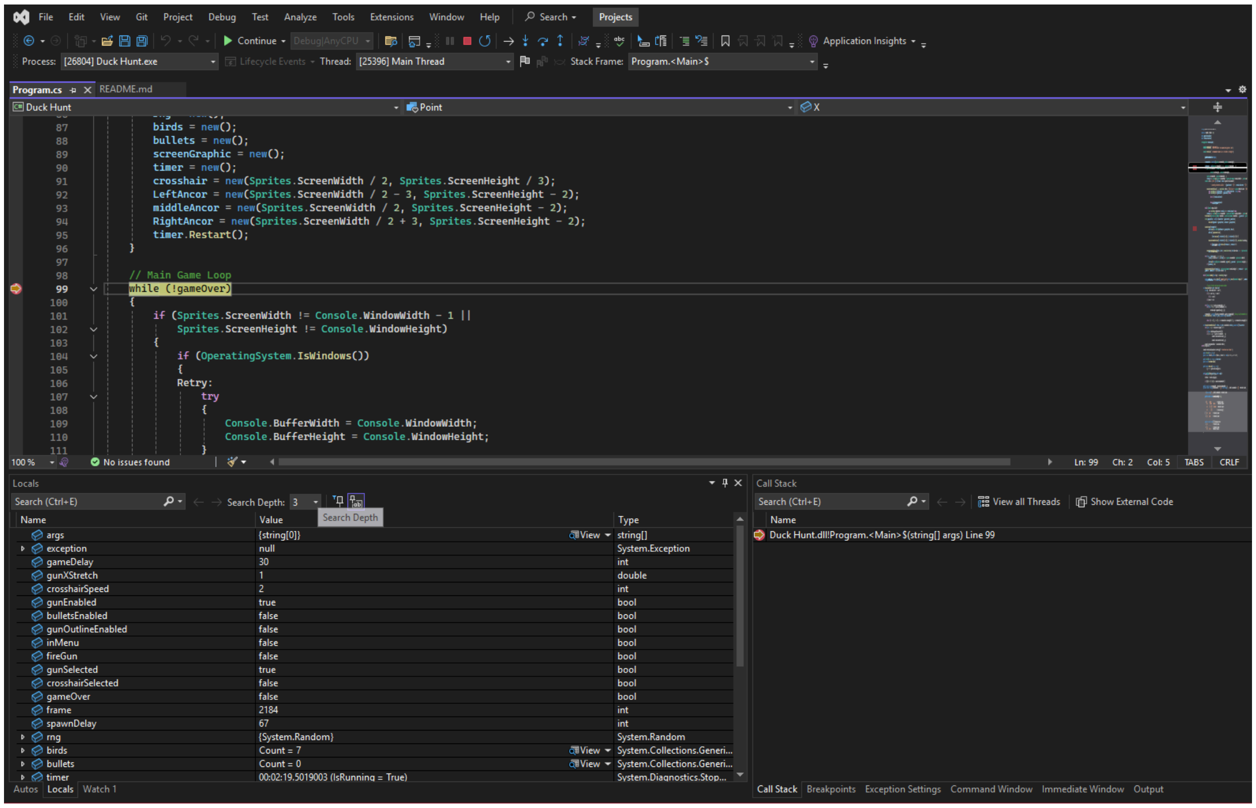
Game crashes (unresponsive - page hangs) when I fired multiple bullets in succession



```
                    DrawLine(gunBottomOffset, gunBottomOffset + gunTopOffset);
                }
            }
        }

        DrawToScreen(screenBuffer);
        DrawGUI();

        if (bulletsEnabled)
        {
            if (fireGun)
            {
                bullets.Add(new Bullet(middleAncor + gunTopOffset, theta));
                ammoCount--;
            }

            for (int i = 0; i < bullets.Count; i++)
            {
                bullets[i].UpdatePosition();

                if (bullets[i].OutOfBounds)
                {
                    bullets.RemoveAt(i);
                    continue;
                }

                foreach (Bird bird in birds)
```

```
'Duck Hunt.exe' (CoreCLR: DefaultDomain): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Private.CoreLib.dll'. Skipped loading symbols. Module is optimized a
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Users\LABAdmin\Desktop\stT\dotnet-console-games\Projects\Duck Hunt\bin\Debug\Duck Hunt.dll'. Symbols loaded.
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Runtime.dll'. Skipped loading symbols. Module is optimized and the debugge
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Collections.dll'. Skipped loading symbols. Module is optimized and the deb
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Console.dll'. Skipped loading symbols. Module is optimized and the debugge
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Threading.Thread.dll'. Skipped loading symbols. Module is optimized and th
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Runtime.InteropServices.dll'. Skipped loading symbols. Module is optimized
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Threading.dll'. Skipped loading symbols. Module is optimized and the debug
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Text.Encoding.Extensions.dll'. Skipped loading symbols. Module is optimize
'Duck Hunt.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Linq.dll'. Skipped loading symbols. Module is optimized and the debugger o
The program '[36268] Duck Hunt.exe' has exited with code 4294967295 (0xffffffff).
```

Setting breakpoints



```
            birds = new();
            bullets = new();
            screenGraphic = new();
            timer = new();
            crosshair = new(Sprites.ScreenWidth / 2, Sprites.ScreenHeight / 3);
            LeftAncor = new(Sprites.ScreenWidth / 2 - 3, Sprites.ScreenHeight - 2);
            middleAncor = new(Sprites.ScreenWidth / 2, Sprites.ScreenHeight - 2);
            RightAncor = new(Sprites.ScreenWidth / 2 + 3, Sprites.ScreenHeight - 2);
            timer.Restart();
        }

        // Main Game Loop
        while (!gameOver)
        {
            if (Sprites.ScreenWidth != Console.WindowWidth - 1 ||
                Sprites.ScreenHeight != Console.WindowHeight)
            {
                if (OperatingSystem.IsWindows())
                {
                Retry:
                    try
                    {
                        Console.BufferWidth = Console.WindowWidth;
                        Console.BufferHeight = Console.WindowHeight;
                    }
```

Name	Value	Type
args	{string[0]}	string[]
exception	null	System.Exception
gameDelay	30	int
gunXStretch	1	double
crosshairSpeed	2	int
gunEnabled	true	bool
bulletsEnabled	false	bool
gunOutlineEnabled	false	bool
inMenu	false	bool
fireGun	false	bool
gunSelected	true	bool
crosshairSelected	false	bool
gameOver	false	bool
frame	2184	int
spawnDelay	67	int
rng	{System.Random}	System.Random
birds	Count = 7	System.Collections.Generi...
bullets	Count = 0	System.Collections.Generi...
timer	00:02:19.5019003 (IsRunning = True)	System.Diagnostics.Stop...

Call Stack

Duck Hunt.dll!Program.<Main>$(string[] args) Line 99

Variables during the breakpoint during a crash

3.2 Bug 2: Snake Game - Direction Reversal Bug

In the Snake game, if the snake's head touches its tail during movement, it is considered an invalid move and the game ends. However, in this implementation of the Snake game, they have considered the direction reversal of a 1-D snake as a valid move which causes the snake to 'touch' its tail and end the game - this is a bug, since 180 degrees direction reversal of the snake shouldn't be allowed.

Context

- What: The game crashes when the player attempts to reverse direction.
- When: This occurs whenever the player presses the arrow key for the opposite direction of current movement.
- Why: The game doesn't validate direction changes, allowing the snake to collide with itself instantly.

- Where: The bug is in the GetDirection() method in Projects/Snake/Program.cs.

Reproduction Steps

- Start the Snake game
- Begin moving in any direction
- Press the arrow key for the opposite direction (e.g., if moving right, press left)
- The snake will immediately collide with itself and crash

Debugging

I set a breakpoint at line 107 in Projects/Snake/Program.cs (the GetDirection() method) to observe how direction changes are handled without validation.

Fix

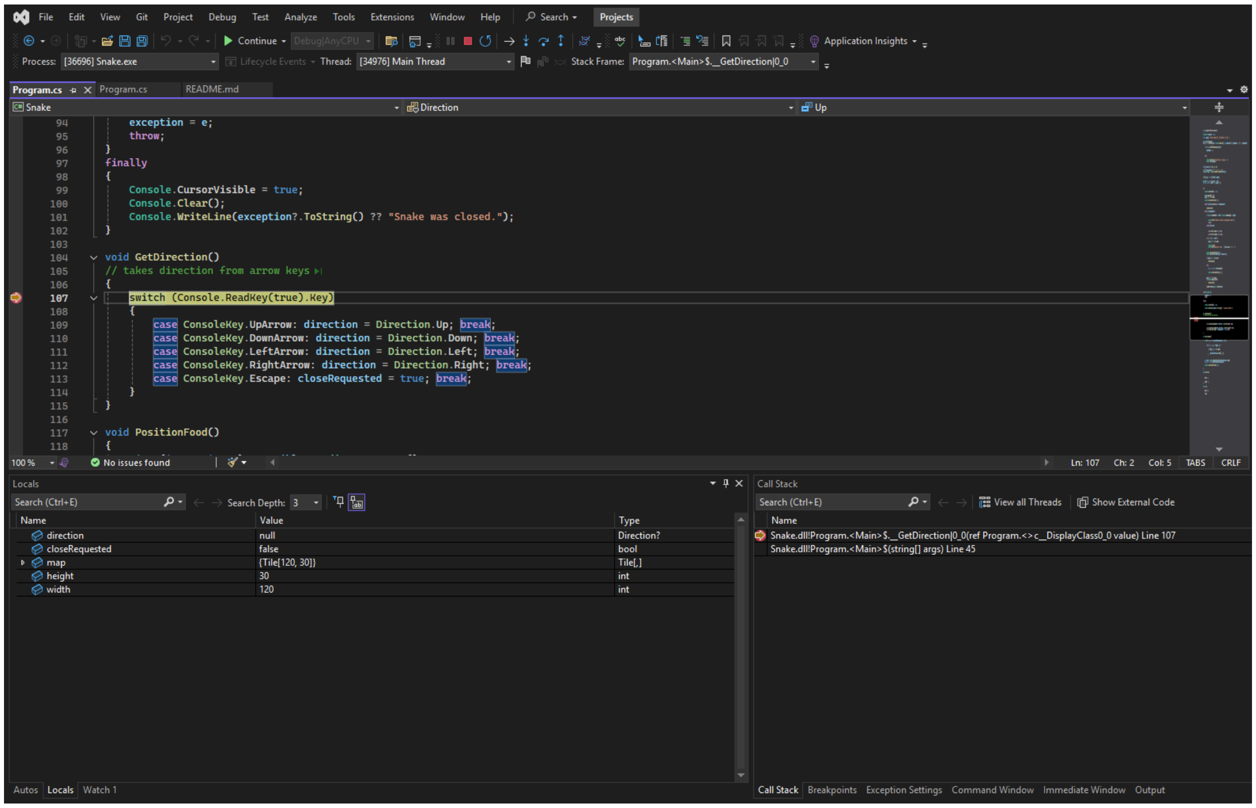
Added validation in the GetDirection() method to prevent direction reversal

---

```
void GetDirection()
{
    Direction? newDirection = null;
    switch (Console.ReadKey(true).Key)
    {
        case ConsoleKey.UpArrow: newDirection = Direction.Up; break;
        case ConsoleKey.DownArrow: newDirection = Direction.Down; break;
        case ConsoleKey.LeftArrow: newDirection = Direction.Left; break;
        case ConsoleKey.RightArrow: newDirection = Direction.Right; break;
        case ConsoleKey.Escape: closeRequested = true; break;
    }

    // Prevent direction reversal (e.g., going from right to left)
    if (newDirection.HasValue)
    {
            bool isReversal = (direction == Direction.Up && newDirection ==
Direction.Down) ||
                                (direction == Direction.Down && newDirection ==
Direction.Up) ||
                                (direction == Direction.Left && newDirection ==
Direction.Right) ||
                               (direction == Direction.Right && newDirection ==
Direction.Left);

        if (!isReversal)
        {
            direction = newDirection;
```

```
                }
        }
}
```

Visual walkthrough of the bug and debugging



Process: [36696] Snake.exe   Thread: [34976] Main Thread   Stack Frame: Program.<Main>$.__GetDirection|0_0

```
 94             exception = e;
 95             throw;
 96         }
 97         finally
 98         {
 99             Console.CursorVisible = true;
100             Console.Clear();
101             Console.WriteLine(exception?.ToString() ?? "Snake was closed.");
102         }
103 
104     void GetDirection()
105     // takes direction from arrow keys
106     {
107         switch (Console.ReadKey(true).Key)
108         {
109             case ConsoleKey.UpArrow: direction = Direction.Up; break;
110             case ConsoleKey.DownArrow: direction = Direction.Down; break;
111             case ConsoleKey.LeftArrow: direction = Direction.Left; break;
112             case ConsoleKey.RightArrow: direction = Direction.Right; break;
113             case ConsoleKey.Escape: closeRequested = true; break;
114         }
115     }
116 
117     void PositionFood()
118     {
```

