

# Lab Assignment 11

## Lab Topic: Analyzing C# Console Games for Bugs

### 1. Introduction

This lab focuses on practical debugging skills within the Visual Studio environment using C# console game examples from the dotnet-console-games collection. The core activity involves utilizing the Visual Studio Debugger to analyze program execution flow, identify existing or injected runtime bugs, and implement code corrections, thereby enhancing understanding of fault diagnosis and code verification.

#### Environment Setup:

- Operating System: Windows
- Software: Visual Studio 2022 (Community Edition), Visual Studio with .NET SDK
- Programming Language: C# 13 or dotnet SDK 9.0.102

#### Key Concepts:

**Runtime Error:** An error that occurs only when the program is executing, often due to specific data, user input, or environmental conditions (e.g., NullReferenceException, DivideByZeroException). Contrasts with compile-time errors.

**Logic Error:** A flaw where the program compiles and runs without crashing but produces incorrect or unexpected results due to faulty reasoning or implementation in the code.

**Mutation (Bug Injection):** Intentionally introducing small changes (e.g., altering operators, constants) into working code to create bugs for the purpose of testing detection or practicing debugging.

**Top-Level Statements:** A C# feature (used in these projects) allowing executable code directly in Program.cs without the traditional Main method boilerplate, simplifying simple console applications.

---

### 2. Methodology and Execution

This is the file structure in the folder Lab9.

#### File Structure:

Lab11/

```
└── dotnet-console-games.sln # VS Solution file (groups all games)
└── dotnet-console-games.slnf# VS Solution filter file (optional view settings)
└── Directory.Build.props # Common build properties (repo-level)
└── Projects/      # Folder containing individual game projects
    └── Rock Paper Scissors/ # Project folder for Rock Paper Scissors
        └── Program.cs      # Game logic and entry point
```

```

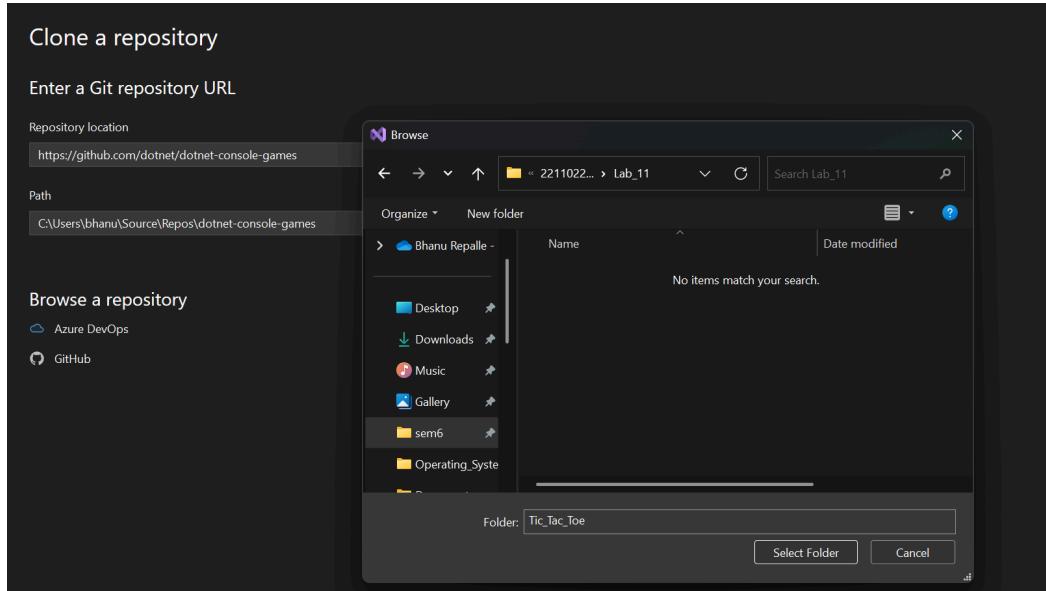
    └── README.md      # Specific game info (optional)
    └── Rock Paper Scissors.csproj # Project build settings
    └── Tic Tac Toe/   # Project folder for Tic Tac Toe
        ├── Program.cs    # Game logic and entry point
        └── README.md      # Specific game info (optional)
        └── Tic Tac Toe.csproj # Project build settings

```

Below is the step-by-step procedure I followed.

