CS202
Assignment 4

Date: - 21st April 2025
Name: - Bobby Sharma
Roll No: - 24120023

Introduction, Setup and Tools:

Objective: This assignment aims to:

- Introduces the analyses of C# console games using Visual Studio. This focuses on understanding the control flow using visual studio debugger.
- Introduce to the event-driven programming in C# Window Forms Apps. using VS code.
- Understand the control flows in response to occurrence of events generated by the user under or state of application under development.

Environment Setup:

- Any operating system (Windows, MacOS, Linux, etc.)
- Python 3.10 or later
- Set up the .NET development

Tools used:

- Git Installation and make the GitHub Account,
- Code Editor (Visual Studio Code)
- Virtual environment setup in Python
- Visual Studio for .NET development

Methodology and Execution

.NET Platform

The .NET platform is the software development platform. The languages you can use for the making the project are C#, F#, Visual Basics. With the help of this platform, we can make many applications such as web, cloud, desktop, mobile, games, IoT and AI applications.

In this case we will be making the console app using the C# language and try to get the fundamentals of C# and get to know about object- oriented programming (OOPs). And we particularly get to know about the **inheritance** property.

Snake Game

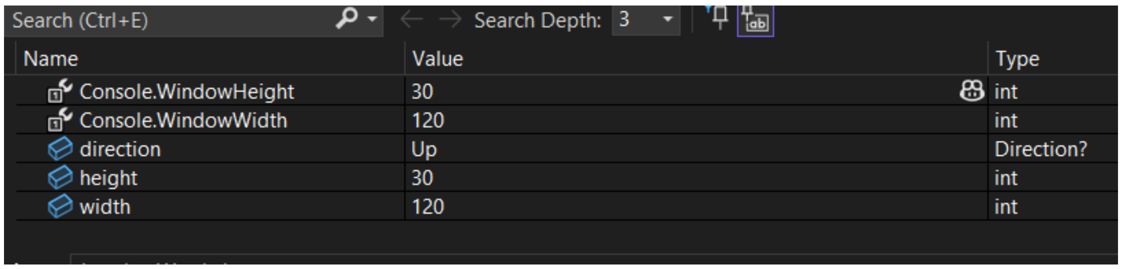
The Snake game is a console based game written in C# from the .NET console games repository. It involves moving the snake to eat food to grow, and avoiding collisions with itself or the boundaries. We will be setting the breakpoints at strategic locations to observe the game loop, input handling and food placement.

With the help of the debugging we can easily get to know about the bugs present in the piece of code. Visual Studio 2022 makes debugging very easy with the use of breakpoints and the concepts like **step-in, step-over** and **step-out**.

- **Step–In:** Step In can be executed with the F11 key. This will take us to the next line to execute. If the next line is function or method, then it will take us into the function and will start executing line by line.
- **Step-Over:** Step Over can be executed with the F10 key. This will take us to the next line to execute. If the next line is a function or method, then it will take us to the next line rather than taking us into the method. It will automatically execute the method and take us to the next line after the function call.
- **Step-Out:** Step Out can be executed with the Shift + F11 key. This will take us out of the method or the function directly rather than executing the next line of the function.

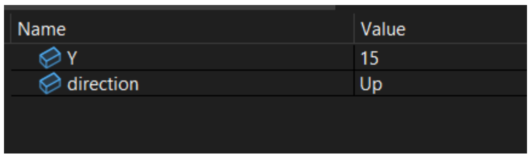
Breakpoints

- Main game loop: Inside the *while(!closeRequested)* loop, at the *switch(direction)* to inspect the movement logic. At this breakpoint we can see, the updation of the coordinates according to the arrow key, there should be the decrement from the y-axis which is shown as follows:

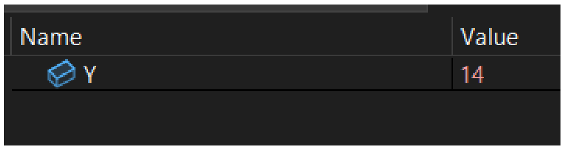


Search (Ctrl+E) Search Depth: 3

Name	Value	Type
Console.WindowHeight	30	int
Console.WindowWidth	120	int
direction	Up	Direction?
height	30	int
width	120	int