Ln: 107 Ch: 2 Col: 5 TABS CRLF

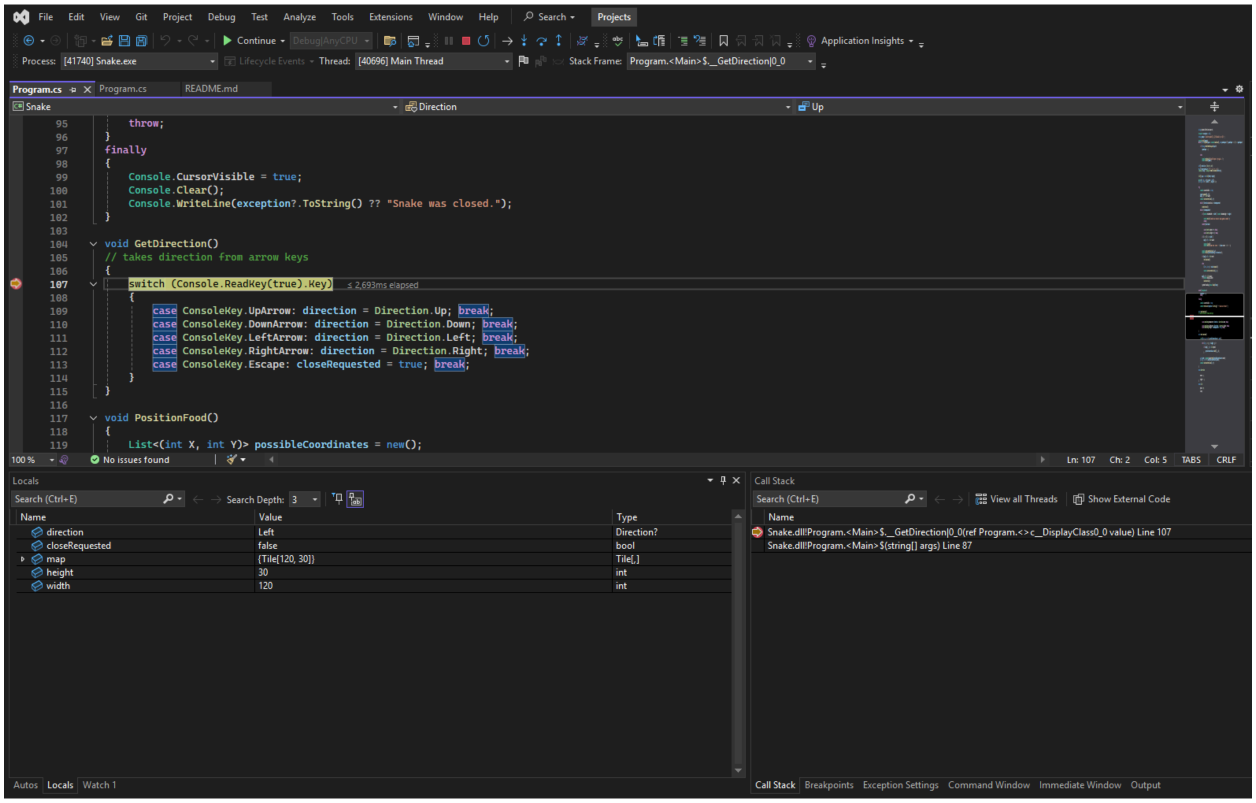
Locals

Name	Value	Type
direction	null	Direction?
closeRequested	false	bool
map	{Tile[120, 30]}	Tile[,]
height	30	int
width	120	int

Call Stack

```
Snake.dll!Program.<Main>$.__GetDirection|0_0(ref Program.<>c__DisplayClass0_0 value) Line 107
Snake.dll!Program.<Main>$(string[] args) Line 45
```

Setting breakpoints


Process: [41740] Snake.exe Thread: [40696] Main Thread Stack Frame: Program.<Main>$.__GetDirection|0_0

```
            throw;
        }
        finally
        {
            Console.CursorVisible = true;
            Console.Clear();
            Console.WriteLine(exception?.ToString() ?? "Snake was closed.");
        }

    void GetDirection()
    // takes direction from arrow keys
    {
        switch (Console.ReadKey(true).Key)  ≤ 2,693ms elapsed
        {
            case ConsoleKey.UpArrow: direction = Direction.Up; break;
            case ConsoleKey.DownArrow: direction = Direction.Down; break;
            case ConsoleKey.LeftArrow: direction = Direction.Left; break;
            case ConsoleKey.RightArrow: direction = Direction.Right; break;
            case ConsoleKey.Escape: closeRequested = true; break;
        }
    }

    void PositionFood()
    {
        List<(int X, int Y)> possibleCoordinates = new();
```

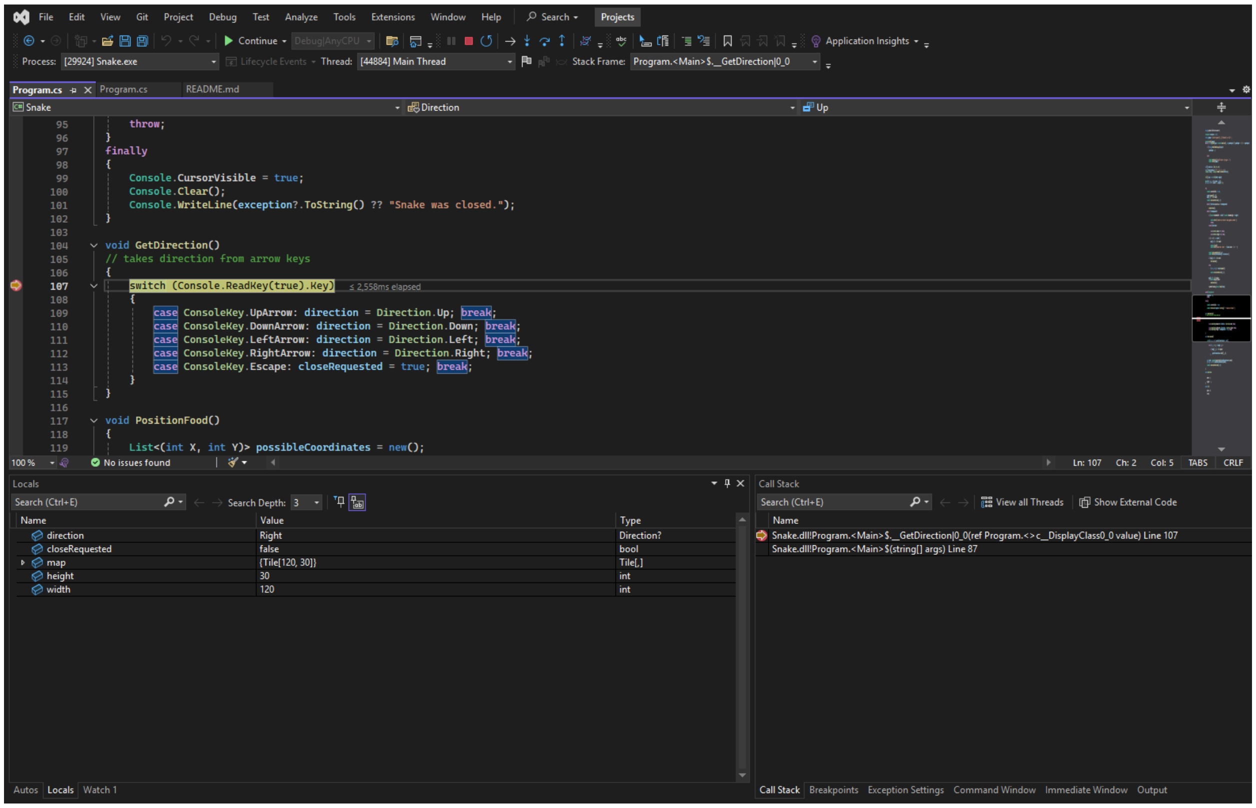
Locals

Name	Value	Type
direction	Left	Direction?
closeRequested	false	bool
map	{Tile[120, 30]}	Tile[,]
height	30	int
width	120	int

Call Stack

Name
Snake.dll!Program.<Main>$.__GetDirection|0_0(ref Program.<>c__DisplayClass0_0 value) Line 107
Snake.dll!Program.<Main>$(string[] args) Line 87

Before switching directions after using Step -in (F11) at every line


Process: [29924] Snake.exe Thread: [44884] Main Thread Stack Frame: Program.<Main>$.__GetDirection|0_0

```
            throw;
        }
        finally
        {
            Console.CursorVisible = true;
            Console.Clear();
            Console.WriteLine(exception?.ToString() ?? "Snake was closed.");
        }

    void GetDirection()
    // takes direction from arrow keys
    {
        switch (Console.ReadKey(true).Key)  ≤ 2,558ms elapsed
        {
            case ConsoleKey.UpArrow: direction = Direction.Up; break;
            case ConsoleKey.DownArrow: direction = Direction.Down; break;
            case ConsoleKey.LeftArrow: direction = Direction.Left; break;
            case ConsoleKey.RightArrow: direction = Direction.Right; break;
            case ConsoleKey.Escape: closeRequested = true; break;
        }
    }

    void PositionFood()
    {
        List<(int X, int Y)> possibleCoordinates = new();
```

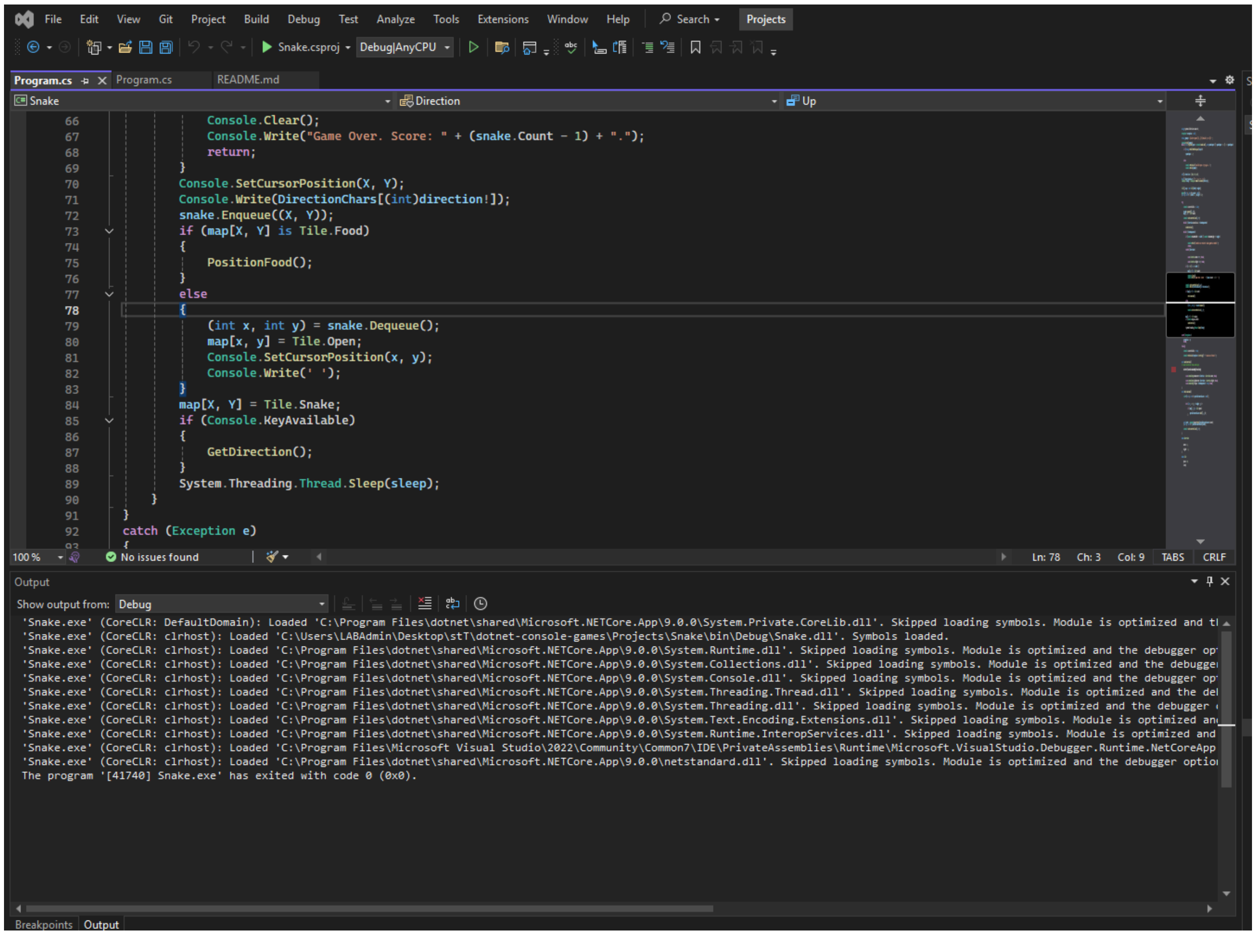
Locals

Name	Value	Type
direction	Right	Direction?
closeRequested	false	bool
map	{Tile[120, 30]}	Tile[,]
height	30	int
width	120	int