## Step 1: Initial Setup

First, I cloned the projects folder.



I chose the 2 games: Tic Tac Toe and Rock Paper Scissors. I then opened the folder with visual studio.

I started by inserting breakpoints at a few critical points and played the game.



## Step 2: Analysing Tic Tac Toe

The input collection is very strict so it was not possible to give any invalid input. I played multiple rounds.

Below are a few screenshots of the output at break points, they are as expected.

The screenshot shows a code editor interface with the following details:

- Code Snippet:**

```
35 }  
36 playerTurn = !playerTurn; // 1ms elapsed  
37 if (CheckForFullBoard())  
38 {  
39     EndGame(" Draw.");  
40     break;  
41 }  
42 }  
43 if (!closeRequested)  
44 {
```
- Status Bar:** Shows "100 %", a search icon, and a green checkmark indicating "No issues found".
- Search Results Panel:** Titled "Autos", it includes:
  - A search bar with placeholder "Search (Ctrl+E)" and a magnifying glass icon.
  - A dropdown menu with icons for "File", "Edit", "Search", "Find", "Replace", and "Copy".
  - A "Search Depth" dropdown set to 3.
  - A "Type" dropdown with a blue square icon.
- Table:** A table showing variable information:

Name	Type
playerTurn	bool

There was one issue which occurred which is not exactly the game logic but it was because of the interaction between console and the debugger and how screen clearing/cursor positioning works.

## The cursor is out of the board

The screenshot shows a code editor with a C# file open. The code defines a 3x3 board variable and an EndGame method. A tooltip for the board variable shows a 3x3 grid of cells. To the right, a game window titled "Tic Tac Toe" displays a 3x3 board with the top-left cell containing an 'X' and the bottom-left cell containing a '0'.

```
134
135
136
137
138
139
140
141
142
143
144 void EndGame(string message)
145 {
146     Console.Clear();
147     RenderBoard();
148     Console.WriteLine();
149     Console.Write(message);
150 }
```

No issues found

Search (Ctrl+E) 🔎 Search Depth: 3 ⌂ Value

System.Runtime.CompilerServices System.Tic Tac Toe

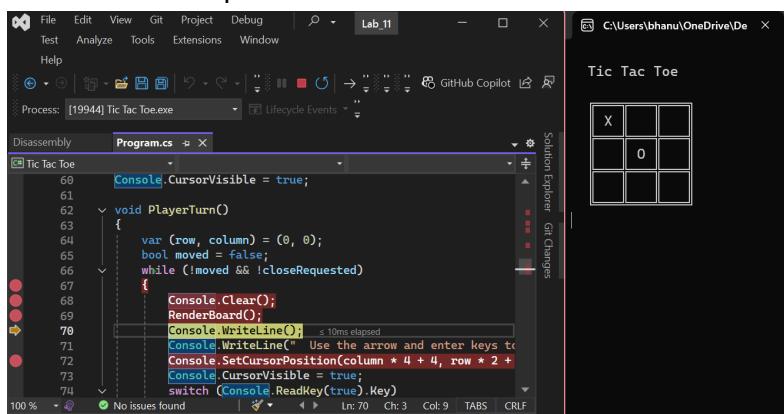
This is observed at the RenderBoard() breakpoint

## Debugging:

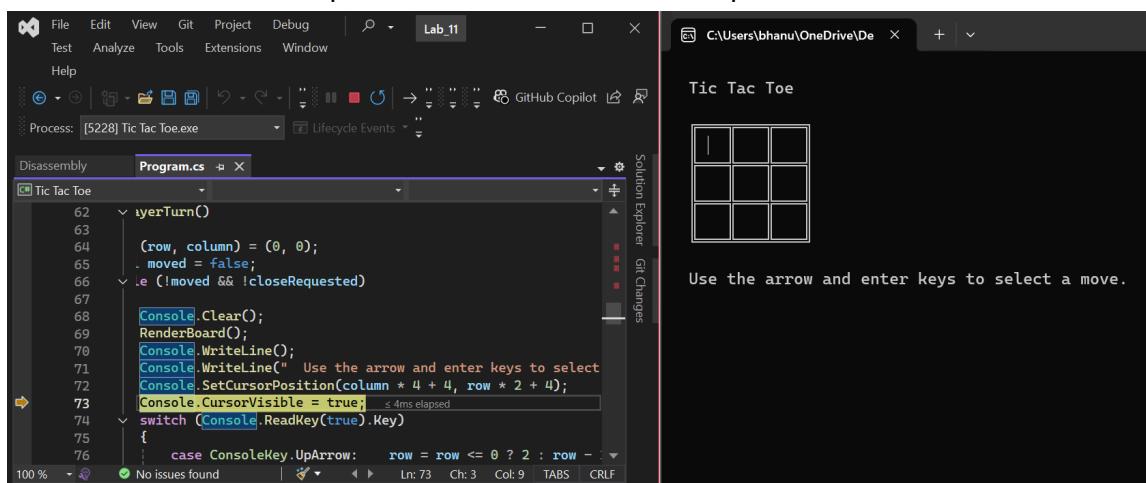
I set multiple bugs along the lines.