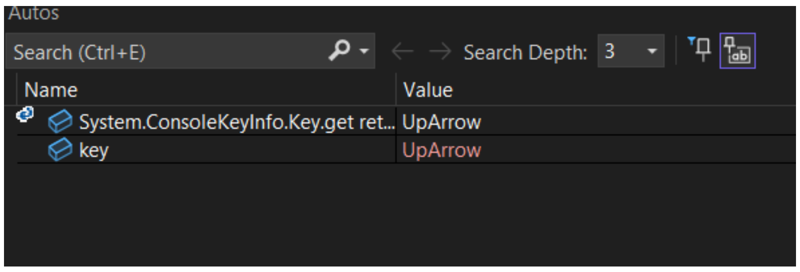


Name	Value
Y	15
direction	Up



Name	Value
Y	14

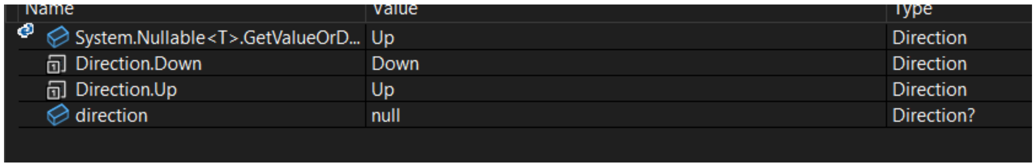
- At the line, *var key = Console.ReadKey(true).Key;* to debug input handling. At this point, we can observe the movement of the snake by throwing the arrow key, in this case, we have chosen the upArrow.



Autos

Search (Ctrl+E) Search Depth: 3

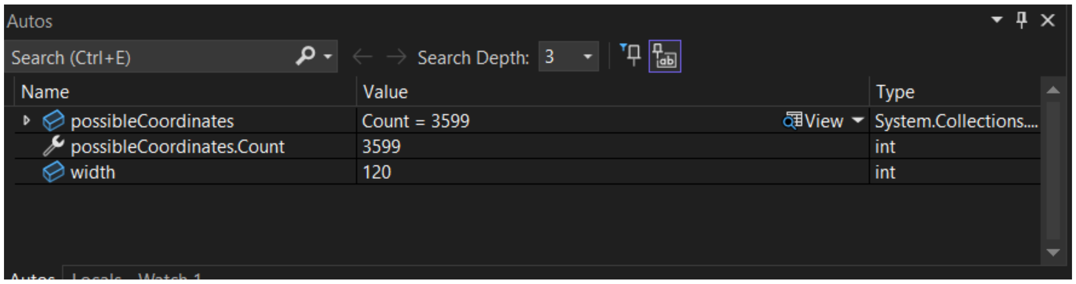
Name	Value
System.ConsoleKeyInfo.Key.get ret...	UpArrow
key	UpArrow



Name	Value	Type
System.Nullable<T>.GetValueOrD...	Up	Direction
Direction.Down	Down	Direction
Direction.Up	Up	Direction
direction	null	Direction?

We can do the inspection of the key variable after pressing the arrow (up arrow). See the direction variable updates if valid i.e. direction = up

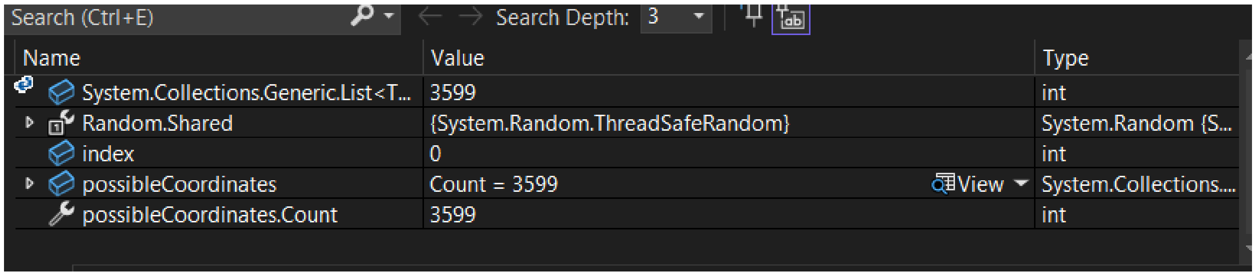
- At the line, *if (possibleCoordinates.Count == 0) return;* to check the food spawning logic.



Autos

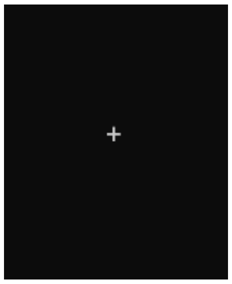
Search (Ctrl+E) Search Depth: 3

Name	Value	Type
possibleCoordinates	Count = 3599	System.Collections....
possibleCoordinates.Count	3599	int
width	120	int



Search (Ctrl+E) Search Depth: 3

Name	Value	Type
System.Collections.Generic.List<T...	3599	int
Random.Shared	{System.Random.ThreadSafeRandom}	System.Random {S...
index	0	int
possibleCoordinates	Count = 3599	System.Collections....
possibleCoordinates.Count	3599	int



```
+
```