Call Stack

Name
Snake.dll!Program.<Main>$.__GetDirection|0_0(ref Program.<>c__DisplayClass0_0 value) Line 107
Snake.dll!Program.<Main>$(string[] args) Line 87

On switching it to right (the opposite direction)



```
                Console.Clear();
                Console.Write("Game Over. Score: " + (snake.Count - 1) + ".");
                return;
            }
            Console.SetCursorPosition(X, Y);
            Console.Write(DirectionChars[(int)direction!]);
            snake.Enqueue((X, Y));
            if (map[X, Y] is Tile.Food)
            {
                PositionFood();
            }
            else
            {
                (int x, int y) = snake.Dequeue();
                map[x, y] = Tile.Open;
                Console.SetCursorPosition(x, y);
                Console.Write(' ');
            }
            map[X, Y] = Tile.Snake;
            if (Console.KeyAvailable)
            {
                GetDirection();
            }
            System.Threading.Thread.Sleep(sleep);
        }
    }
    catch (Exception e)
    {
```

```
'Snake.exe' (CoreCLR: DefaultDomain): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Private.CoreLib.dll'. Skipped loading symbols. Module is optimized and t
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Users\LABAdmin\Desktop\stT\dotnet-console-games\Projects\Snake\bin\Debug\Snake.dll'. Symbols loaded.
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Runtime.dll'. Skipped loading symbols. Module is optimized and the debugger op
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Collections.dll'. Skipped loading symbols. Module is optimized and the debugge
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Console.dll'. Skipped loading symbols. Module is optimized and the debugger op
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Threading.Thread.dll'. Skipped loading symbols. Module is optimized and the de
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Threading.dll'. Skipped loading symbols. Module is optimized and the debugger
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Text.Encoding.Extensions.dll'. Skipped loading symbols. Module is optimized an
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Runtime.InteropServices.dll'. Skipped loading symbols. Module is optimized and
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE\PrivateAssemblies\Runtime\Microsoft.VisualStudio.Debugger.Runtime.NetCoreApp
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\netstandard.dll'. Skipped loading symbols. Module is optimized and the debugger optio
The program '[41740] Snake.exe' has exited with code 0 (0x0).
```

Game ends on switching direction by 180 degrees

3.3 Bug 3: Tanks Game - Null Reference Bug

Context

- What: The game can crash due to a null reference exception in the bullet collision detection.
- When: This occurs when checking for collisions between bullets and tanks.
- Why: The BulletCollisionCheck method sets collidingTank to null but then tries to access it without proper null checking.
- Where: The bug is in the BulletCollisionCheck() method in Projects/Tanks/Program.cs.

Reproduction Steps

- Start the Tanks game

- Fire bullets and play until a bullet collides with a wall or obstacle
- The game may crash due to a null reference exception

Debugging

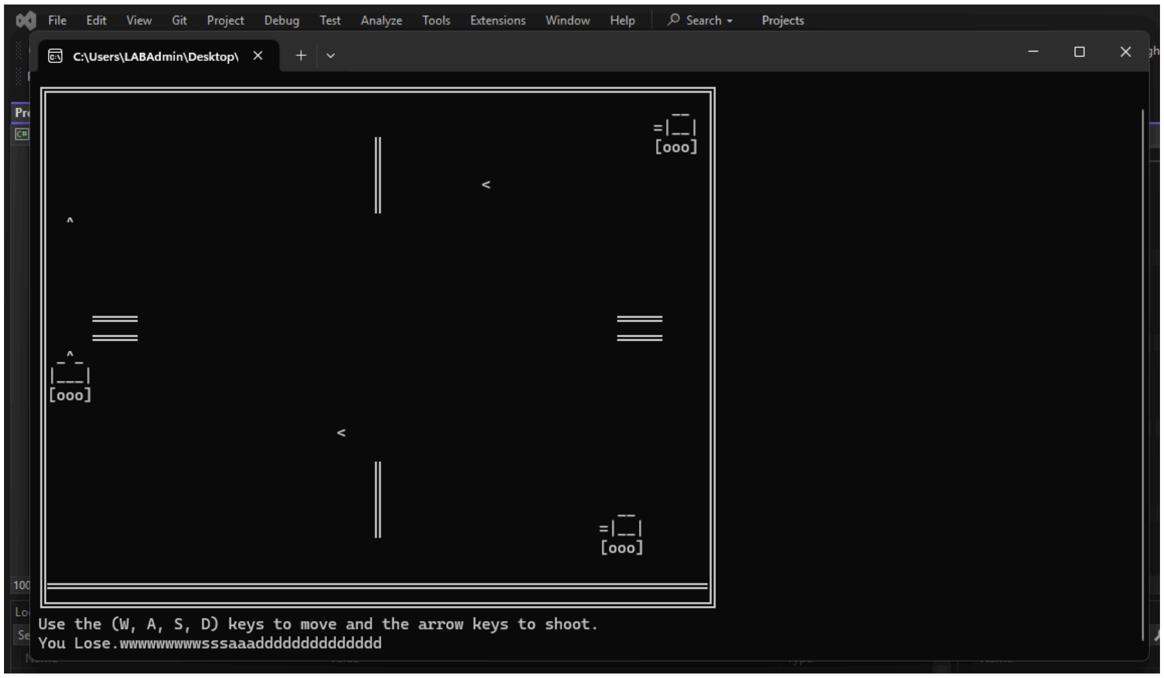
I set breakpoints at various lines in the code, but particularly at line 350 in Projects/Tanks/Program.cs (in the BulletCollisionCheck() method) to observe the potential null reference.

Fix

Added proper null checking before accessing the collidingTank variable:

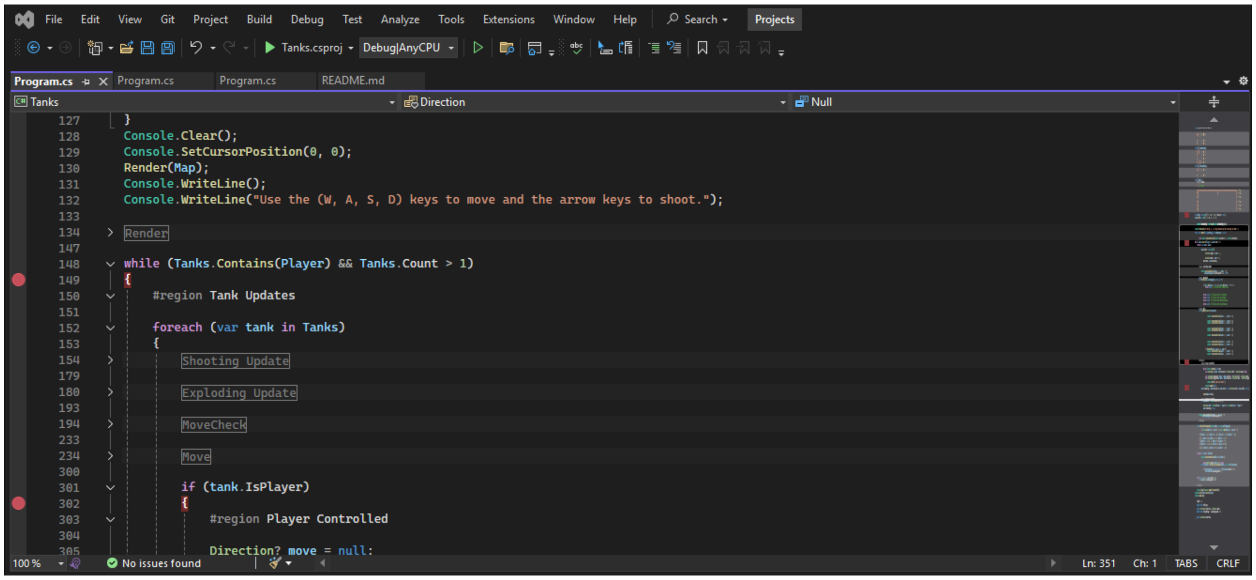
```
bool BulletCollisionCheck(Bullet bullet, out Tank collidingTank)
{
    collidingTank = null;
    foreach (var tank in Tanks)
    {
        if (Math.Abs(bullet.X - tank.X) < 3 && Math.Abs(bullet.Y - tank.Y) < 2)
        {
            collidingTank = tank;
            return true;
        }
    }
    // Other collision checks...
    return false;
}

// In the bullet update code:
if (collision)
{
    if (collisionTank is not null && --collisionTank.Health <= 0)
    {
        collisionTank.ExplodingFrame = 1;
    }
    tank.Bullet = null;
}
```

Visual walkthrough of the bug and debugging



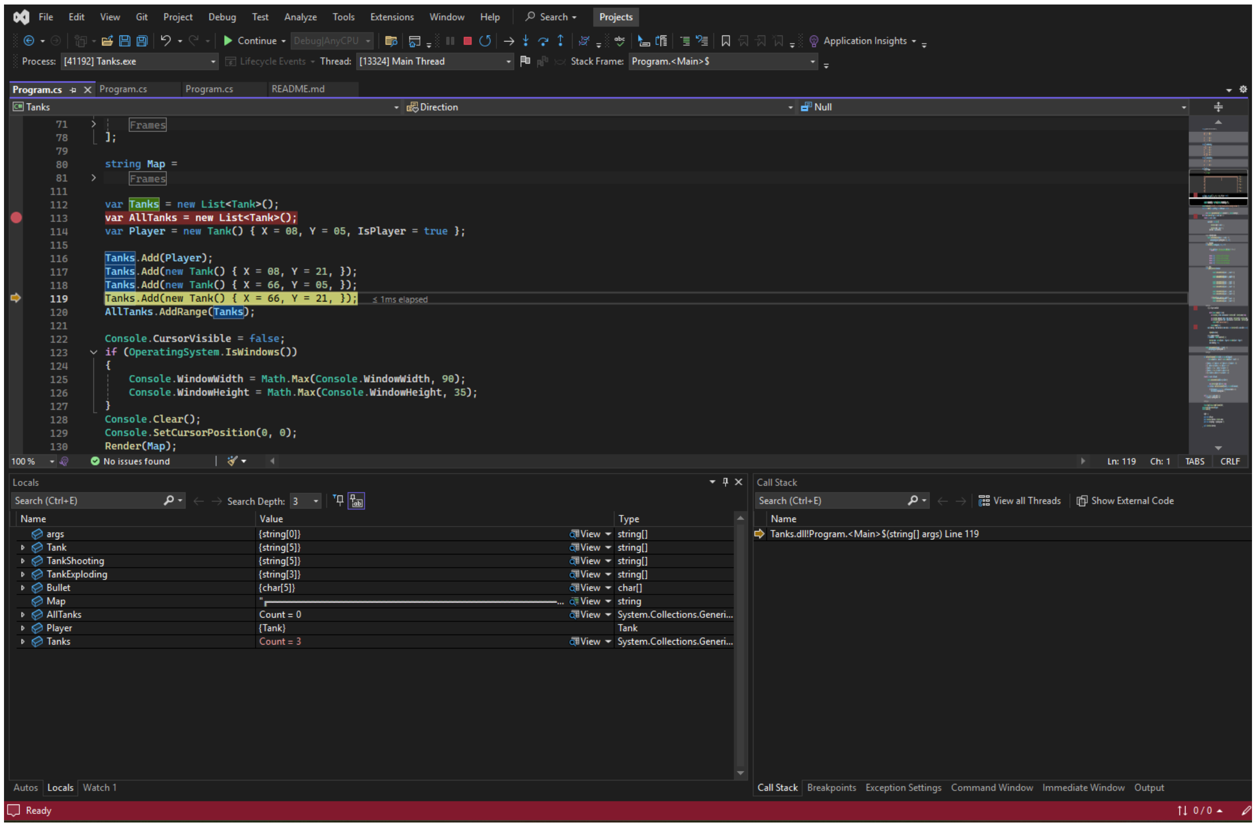
```
Use the (W, A, S, D) keys to move and the arrow keys to shoot.
You Lose.wwwwwwwwwsssaaadddddddddddddd
```

Crash bug found



```
}
Console.Clear();
Console.SetCursorPosition(0, 0);
Render(Map);
Console.WriteLine();
Console.WriteLine("Use the (W, A, S, D) keys to move and the arrow keys to shoot.");

Render

while (Tanks.Contains(Player) && Tanks.Count > 1)
{
    #region Tank Updates

    foreach (var tank in Tanks)
    {
        Shooting Update

        Exploding Update

        MoveCheck

        Move

        if (tank.IsPlayer)
        {
            #region Player Controlled

            Direction? move = null;
```

Setting breakpoints



```
    Frames
];

string Map =
    Frames

var Tanks = new List<Tank>();
var AllTanks = new List<Tank>();
var Player = new Tank() { X = 08, Y = 05, IsPlayer = true };

Tanks.Add(Player);
Tanks.Add(new Tank() { X = 08, Y = 21, });
Tanks.Add(new Tank() { X = 66, Y = 05, });
Tanks.Add(new Tank() { X = 66, Y = 21, });  ≤ 1ms elapsed
AllTanks.AddRange(Tanks);

Console.CursorVisible = false;
if (OperatingSystem.IsWindows())
{
    Console.WindowWidth = Math.Max(Console.WindowWidth, 90);
    Console.WindowHeight = Math.Max(Console.WindowHeight, 35);
}
Console.Clear();
Console.SetCursorPosition(0, 0);
Render(Map);
```

Locals

Name	Value	Type
args	{string[0]}	string[]
Tank	{string[5]}	string[]
TankShooting	{string[5]}	string[]
TankExploding	{string[3]}	string[]
Bullet	{char[5]}	char[]
Map	"	string
AllTanks	Count = 0	System.Collections.Generi...
Player	{Tank}	Tank
Tanks	Count = 3	System.Collections.Generi...