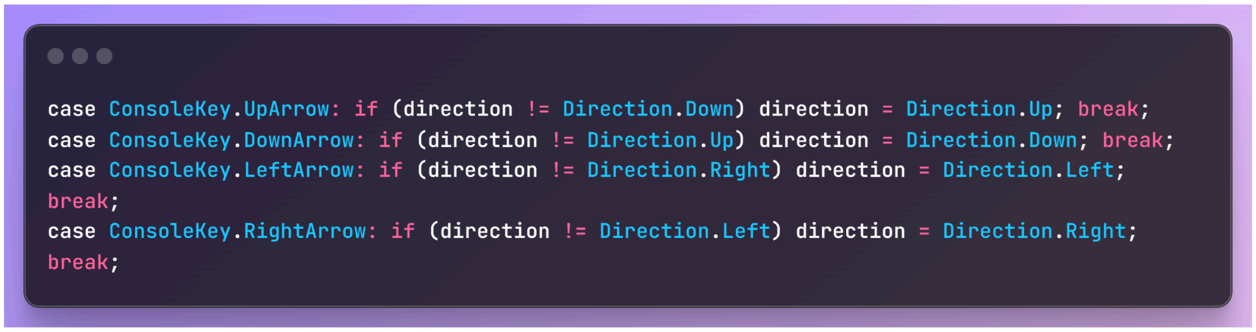
As we can see the result, the step-over just skips the stepCursorOver function and directly implements the next lines and prints the food in this case, “+”. It mainly verifies that the food spawns in an open space (map[X, Y] = Food).

Bug Fixing

Bug1: Snake can reverse into Itself

The Snake could reverse its direction and collide itself, causing an instant game over. It mainly happen when the player rapidly presses the opposite direction key (i.e. Up and down). It mainly occurs because the GetDirecction() method did not check the current direction before updating it.

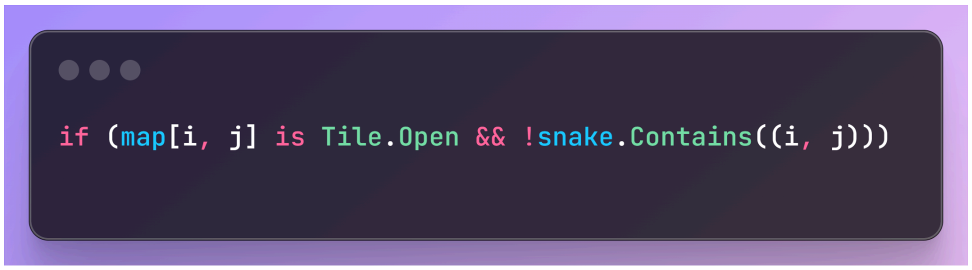
It was fixed by adding the checks. It was verified by pressing the opposite directionkeys rapidly. Now the snake only changes direction to valid non - opposite directions.



```
case ConsoleKey.UpArrow: if (direction != Direction.Down) direction = Direction.Up; break;
case ConsoleKey.DownArrow: if (direction != Direction.Up) direction = Direction.Down; break;
case ConsoleKey.LeftArrow: if (direction != Direction.Right) direction = Direction.Left;
break;
case ConsoleKey.RightArrow: if (direction != Direction.Left) direction = Direction.Right;
break;
```

Bug 2: Food Spawns on Snake

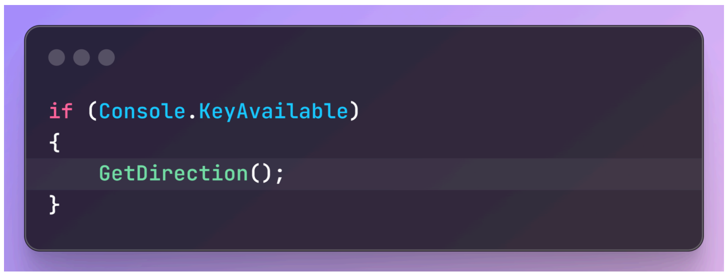
Food could spawn on a title occupied by the snake, making it unreachable or causing unexpected behaviour. It occurs when Position food selects the random coordinate. The original code doesn't check the food coordinates overlapped to the coordinates of the snake's body. This bug was fixed by checking that it the coordinates just do not contain the coordinates on the snake's body.



```
if (map[i, j] is Tile.Open && !snake.Contains((i, j)))
```

Bug 3: Blocking input handling

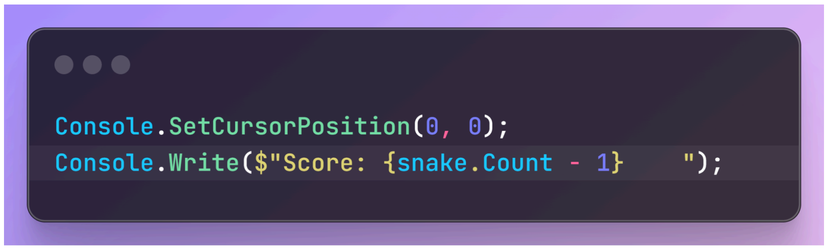
The game blocked the execution while waiting for input, causing sluggish responses to key presses. It occurs when the GetDirection call in the main loop. The previous code just reads the user input irrespective of checking if the key is available. So to fix it we just need to add the checking statement in order to fix it.



```
if (Console.KeyAvailable)
{
    GetDirection();
}
```