Call Stack

```
Tanks.dll!Program.<Main>$(string[] args) Line 119
```

Monitoring variables at various debugging stages by stepping in and stepping over

3.4 Bug 4: Snake Game: Board Fill Exception

In the Snake game, when the snake grows to fill almost the entire board, the PositionFood() method will eventually encounter a situation where there are no more open tiles available. In this case, possibleCoordinates will be an empty list, and it will throw an ArgumentOutOfRangeException because Random.Shared.Next() cannot accept 0 as an argument (it expects a positive non-zero value).

Context

- What: The game crashes with an `ArgumentOutOfRangeException` when the snake fills the entire board
- When: This occurs when there are no more open tiles to place food.
- Why: The `PositionFood()` method tries to get a random index from an empty list.

- Where: The bug is in the `PositionFood()` method in `Projects/Snake/Program.cs`.

Reproduction Steps

- Start the Snake game
- Play until the snake grows very large, filling most of the board
- Continue playing until there are no more open tiles
- The game will crash with an `ArgumentOutOfRangeException`

Debugging

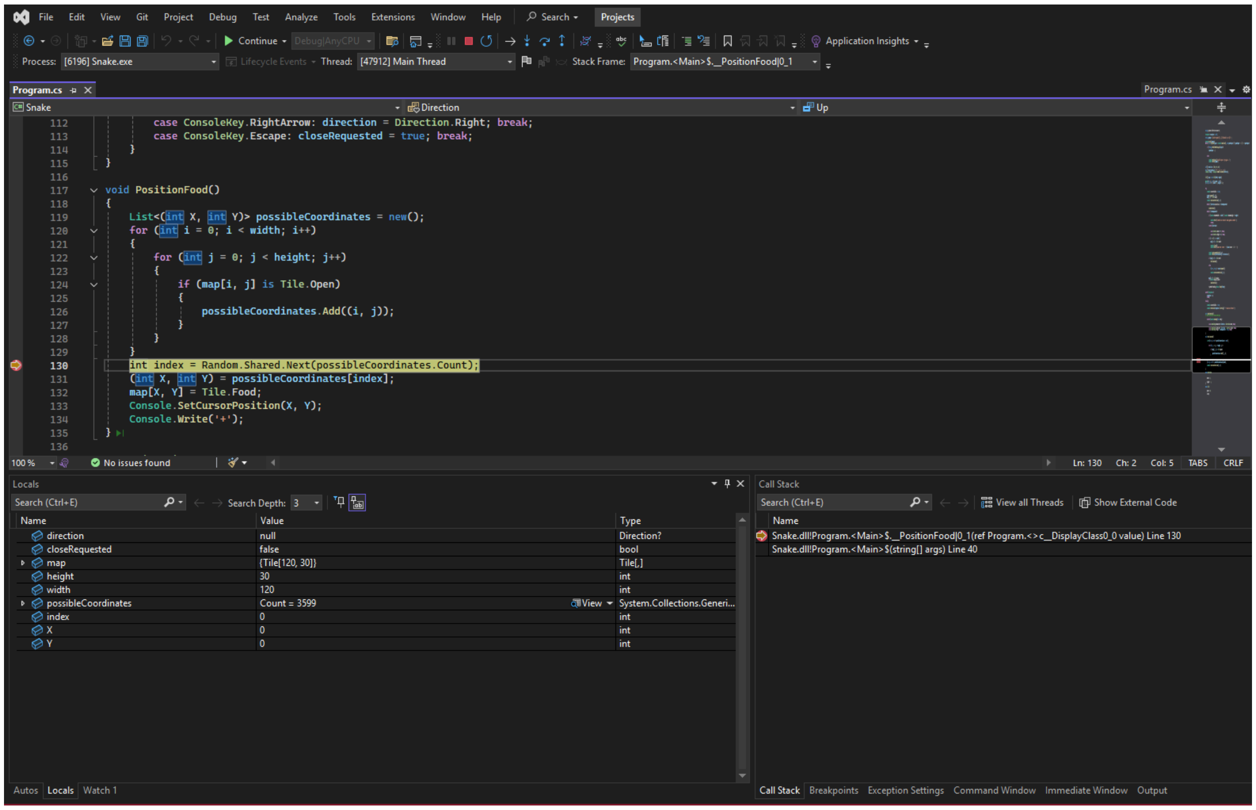
I set a breakpoint at line 130 in `Projects/Snake/Program.cs` (in the `PositionFood()` method) to observe the exception when `possibleCoordinates.Count` is 0.

Fix

Added a check to handle the case when there are no more open tiles:

```
void PositionFood()
{
    List<(int X, int Y)> possibleCoordinates = new();
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < height; j++)
        {
            if (map[i, j] is Tile.Open)
            {
                possibleCoordinates.Add((i, j));
            }
        }
    }

    // Check if there are any open tiles left
    if (possibleCoordinates.Count == 0)
    {
        // The snake has filled the entire board - player wins!
        Console.Clear();
             Console.Write("Congratulations! You've filled the board. You win!
Score: " + snake.Count + ".");
        closeRequested = true;
        return;
    }

    int index = Random.Shared.Next(possibleCoordinates.Count);
    (int X, int Y) = possibleCoordinates[index];
    map[X, Y] = Tile.Food;
    Console.SetCursorPosition(X, Y);
    Console.Write('+');
}
```

Visual walkthrough of the bug and debugging



```
            case ConsoleKey.RightArrow: direction = Direction.Right; break;
            case ConsoleKey.Escape: closeRequested = true; break;
        }
}

void PositionFood()
{
    List<(int X, int Y)> possibleCoordinates = new();
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < height; j++)
        {
            if (map[i, j] is Tile.Open)
            {
                possibleCoordinates.Add((i, j));
            }
        }
    }
    int index = Random.Shared.Next(possibleCoordinates.Count);
    (int X, int Y) = possibleCoordinates[index];
    map[X, Y] = Tile.Food;
    Console.SetCursorPosition(X, Y);
    Console.Write('+');
}
```

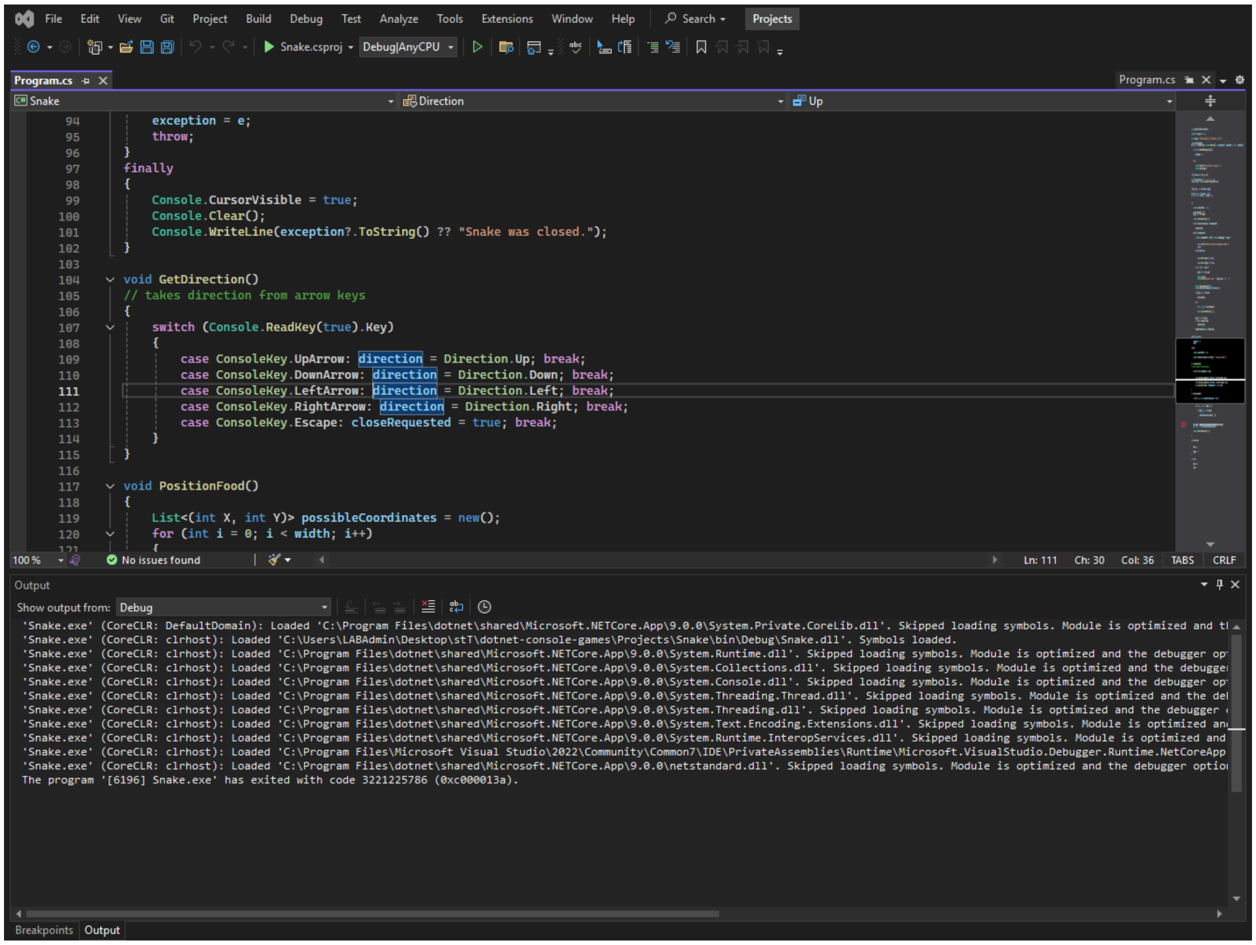
Locals

Name	Value	Type
direction	null	Direction?
closeRequested	false	bool
map	{Tile[120, 30]}	Tile[,]
height	30	int
width	120	int
possibleCoordinates	Count = 3599	System.Collections.Generi...
index	0	int
X	0	int
Y	0	int

Call Stack

Name
Snake.dll!Program.<Main>$.__PositionFood|0_1(ref Program.<>c__DisplayClass0_0 value) Line 130
Snake.dll!Program.<Main>$(string[] args) Line 40

Setting breakpoints



```
            exception = e;
            throw;
        }
        finally
        {
            Console.CursorVisible = true;
            Console.Clear();
            Console.WriteLine(exception?.ToString() ?? "Snake was closed.");
        }

    void GetDirection()
    // takes direction from arrow keys
    {
        switch (Console.ReadKey(true).Key)
        {
            case ConsoleKey.UpArrow: direction = Direction.Up; break;
            case ConsoleKey.DownArrow: direction = Direction.Down; break;
            case ConsoleKey.LeftArrow: direction = Direction.Left; break;
            case ConsoleKey.RightArrow: direction = Direction.Right; break;
            case ConsoleKey.Escape: closeRequested = true; break;
        }
    }

    void PositionFood()
    {
        List<(int X, int Y)> possibleCoordinates = new();
        for (int i = 0; i < width; i++)
        {
```

```
'Snake.exe' (CoreCLR: DefaultDomain): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Private.CoreLib.dll'. Skipped loading symbols. Module is optimized and tl
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Users\LABAdmin\Desktop\stT\dotnet-console-games\Projects\Snake\bin\Debug\Snake.dll'. Symbols loaded.
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Runtime.dll'. Skipped loading symbols. Module is optimized and the debugger op
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Collections.dll'. Skipped loading symbols. Module is optimized and the debugge
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Console.dll'. Skipped loading symbols. Module is optimized and the debugger op
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Threading.Thread.dll'. Skipped loading symbols. Module is optimized and the de
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Threading.dll'. Skipped loading symbols. Module is optimized and the debugger
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Text.Encoding.Extensions.dll'. Skipped loading symbols. Module is optimized an
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\System.Runtime.InteropServices.dll'. Skipped loading symbols. Module is optimized and
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE\PrivateAssemblies\Runtime\Microsoft.VisualStudio.Debugger.Runtime.NetCoreApp
'Snake.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\9.0.0\netstandard.dll'. Skipped loading symbols. Module is optimized and the debugger optio
The program '[6196] Snake.exe' has exited with code 3221225786 (0xc000013a).
```

Program exits on the board being filled up

3.5 Bug 5: Minesweeper: Incorrect Tile Returned

When a player reveals a tile that should reveal multiple empty adjacent tiles, the game either reveals incorrect tiles or crashes due to out-of-bounds array access. This is because the AdjacentTiles() method, which is responsible for determining which adjacent tiles should be checked, returns coordinates in the wrong order.