Bug 4: Score not displayed in the Real-Time

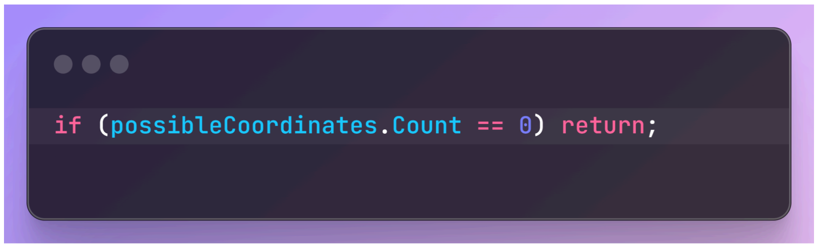
As the score was only shown at the game over, not during each achievement of getting food and growing its body. This happens only because the score display logic was only in the game - over condition. So, we can fixed that by adding the following lines:



```
Console.SetCursorPosition(0, 0);
Console.Write($"Score: {snake.Count - 1}    ");
```

Bug5: Crash when no space for food

The game automatically gets crashed when there is no space to spawn foods. It will occur at the moment when PositionFood tries to access an empty possibleCoordinates list. As the case, when the count of possibleCoordinates is zero, the original code does not return from this situation. In order to fix this if a statement is necessary in order to fix this bug.



```
if (possibleCoordinates.Count == 0) return;
```

Event Driven Programming

Event driven programming is the programming paradigm in which the flow of the program is determined by the external events. UI events from keyboards, mouse, touchpads and touchscreens. Events can be programmatically generated, such as messages from other programmes.

An **event** refers to a significant occurrence or changes in the state within a system or applications. In simple words, an event record that something has happened. Here are some common examples of events:

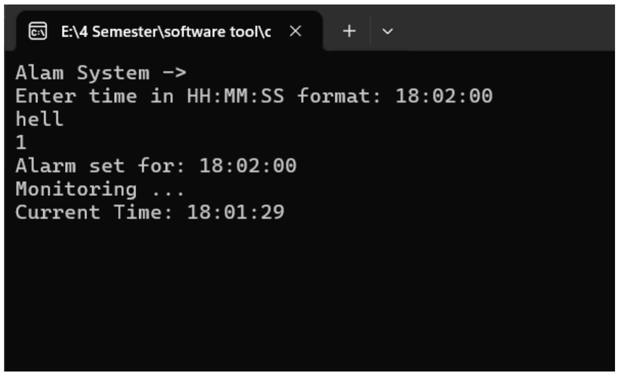
- A user click button in UI
- Someone makes an online payment
- A smartwatch detects an irregular heartbeat
- A sensor reports a vehicle’s location and speed has changed
- A record has been updated, deleted or inserted in the database.
- A warning that disc space is running low.

To handle this type of Events, we have event handler routines. These routines handle the events to which the main program will respond. For example, a single left-button mouse clicks on a command button in GUI event templates, allowing the programmers to focus on writing the event code.

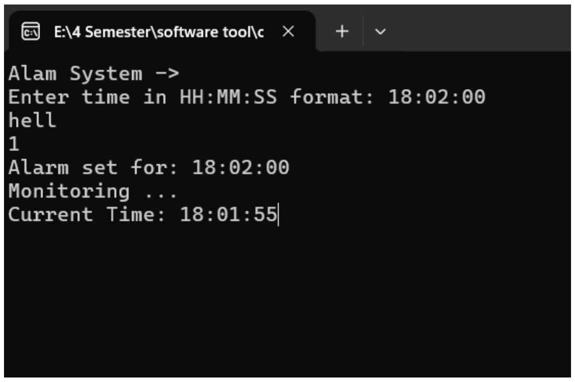
Output

The output that can be observed of the Alarm Clock as a C# console application to accept time from user in HH:MM:SS format and keep on checking it with the current time. As soon as the user supplied time and current system time become the same, raiseAlarm, a user defined event, will call the function Ring_alarm() to display “Uth ja bhai time ho gya”.

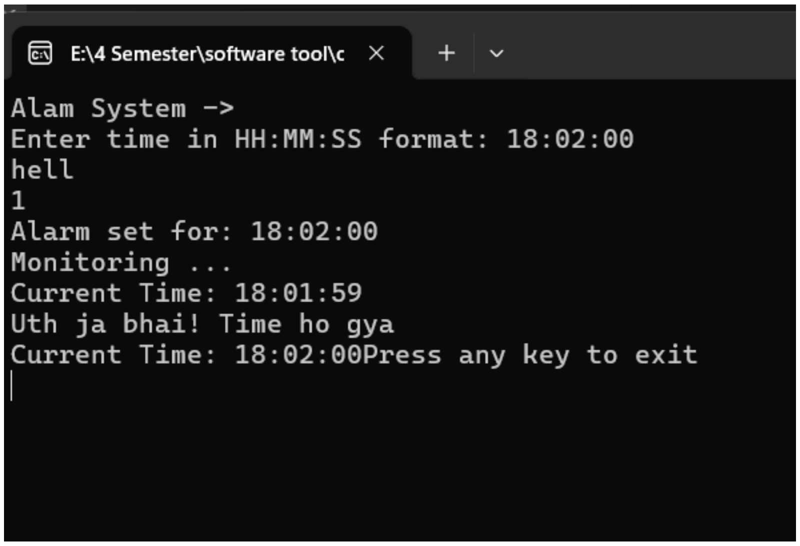
The output can be visualised by the below outputs.



```
Alam System ->
Enter time in HH:MM:SS format: 18:02:00
hell
1
Alarm set for: 18:02:00
Monitoring ...
Current Time: 18:01:29
```