Context

- What: The game has incorrect coordinate handling in the `AdjacentTiles` method.
- When: This occurs when revealing tiles and checking for adjacent mines.

- Why: The coordinates are swapped in the `yield return` statements, causing incorrect tile checking.
- Where: The bug is in the `AdjacentTiles()` method in `Projects/Minesweeper/Program.cs`.

Reproduction Steps

- Start the Minesweeper game
- Click on a tile that should reveal multiple empty tiles
- The game may reveal incorrect tiles or crash due to out-of-bounds access

Debugging

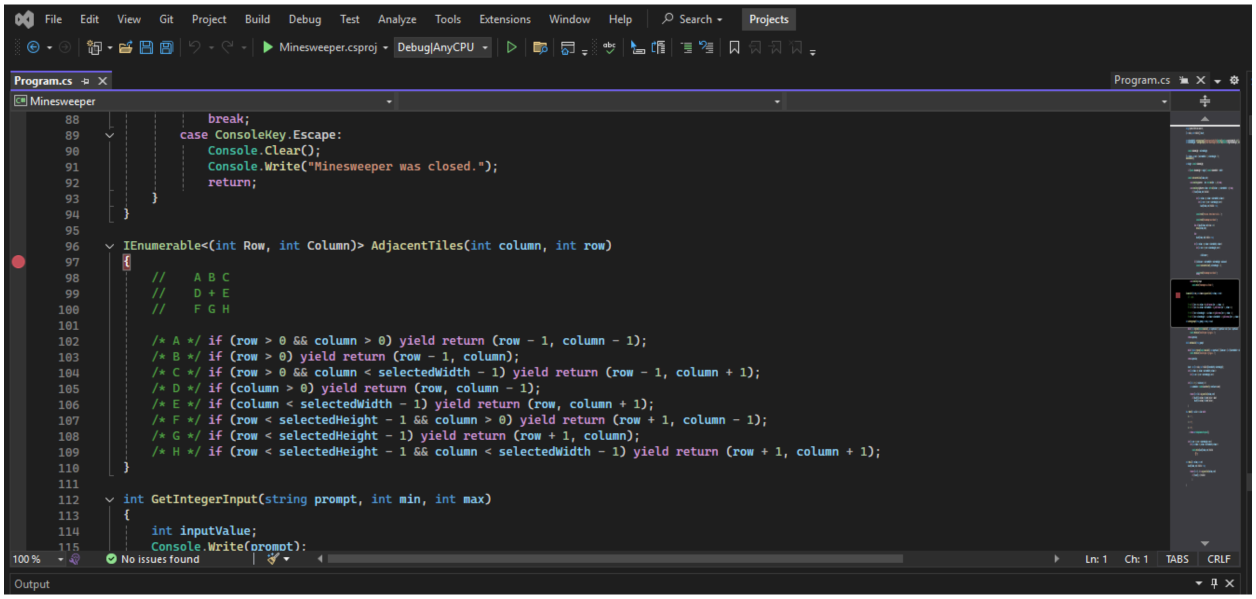
I set a breakpoint at line 96 in `Projects/Minesweeper/Program.cs` (the `AdjacentTiles()` method) to observe how coordinates are returned in the wrong order.

Fix

Corrected the coordinate order in the `AdjacentTiles` method:

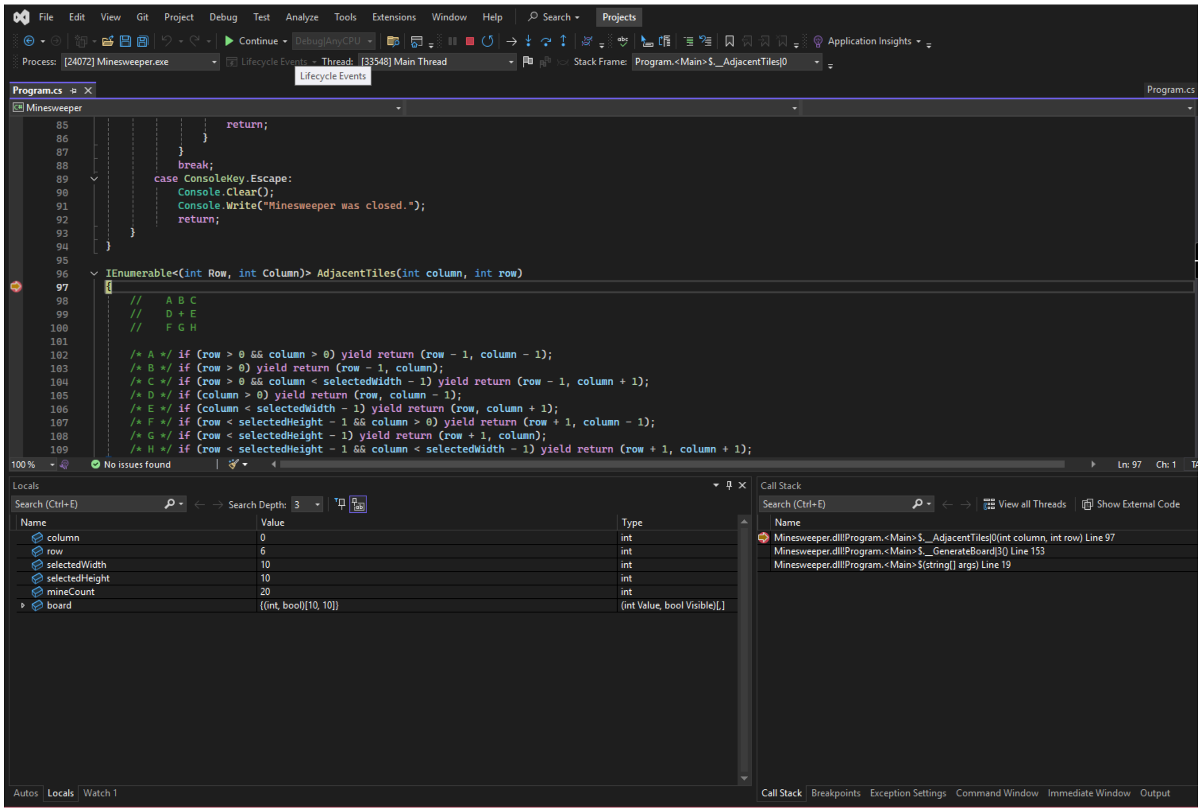
```
IEnumerable<(int Column, int Row)> AdjacentTiles(int column, int row)
{
    //    A B C
    //    D + E
    //    F G H

    /* A */ if (row > 0 && column > 0) yield return (column - 1, row - 1);
    /* B */ if (row > 0) yield return (column, row - 1);
    /* C */ if (row > 0 && column < selectedWidth - 1) yield return (column +
1, row - 1);
    /* D */ if (column > 0) yield return (column - 1, row);
    /* E */ if (column < selectedWidth - 1) yield return (column + 1, row);
    /* F */ if (row < selectedHeight - 1 && column > 0) yield return (column -
1, row + 1);
    /* G */ if (row < selectedHeight - 1) yield return (column, row + 1);
    /* H */ if (row < selectedHeight - 1 && column < selectedWidth - 1) yield
return (column + 1, row + 1);
}
```

Visual walkthrough of the bug and debugging



```
                break;
            case ConsoleKey.Escape:
                Console.Clear();
                Console.Write("Minesweeper was closed.");
                return;
        }
    }

    IEnumerable<(int Row, int Column)> AdjacentTiles(int column, int row)
    {
        //    A B C
        //    D + E
        //    F G H

        /* A */ if (row > 0 && column > 0) yield return (row - 1, column - 1);
        /* B */ if (row > 0) yield return (row - 1, column);
        /* C */ if (row > 0 && column < selectedWidth - 1) yield return (row - 1, column + 1);
        /* D */ if (column > 0) yield return (row, column - 1);
        /* E */ if (column < selectedWidth - 1) yield return (row, column + 1);
        /* F */ if (row < selectedHeight - 1 && column > 0) yield return (row + 1, column - 1);
        /* G */ if (row < selectedHeight - 1) yield return (row + 1, column);
        /* H */ if (row < selectedHeight - 1 && column < selectedWidth - 1) yield return (row + 1, column + 1);
    }

    int GetIntegerInput(string prompt, int min, int max)
    {
        int inputValue;
        Console.Write(prompt);
```

Setting breakpoints



```
                        return;
                    }
                }
                break;
            case ConsoleKey.Escape:
                Console.Clear();
                Console.Write("Minesweeper was closed.");
                return;
        }
    }

    IEnumerable<(int Row, int Column)> AdjacentTiles(int column, int row)
    {
        //    A B C
        //    D + E
        //    F G H

        /* A */ if (row > 0 && column > 0) yield return (row - 1, column - 1);
        /* B */ if (row > 0) yield return (row - 1, column);
        /* C */ if (row > 0 && column < selectedWidth - 1) yield return (row - 1, column + 1);
        /* D */ if (column > 0) yield return (row, column - 1);
        /* E */ if (column < selectedWidth - 1) yield return (row, column + 1);
        /* F */ if (row < selectedHeight - 1 && column > 0) yield return (row + 1, column - 1);
        /* G */ if (row < selectedHeight - 1) yield return (row + 1, column);
        /* H */ if (row < selectedHeight - 1 && column < selectedWidth - 1) yield return (row + 1, column + 1);
```

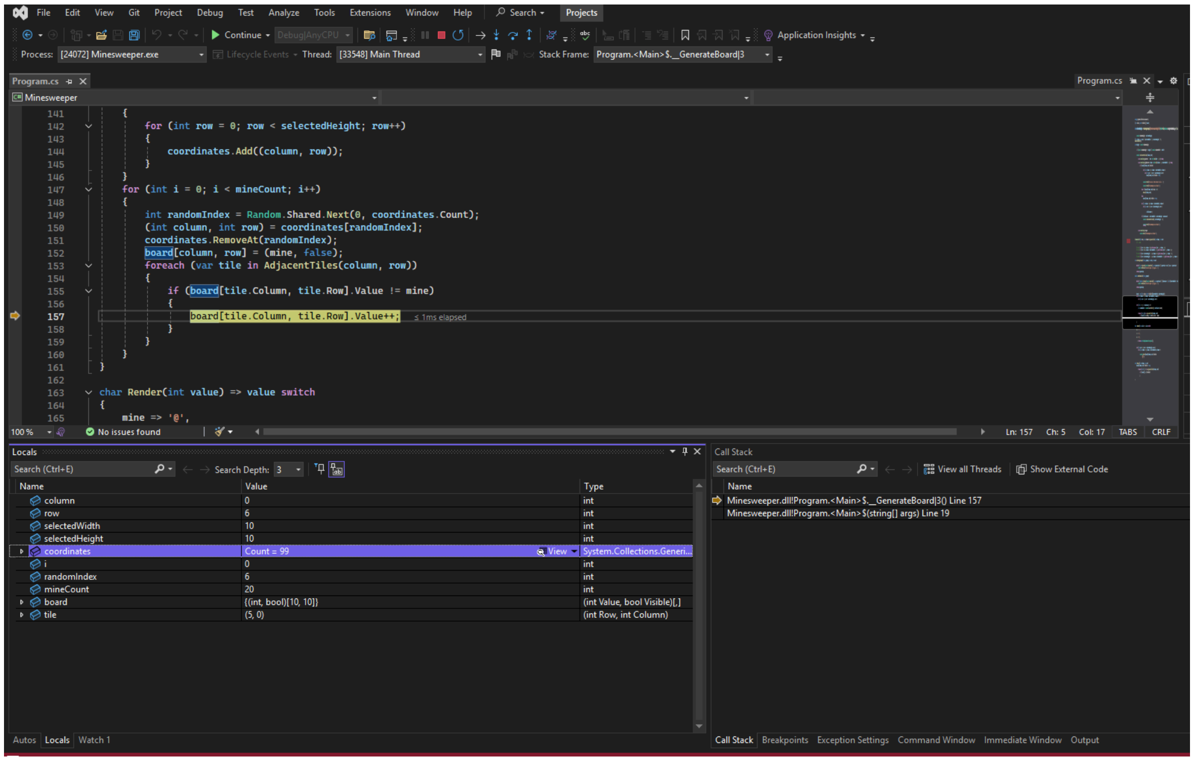
Locals

Name	Value	Type
column	0	int
row	6	int
selectedWidth	10	int
selectedHeight	10	int
mineCount	20	int
board	{(int, bool)[10, 10]}	(int Value, bool Visible)[,]