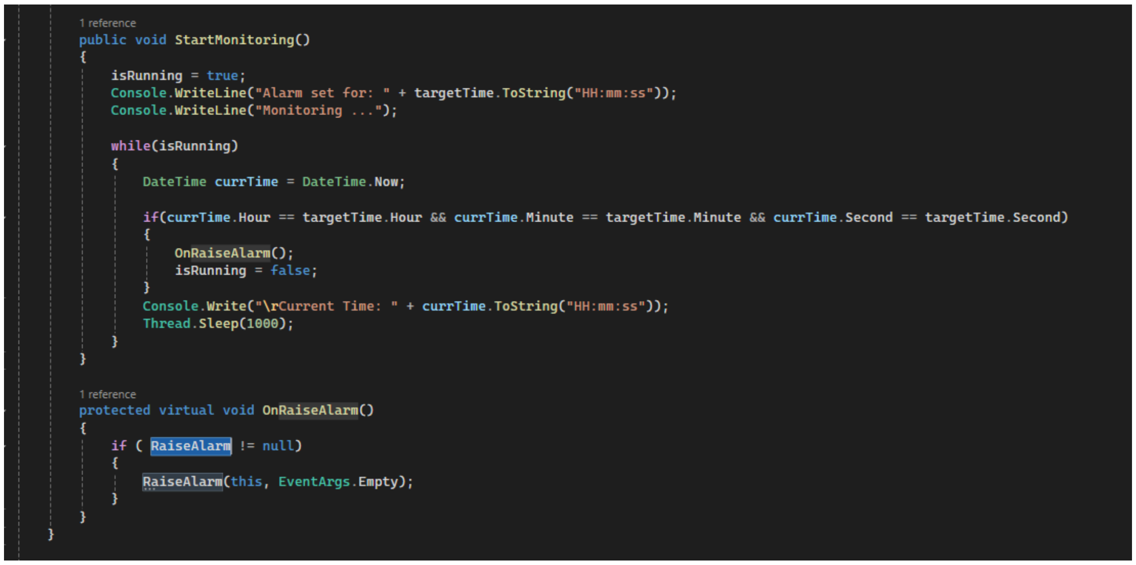


```
Alam System ->
Enter time in HH:MM:SS format: 18:02:00
hell
1
Alarm set for: 18:02:00
Monitoring ...
Current Time: 18:01:55
```



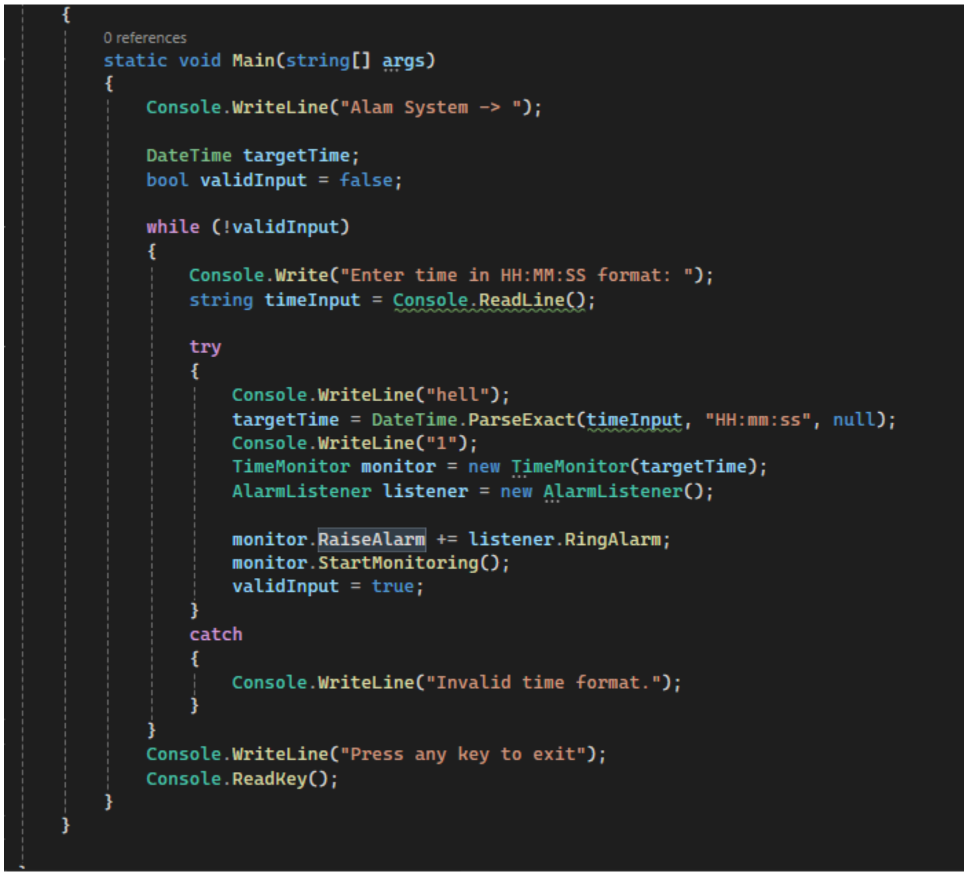
```
Alam System ->
Enter time in HH:MM:SS format: 18:02:00
hell
1
Alarm set for: 18:02:00
Monitoring ...
Current Time: 18:01:59
Uth ja bhai! Time ho gya
Current Time: 18:02:00Press any key to exit
```

The raiseAlarm event calling the Ring_Alarm() method can be seen in the below image, In which the function is called whenever the matching time condition satisfies:



```
        1 reference
        public void StartMonitoring()
        {
            isRunning = true;
            Console.WriteLine("Alarm set for: " + targetTime.ToString("HH:mm:ss"));
            Console.WriteLine("Monitoring ...");

            while(isRunning)
            {
                DateTime currTime = DateTime.Now;

                if(currTime.Hour == targetTime.Hour && currTime.Minute == targetTime.Minute && currTime.Second == targetTime.Second)
                {
                    OnRaiseAlarm();
                    isRunning = false;
                }
                Console.Write("\rCurrent Time: " + currTime.ToString("HH:mm:ss"));
                Thread.Sleep(1000);
            }
        }

        1 reference
        protected virtual void OnRaiseAlarm()
        {
            if ( RaiseAlarm != null)
            {
                RaiseAlarm(this, EventArgs.Empty);
            }
        }
    }
```

The usage of the Ring_Alarm() method is in the main function which is calling the StartMonitoring() function and always checking or comparing the current time to the user time for raising the alarm.

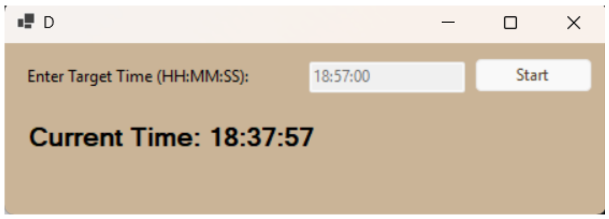


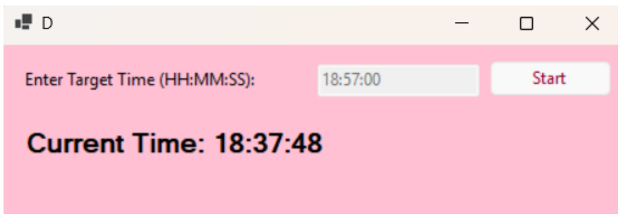
```
    {
        0 references
        static void Main(string[] args)
        {
            Console.WriteLine("Alam System -> ");

            DateTime targetTime;
            bool validInput = false;

            while (!validInput)
            {
                Console.Write("Enter time in HH:MM:SS format: ");
                string timeInput = Console.ReadLine();

                try
                {
                    Console.WriteLine("hell");
                    targetTime = DateTime.ParseExact(timeInput, "HH:mm:ss", null);
                    Console.WriteLine("1");
                    TimeMonitor monitor = new TimeMonitor(targetTime);
                    AlarmListener listener = new AlarmListener();

                    monitor.RaiseAlarm += listener.RingAlarm;
                    monitor.StartMonitoring();
                    validInput = true;
                }
                catch
                {
                    Console.WriteLine("Invalid time format.");
                }
            }
            Console.WriteLine("Press any key to exit");
            Console.ReadKey();
        }
    }
```

The task is now to modify the same C# console application to perform input/output activities only through **Windows Form.** Now, will be the time that the user will enter the time in the text box and the user will start from the start button.