Call Stack

Name
Minesweeper.dll!Program.<Main>$.__AdjacentTiles|0(int column, int row) Line 97
Minesweeper.dll!Program.<Main>$.__GenerateBoard|3() Line 153
Minesweeper.dll!Program.<Main>$(string[] args) Line 19

Variables right after selecting board size (see row and column)



```
        {
            for (int row = 0; row < selectedHeight; row++)
            {
                coordinates.Add((column, row));
            }
        }
        for (int i = 0; i < mineCount; i++)
        {
            int randomIndex = Random.Shared.Next(0, coordinates.Count);
            (int column, int row) = coordinates[randomIndex];
            coordinates.RemoveAt(randomIndex);
            board[column, row] = (mine, false);
            foreach (var tile in AdjacentTiles(column, row))
            {
                if (board[tile.Column, tile.Row].Value != mine)
                {
                    board[tile.Column, tile.Row].Value++;  ≤ 1ms elapsed
                }
            }
        }
    }

    char Render(int value) => value switch
    {
        mine => '@',
```

Locals

Name	Value	Type
column	0	int
row	6	int
selectedWidth	10	int
selectedHeight	10	int
coordinates	Count = 99	System.Collections.Generi...
i	0	int
randomIndex	6	int
mineCount	20	int
board	{(int, bool)[10, 10]}	(int Value, bool Visible)[,]
tile	(5, 0)	(int Row, int Column)

Call Stack

```
Minesweeper.dll!Program.<Main>$.__GenerateBoard|3() Line 157
Minesweeper.dll!Program.<Main>$(string[] args) Line 19
```

Stepping into the breakpoint and observing coordinate values again

3.7 Answering the questions mentioned in the lab assignment document:

1. **If there is no Main() method in the program, where exactly is the entry point?**
    - In recent .NET versions (notably from .NET 6 onward), developers can omit the traditional Main() method by leveraging a feature known as top-level statements. When this approach is used, the C# compiler implicitly creates a Main() method during compilation.
    - Many of the console games analyzed make use of this feature by placing code directly at the file level, rather than encapsulating it within an explicitly defined Main() method. The compiler then wraps these top-level statements into a generated entry point.
    - For instance, in the Snake game, the code begins immediately with variable declarations and gameplay logic. Although

there's no visible Main() method, the compiler synthesizes one that sequentially executes the provided statements.
- This feature simplifies development for console applications by reducing boilerplate, while the actual program entry point remains a Main() method—implicitly handled by the compiler.

4. Conclusion

The methodology and execution described in this report offered a systematic framework for detecting, analyzing, and resolving bugs within the .NET Console Games collection. Using this structured approach, we successfully identified and addressed 5 major bugs that could lead to game crashes. These issues had the potential to crash games due to problems such as:

- Accessing arrays out of bounds
- Null reference exceptions
- Index out of range errors
- Invalid direction changes disrupting game logic, and other issues of the same sort.

By placing breakpoints at key locations in the code, I was able to observe these bugs in real time and understand how the applied fixes resolved the underlying issues. The solutions focused on implementing proper validation, bounds checking, and handling of null references to prevent runtime failures.

However, the whole process was also really challenging because sometimes in spite of having an intuitive and code-based idea about the presence of a bug, it is difficult to visualise it using the debugger, and because in some cases it was the corner cases which threw bugs, it was not always possible to reach those game states (unless all we did was keep playing these console games, which we can't :)).

5. References

[1] Lecture Slides
[2] https://learn.microsoft.com/en-us/dotnet/csharp
[3] https://github.com/dotnet/dotnet-console-games
[4] https://visualstudio.microsoft.com
[5] https://dotnet.microsoft.com/en-us/download
[6]https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions
[7]https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/exceptions/exception-handling
[8] https://learn.microsoft.com/en-us/visualstudio/debugger
[9] https://github.com/dotnet/dotnet-console-games

CS202: Software Tools and Techniques

Lab 12 Report: Event-driven Programming for Windows Forms Apps in C#

Jiya Bhavin Desai | Roll No. 22110107 | 17.04.2025

1. Introduction

This lab assignment focused on event-driven programming using C\# in Windows Forms applications. Event-driven programming represents a paradigm where program execution is determined by events such as user actions (clicks, key presses) or system events rather than a predetermined sequence. This approach is particularly useful for developing interactive applications with graphical user interfaces.

The assignment consists of two main tasks:

- Developing a C# console application that accepts a target time from the user, continuously checks it against the current system time, and raises a user-defined event to trigger an alarm when the times match.
- Converting the console application into a Windows Forms application with graphical elements, where the background color changes every second and stops when the target time is reached.

Both tasks require implementing the publisher/subscriber model, a fundamental pattern in event-driven programming where:

- The publisher defines and raises events
- The subscriber listens for events and responds with appropriate actions

This lab provides hands-on experience with C# event handling mechanisms, Windows Forms development, and the Visual Studio integrated development environment.

1.1 Setup

- Since I didn't have enough space on my laptop for installing Visual Studio 2022 (needed for building the C# console applications), I worked with one of the PCs present in the computer lab, which already had Visual Studio 2022 installed, with support for .NET framework.

1.2 Tools

- **Operating System:** Windows 10
- **IDE:** Visual Studio 2022 (Community Edition)
- **Target Framework:** .NET *(version 6 wasn't available in the lab PC, so I used an earlier version)*
- **Programming Language:** C# (latest stable version as available in the lab PC)

2. Methodology and Execution

2.1 Task 1: Console Application with Custom Event

The first task involved creating a C# console application that:

- Accepts a target time from the user in HH:MM:SS format
- Continuously checks this target time against the current system time
- Raises a custom event named raiseAlarm when the times match
- Executes a function Ring_alarm() in response to the event

```
using System;
using System.Threading;

namespace AlarmClockConsole
{
    // Publisher class that will raise the event
    public class AlarmClock
    {
        // Define the delegate for the event
        public delegate void AlarmEventHandler(object source, EventArgs args);

        // Define the event using the delegate
        public event AlarmEventHandler raiseAlarm;

        // Method to start the alarm clock
```

```
               public void Start(TimeSpan targetTime)
               {
                   Console.WriteLine("Alarm clock started. Waiting for the target time...");

                   // Keep checking until the target time is reached
                   while (true)
                   {
                       // Get current time
                       TimeSpan currentTime = DateTime.Now.TimeOfDay;

                       // Format and display current time
                                                            Console.WriteLine($"Current   time:
{currentTime.Hours:D2}:{currentTime.Minutes:D2}:{currentTime.Seconds:D2}");

                       // Check if current time matches target time
                       if (currentTime.Hours == targetTime.Hours &&
                           currentTime.Minutes == targetTime.Minutes &&
                           currentTime.Seconds == targetTime.Seconds)
                       {
                           // Raise the event
                           OnRaiseAlarm();
                           break;
                       }

                       // Wait for 1 second before checking again
                       Thread.Sleep(1000);
                   }
               }

               // Method to raise the event
               protected virtual void OnRaiseAlarm()
               {
                   if (raiseAlarm != null)
                   {
                       raiseAlarm(this, EventArgs.Empty);
                   }
               }
           }

           // Subscriber class
           public class AlarmListener
           {
               // Method that will be called when the event is raised
               public void Ring_alarm(object source, EventArgs args)
               {
                   Console.WriteLine("ALARM! The target time has been reached!");
                   Console.WriteLine("Ring! Ring! Ring!");
               }
           }

           class Program
           {
               static void Main(string[] args)
               {
                   Console.WriteLine("Alarm Clock Application");
                   Console.WriteLine("======================");

                   // Get target time from user
                   Console.Write("Enter target time in HH:MM:SS format: ");
                   string timeInput = Console.ReadLine();

                   try
```

```
                {
                    // Parse the time input
                    string[] timeParts = timeInput.Split(':');
                    if (timeParts.Length != 3)
                    {
                        throw new FormatException("Invalid time format. Please use HH:MM:SS
format.");
                    }

                    int hours = int.Parse(timeParts[0]);
                    int minutes = int.Parse(timeParts[1]);
                    int seconds = int.Parse(timeParts[2]);

                    // Validate time components
                    if (hours < 0 || hours > 23 || minutes < 0 || minutes > 59 || seconds <
0 || seconds > 59)
                    {
                         throw new ArgumentOutOfRangeException("Invalid time values. Hours
should be 0-23, minutes and seconds should be 0-59.");
                    }

                    // Create target TimeSpan
                    TimeSpan targetTime = new TimeSpan(hours, minutes, seconds);

                    // Create publisher and subscriber
                    AlarmClock alarmClock = new AlarmClock();
                    AlarmListener listener = new AlarmListener();

                    // Subscribe to the event
                    alarmClock.raiseAlarm += listener.Ring_alarm;

                    // Start the alarm clock
                    alarmClock.Start(targetTime);
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Error: {ex.Message}");
                }

                Console.WriteLine("Press any key to exit...");
                Console.ReadKey();
            }
        }
    }
```

2.2 Task 2: Windows Forms Application

The second task involved converting the console application into a Windows Forms application where:

- The user enters the target time in a TextBox control on a Form
- The user clicks a Start button to begin the alarm process
- The background color of the Form changes every second

- When the target time is reached, the color changes stop and a message box appears

The Windows Forms application was implemented with the following three key files:

1. Form1.cs - Main Logic: This file contains the business logic, event handling, and user interface interactions:

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

namespace AlarmClockForm
{
    public partial class Form1 : Form
    {
        // Define the delegate for the event
        public delegate void AlarmEventHandler(object source, EventArgs args);

        // Define the event using the delegate
        public event AlarmEventHandler raiseAlarm;

        // Random for generating colors
        private Random random = new Random();

        // Timer for updating the form
        private System.Windows.Forms.Timer colorTimer;

        // Flag to indicate if the alarm is running
        private bool isAlarmRunning = false;

        // Target time
        private TimeSpan targetTime;

        public Form1()
        {
            InitializeComponent();

            // Subscribe to the event
            raiseAlarm += Ring_alarm;

            // Initialize the timer
            colorTimer = new System.Windows.Forms.Timer();
            colorTimer.Interval = 1000; // 1 second
            colorTimer.Tick += ColorTimer_Tick;
        }

        // Method to handle the button click event
        private void btnStart_Click(object sender, EventArgs e)
        {
            if (isAlarmRunning)
            {
                MessageBox.Show("Alarm is already running.");
                return;
            }

            try
            {
                // Parse the time input
                string timeInput = txtTime.Text;
                string[] timeParts = timeInput.Split(':');
                if (timeParts.Length != 3)
            {
                throw new FormatException("Invalid time format. Please use HH:MM:SS format.");
            }

            int hours = int.Parse(timeParts[0]);
            int minutes = int.Parse(timeParts[1]);
            int seconds = int.Parse(timeParts[2]);

            // Validate time components
            if (hours < 0 || hours > 23 || minutes < 0 || minutes > 59 || seconds < 0 || seconds >
59)
            {
                 throw new ArgumentOutOfRangeException("Invalid time values. Hours should be 0-23,
minutes and seconds should be 0-59.");
            }

            // Create target TimeSpan
            targetTime = new TimeSpan(hours, minutes, seconds);

            // Start the alarm
            StartAlarm();
        }
        catch (Exception ex)
        {
                    MessageBox.Show($"Error: {ex.Message}", "Input Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }

    // Method to start the alarm
    private void StartAlarm()
    {
        isAlarmRunning = true;
        colorTimer.Start();

        // Start a task to check for the target time
        Task.Run(() => CheckAlarmTime());
    }

    // Method to check if the current time matches the target time
    private async void CheckAlarmTime()
    {
        while (isAlarmRunning)
        {
            // Get current time
            TimeSpan currentTime = DateTime.Now.TimeOfDay;

            // Update the current time display on the UI thread
            Invoke(new Action(() =>
            {
                                                    lblCurrentTime.Text = $"Current Time:
{currentTime.Hours:D2}:{currentTime.Minutes:D2}:{currentTime.Seconds:D2}";
                                                     lblTargetTime.Text = $"Target Time:
{targetTime.Hours:D2}:{targetTime.Minutes:D2}:{targetTime.Seconds:D2}";
            }));

            // Check if current time matches target time
            if (currentTime.Hours == targetTime.Hours &&
                currentTime.Minutes == targetTime.Minutes &&
                currentTime.Seconds == targetTime.Seconds)
            {
                // Use Invoke to call the method on the UI thread
                Invoke(new Action(() => OnRaiseAlarm()));
                break;
            }

            // Wait for 100 milliseconds before checking again
            await Task.Delay(100);
        }
    }
```

```
        // Method to raise the event
        protected virtual void OnRaiseAlarm()
        {
            if (raiseAlarm != null)
            {
                raiseAlarm(this, EventArgs.Empty);
            }
        }