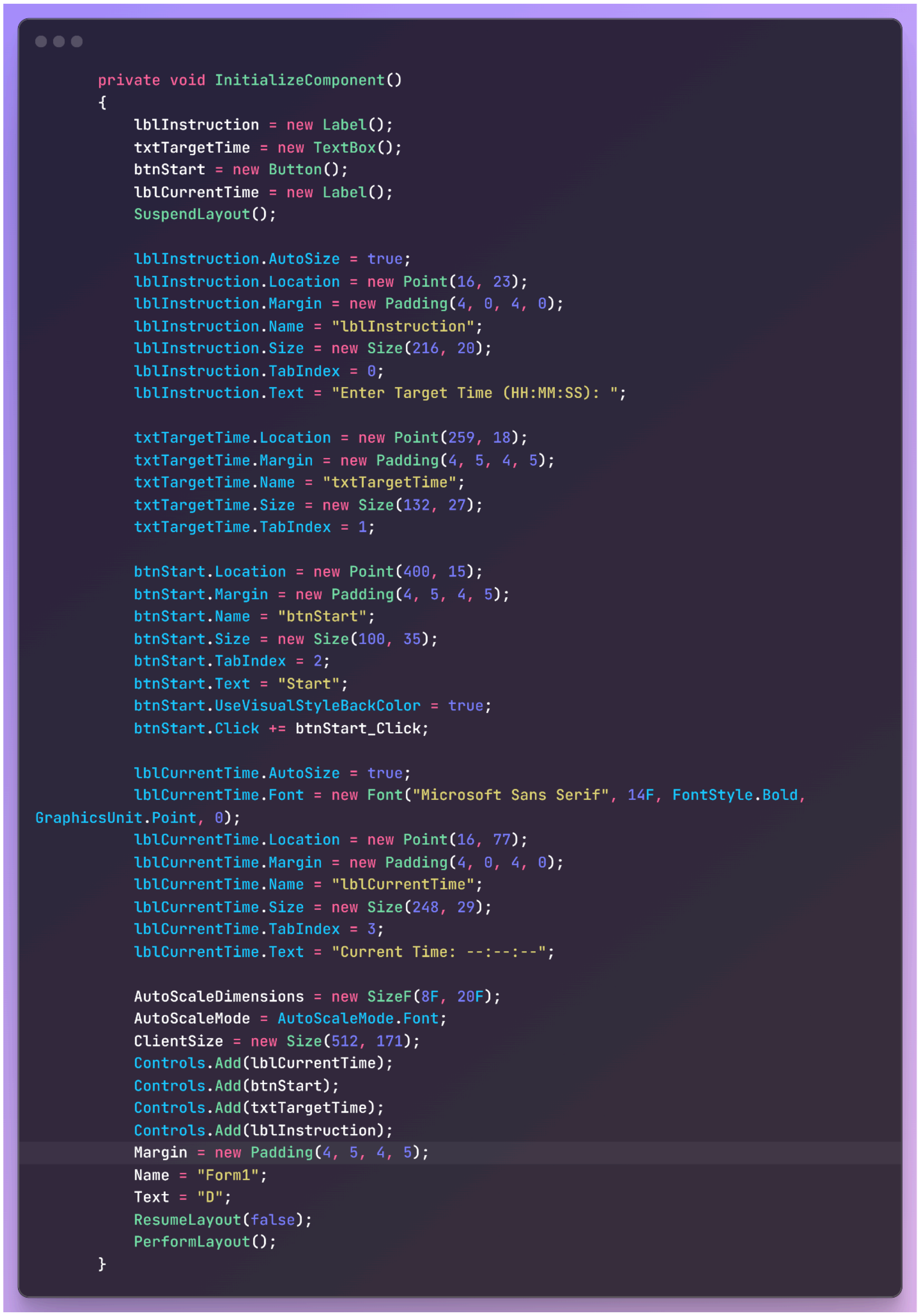
During which we are going to change the color of the background of the form. On reaching the set time. It will stop the changing color and an alert box type shows up which will tell us the time has been reached.

The UI for the button, the message that shows the (asking user to set the time in the HH:MM:SS format) and current time with the variation of the background color:



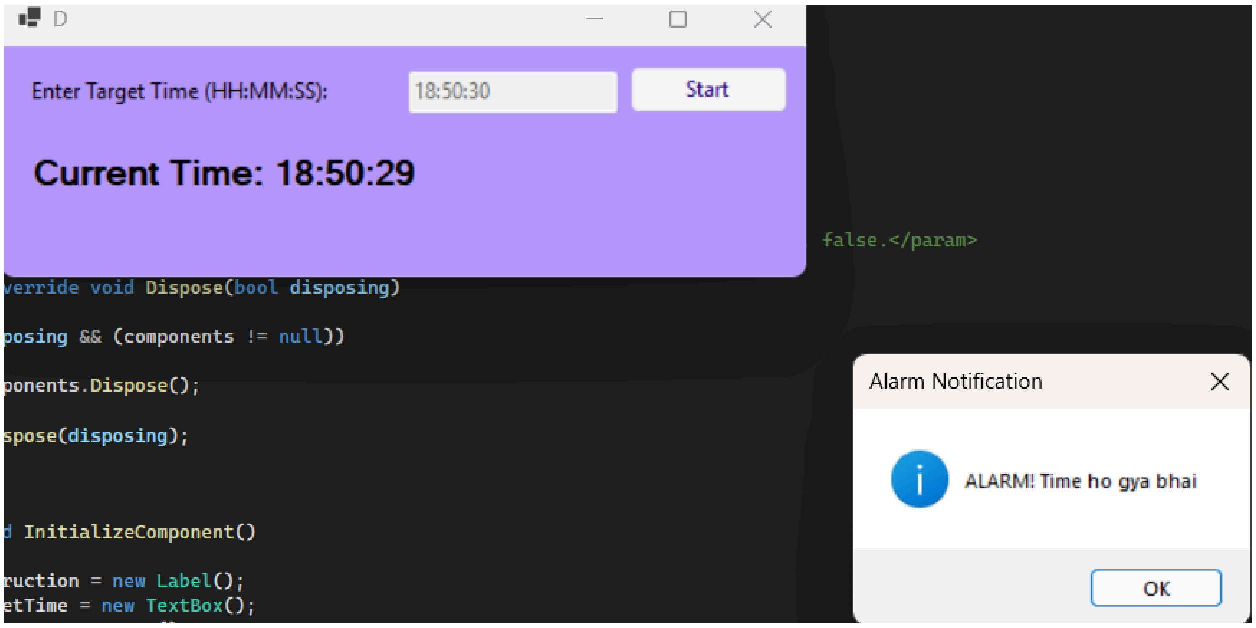


The code for the UI and their respective components that lead us to this output are as follows:



```
        private void InitializeComponent()
        {
            lblInstruction = new Label();
            txtTargetTime = new TextBox();
            btnStart = new Button();
            lblCurrentTime = new Label();
            SuspendLayout();

            lblInstruction.AutoSize = true;
            lblInstruction.Location = new Point(16, 23);
            lblInstruction.Margin = new Padding(4, 0, 4, 0);
            lblInstruction.Name = "lblInstruction";
            lblInstruction.Size = new Size(216, 20);
            lblInstruction.TabIndex = 0;
            lblInstruction.Text = "Enter Target Time (HH:MM:SS): ";

            txtTargetTime.Location = new Point(259, 18);
            txtTargetTime.Margin = new Padding(4, 5, 4, 5);
            txtTargetTime.Name = "txtTargetTime";
            txtTargetTime.Size = new Size(132, 27);
            txtTargetTime.TabIndex = 1;

            btnStart.Location = new Point(400, 15);
            btnStart.Margin = new Padding(4, 5, 4, 5);
            btnStart.Name = "btnStart";
            btnStart.Size = new Size(100, 35);
            btnStart.TabIndex = 2;
            btnStart.Text = "Start";
            btnStart.UseVisualStyleBackColor = true;
            btnStart.Click += btnStart_Click;

            lblCurrentTime.AutoSize = true;
            lblCurrentTime.Font = new Font("Microsoft Sans Serif", 14F, FontStyle.Bold,
GraphicsUnit.Point, 0);
            lblCurrentTime.Location = new Point(16, 77);
            lblCurrentTime.Margin = new Padding(4, 0, 4, 0);
            lblCurrentTime.Name = "lblCurrentTime";
            lblCurrentTime.Size = new Size(248, 29);
            lblCurrentTime.TabIndex = 3;
            lblCurrentTime.Text = "Current Time: --:--:--";

            AutoScaleDimensions = new SizeF(8F, 20F);
            AutoScaleMode = AutoScaleMode.Font;
            ClientSize = new Size(512, 171);
            Controls.Add(lblCurrentTime);
            Controls.Add(btnStart);
            Controls.Add(txtTargetTime);
            Controls.Add(lblInstruction);
            Margin = new Padding(4, 5, 4, 5);
            Name = "Form1";
            Text = "D";
            ResumeLayout(false);
            PerformLayout();
        }
```

After the initialization of all the components in the UI, the main logical code will be working in the Form1.cs file whereas the UI designer code will be in the Form1.Designer.cs file. The full running of this GUI and the implementation of the event as well as logic will going to lead us to the following output:



Current Time: 18:50:29

ALARM! Time ho gya bhai

```
                                                  false.</param>
verride void Dispose(bool disposing)

posing && (components != null))

ponents.Dispose();

spose(disposing);


d InitializeComponent()

ruction = new Label();
etTime = new TextBox();
```

Challenges

- The debugging of the console game was quite difficult. The appropriate use of the debugger and breaking points help in getting to the result.
- During the formation of the windows application of the alarm clock. The logic at the beginning might be tricky like when and where we have to define the classes..

Lesson Learned

We get to know:

- We get to know the control flow of the c# console games and the use of the visual studio debugger.
- We learned the applications of concepts step-in, step-over and step-out. And the usage of the breakpoints at appropriate places.
- We develop and analyze the windows forms apps using c#.
- Hands-on experience with the application due to user interaction or triggered by a certain state of the application during runtime.

Summary

In this assignment, we get to know about hands-on experience in the analysis of console gaming as well as debugging them with the help of the tools provided by Visual Studio. We have made the alarm clock on the console as well as the windows application. This led us to know more about the .NET environment as well as the Windows applications.