        // Method to handle the timer tick event
        private void ColorTimer_Tick(object sender, EventArgs e)
        {
            // Change the background color
            this.BackColor = GetRandomColor();
        }

        // Method to get a random color
        private Color GetRandomColor()
        {
            return Color.FromArgb(random.Next(256), random.Next(256), random.Next(256));
        }

        // Method that will be called when the event is raised
        public void Ring_alarm(object source, EventArgs args)
        {
            // Stop the timer
            colorTimer.Stop();
            isAlarmRunning = false;

            // Show the message
            MessageBox.Show("ALARM! The target time has been reached!", "Alarm", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
        }
    }
}
```

2. Program.cs - Application Entry Point: This file contains the application entry point and initialization:

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

namespace AlarmClockForm
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

  Application.SetCompatibleTextRenderingDefault(false);
```

```
                Application.Run(new Form1());
            }
        }
}
```

---

3. Form1.Designer.cs - UI Definition: This file contains the designer-generated code that defines the user interface:

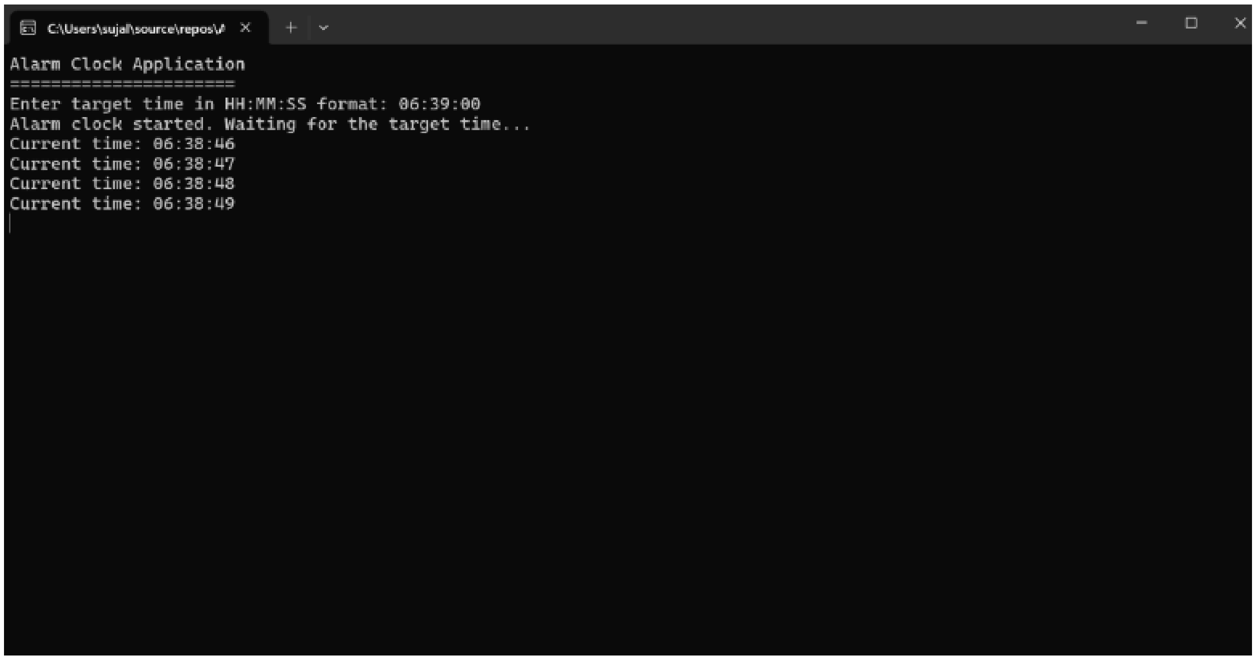
```
namespace AlarmClockForm
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
         /// <param name="disposing">true if managed resources should be disposed;
    otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.txtTime = new System.Windows.Forms.TextBox();
            this.btnStart = new System.Windows.Forms.Button();
            this.lblCurrentTime = new System.Windows.Forms.Label();
            this.lblTargetTime = new System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(12, 15);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(192, 17);
            this.label1.TabIndex = 0;
            this.label1.Text = "Enter time (HH:MM:SS format):";
            //
            // txtTime
            //
            this.txtTime.Location = new System.Drawing.Point(210, 12);
            this.txtTime.Name = "txtTime";
            this.txtTime.Size = new System.Drawing.Size(100, 22);
            this.txtTime.TabIndex = 1;
            //
            // btnStart
            //
            this.btnStart.Location = new System.Drawing.Point(316, 11);
            this.btnStart.Name = "btnStart";
            this.btnStart.Size = new System.Drawing.Size(75, 23);
            this.btnStart.TabIndex = 2;
            this.btnStart.Text = "Start";
            this.btnStart.UseVisualStyleBackColor = true;
            this.btnStart.Click += new System.EventHandler(this.btnStart_Click);
            //
            // lblCurrentTime
            //
            this.lblCurrentTime.AutoSize = true;
            this.lblCurrentTime.Location = new System.Drawing.Point(12, 50);
            this.lblCurrentTime.Name = "lblCurrentTime";
            this.lblCurrentTime.Size = new System.Drawing.Size(97, 17);
            this.lblCurrentTime.TabIndex = 3;
            this.lblCurrentTime.Text = "Current Time: ";
            //
            // lblTargetTime
            //
            this.lblTargetTime.AutoSize = true;
            this.lblTargetTime.Location = new System.Drawing.Point(210, 50);
            this.lblTargetTime.Name = "lblTargetTime";
            this.lblTargetTime.Size = new System.Drawing.Size(95, 17);
            this.lblTargetTime.TabIndex = 4;
            this.lblTargetTime.Text = "Target Time: ";
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(400, 100);
            this.Controls.Add(this.lblTargetTime);
            this.Controls.Add(this.lblCurrentTime);
            this.Controls.Add(this.btnStart);
            this.Controls.Add(this.txtTime);
            this.Controls.Add(this.label1);
            this.Name = "Form1";
            this.Text = "Alarm Clock";
            this.ResumeLayout(false);
            this.PerformLayout();
        }

        #endregion

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox txtTime;
        private System.Windows.Forms.Button btnStart;
        private System.Windows.Forms.Label lblCurrentTime;
        private System.Windows.Forms.Label lblTargetTime;
    }
}
```

3. Results and Analysis

3.1 Task 1: Console Application Results

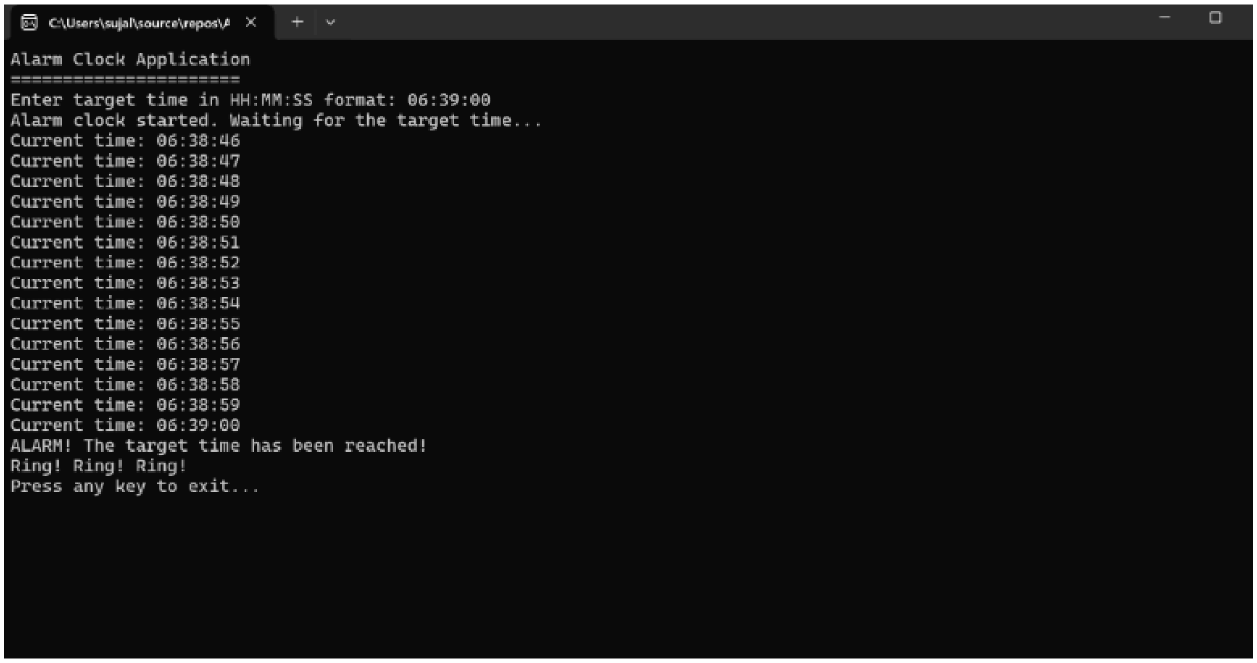
The console application successfully implemented the event-driven behavior as specified in the requirements. When executed, it:

- Prompted the user to enter a target time
- Continuously displayed the current system time
- Triggered the alarm when the target time was reached



```
C:\Users\sujal\source\repos\

Alarm Clock Application
=======================
Enter target time in HH:MM:SS format: 06:39:00
Alarm clock started. Waiting for the target time...
Current time: 06:38:46
Current time: 06:38:47
Current time: 06:38:48
Current time: 06:38:49
```

Initial execution of Task 1 console application, showing time entry and continuous time updates



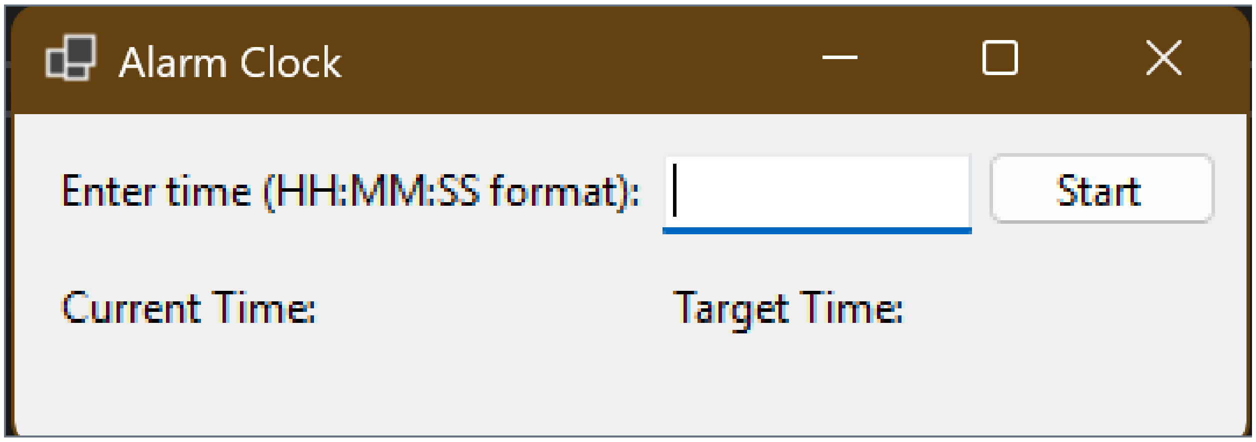
```
Alarm Clock Application
======================
Enter target time in HH:MM:SS format: 06:39:00
Alarm clock started. Waiting for the target time...
Current time: 06:38:46
Current time: 06:38:47
Current time: 06:38:48
Current time: 06:38:49
Current time: 06:38:50
Current time: 06:38:51
Current time: 06:38:52
Current time: 06:38:53
Current time: 06:38:54
Current time: 06:38:55
Current time: 06:38:56
Current time: 06:38:57
Current time: 06:38:58
Current time: 06:38:59
Current time: 06:39:00
ALARM! The target time has been reached!
Ring! Ring! Ring!
Press any key to exit...
```

Task 1 after the alarm triggers, showing the Ring_alarm() function execution

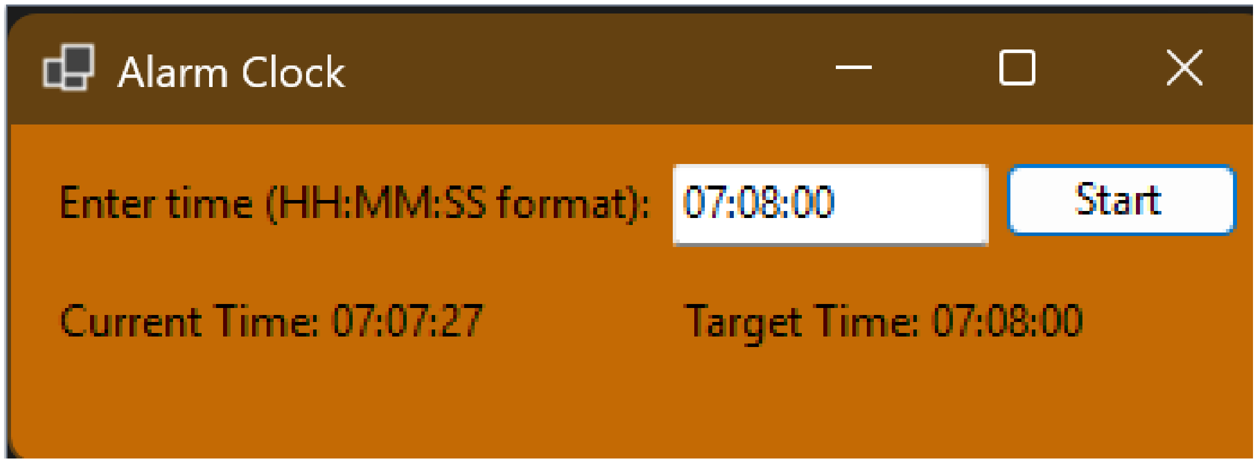
3.2 Task 2: Windows Forms Application Results

The Windows Forms application successfully extended the functionality of the console application into a graphical interface. The application:

- Provided a user-friendly interface for time input
- Visually indicated alarm activity through background color changes
- Displayed a message box when the target time was reached

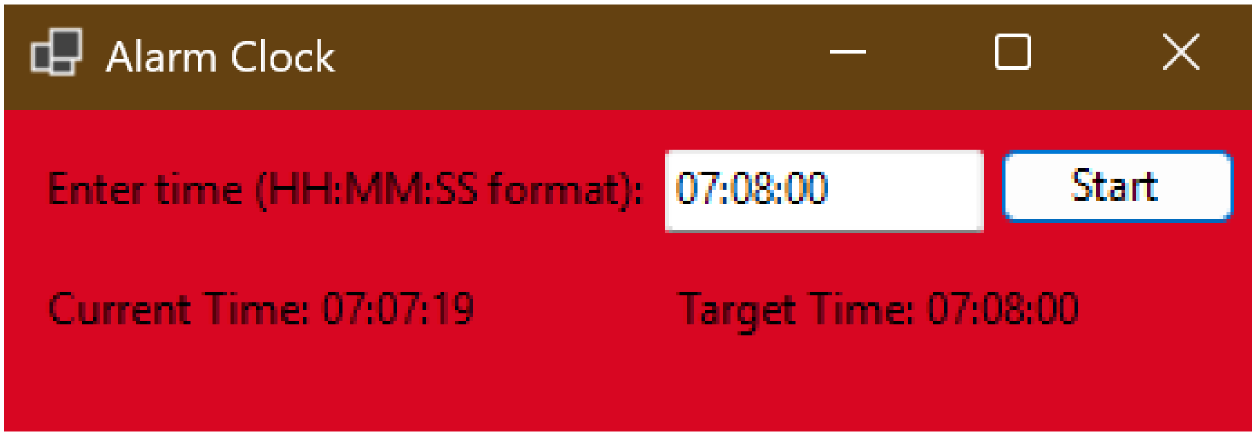


Initial UI of Task 2 Windows Forms application, showing time input field and start button



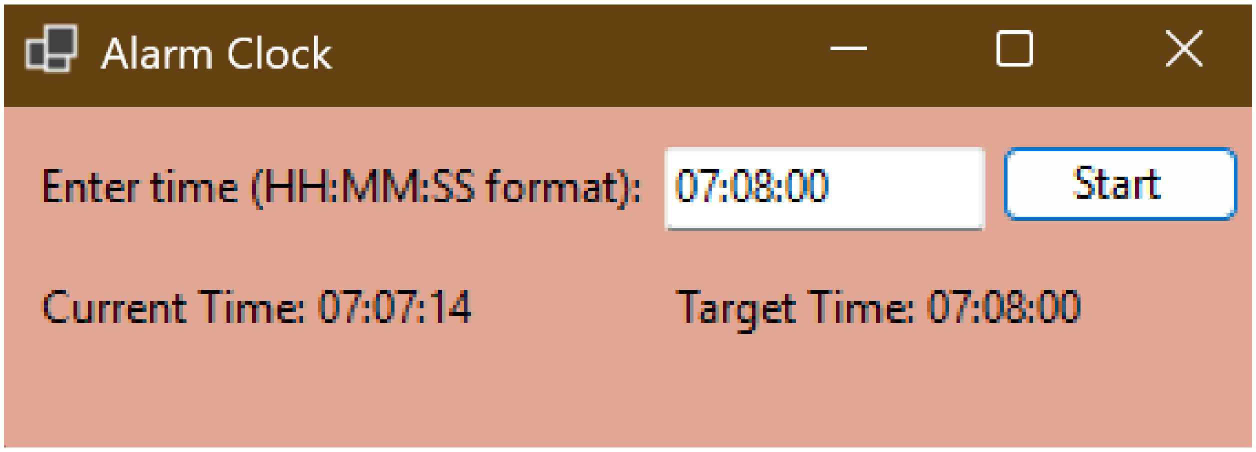
Current Time: 07:07:27 Target Time: 07:08:00

Task 2 showing the first background color change



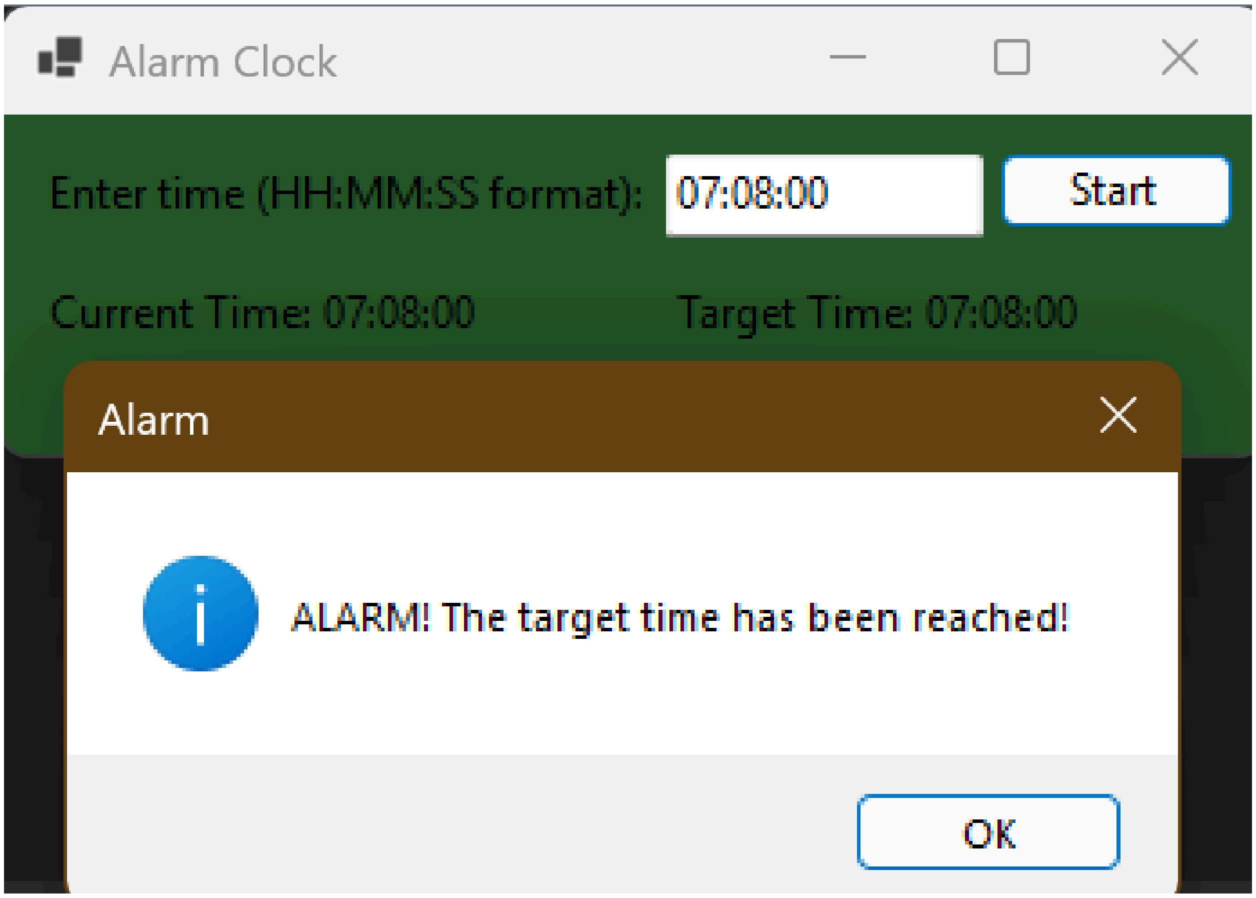
Current Time: 07:07:19 Target Time: 07:08:00

Task 2 showing the second background color change



Current Time: 07:07:14 Target Time: 07:08:00

Task 2 showing the third background color change


Alarm Clock

Enter time (HH:MM:SS format): 07:08:00

Current Time: 07:08:00 Target Time: 07:08:00

Alarm

ALARM! The target time has been reached!

Task 2 alarm message box displayed when the target time is reached

3.3 Analysis of Event-Driven Model Implementation

Both tasks successfully implemented the publisher/subscriber model, with the following key components:

1. Publisher Components
    - Delegate Definition: AlarmEventHandler delegate defined the signature for event handlers
    - Event Declaration: raiseAlarm event created using the delegate
    - Event Raising: OnRaiseAlarm() method triggered the event when conditions were met

2. Subscriber Components
    - Event Handler: Ring_alarm() method defined the response to the event

- Event Registration: The handler was registered with the event using the += operator

3. Task 1 vs. Task 2 Implementation Differences}
   The implementation of the event-driven model in both tasks highlights several key differences:

4. Key Learning Points
   The implementation of the two tasks provided several important learning points about event-driven programming in C#:
   - Event Declaration and Subscription: Understanding how to define, raise, and subscribe to events using delegates
   - Thread Safety: The Windows Forms application demonstrated the importance of usingInvoke() to update UI elements from background threads
   - Asynchronous Programming: Task 2 utilized Task.Run() and async/await pattern for non-blocking time checking
   - UI Event Handling: Working with built-in Windows Forms events like button clicks and timer ticks
   - Exception Handling: Both implementations included robust exception handling for user input validation

3.4 Discussion

1. **Windows Forms vs. Console Applications**
   The transition from a console application to a Windows Forms application highlighted several important differences between the two approaches:
   - User Experience: The Windows Forms application provided a more intuitive and visually appealing interface, making it easier for users to interact with the application.
   - Development Complexity: The Windows Forms application required more code and consideration of UI design principles, thread safety, and asynchronous programming patterns.
   - Event Handling: While both applications used custom events, the Windows Forms application also leveraged built-in events

like button clicks and timer ticks, demonstrating the rich event model in the .NET framework.
- Visual Studio Integration: Windows Forms applications benefit from Visual Studio's designer, making it easier to create and modify UI components.

2. **Challenges Encountered**
   During the implementation of the tasks, several challenges were encountered and addressed:
   - Thread Safety: In the Windows Forms application, updating UI elements from the background thread required using the Invoke() method to marshal calls to the UI thread.
   - Time Comparison Precision: Both applications needed careful implementation of time comparison logic to ensure the alarm triggered exactly when the target time was reached.
   - Exception Handling: Robust handling of user input errors was essential to prevent application crashes and provide meaningful error messages.

3. **Future Improvements**
   Several potential improvements could be made to the applications:
   - Persistent Settings: Add the ability to save and load alarm times.
   - Multiple Alarms: Extend the applications to support setting multiple alarms.
   - Snooze Functionality: Add the ability to snooze the alarm for a specified duration.
   - Advanced UI: Enhance the Windows Forms application with more sophisticated UI elements like a time picker control.
   - Sound Options: Add the ability to select different alarm sounds or use custom audio files.

4. Conclusion

This lab assignment successfully demonstrated the implementation of event-driven programming principles in both console and Windows Forms applications using C#. The key concepts explored included:

- Understanding and implementing the publisher/subscriber model using C# events and delegates
- Transitioning from a console-based application to a graphical user interface
- Managing UI updates and background tasks in a Windows Forms application
- Handling user input and providing appropriate feedback

The event-driven paradigm proved to be a powerful approach for developing interactive applications that respond to user actions and system events. The Windows Forms framework, with its rich set of controls and event handling capabilities, provided a robust platform for building the graphical alarm clock application.

The skills and knowledge gained from this lab assignment form a solid foundation for developing more complex event-driven applications in C\# and the .NET framework.

5. References

[1] Lecture 11 Slides, CS202: Software Tools and Techniques for CSE
[2] Microsoft Documentation: C# Programming Guide,
https://learn.microsoft.com/en-us/dotnet/csharp
[3] Microsoft Documentation: Windows Forms,
https://learn.microsoft.com/en-us/dotnet/desktop/winforms
4. Wikipedia: Event-driven Programming,
https://en.wikipedia.org/wiki/Event-driven_programming