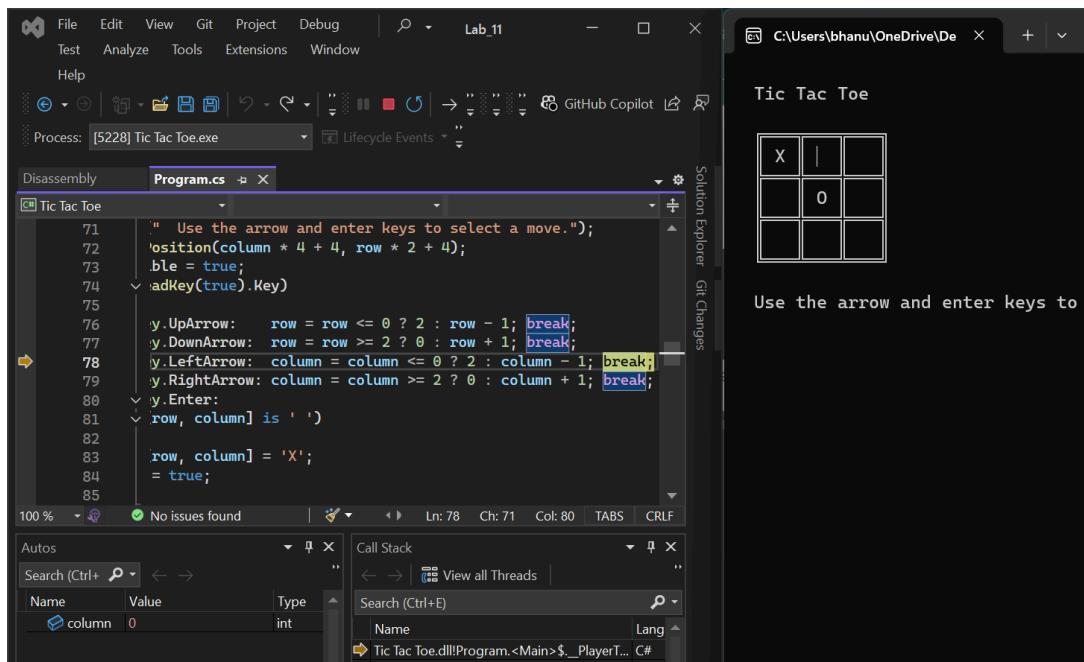
```
    65     bootMoved = false;
    66     while (!moved && !closeRequested)
    67     {
    68         Console.Clear();
    69         RenderBoard();
    70         Console.WriteLine();      ↳ 7ms elapsed
    71         Console.WriteLine(" Use the arrow and enter keys to sel");
    72         Console.SetCursorPosition(column * 4 + 4, row * 2 + 4);
    73         Console.CursorVisible = true;
    74         switch (Console.ReadKey(true).Key)
```

This is the exact position when the cursor is seen outside the board.



I believe that the error because of the selection of the break points and using step over So removed all the breakpoints in the area and used step in then the arrow location is as expected.





Other than that, I could not find any logical bugs so I inserted 5 major bugs into program.cs code.

#### Tic Tac Toe:

1. Changed && to || in PlayerTurn()

<pre><code>void PlayerTurn() {     var (row, column) = (0, 0);     bool moved = false;     while (!moved &amp;&amp; !closeRequested)     {</code></pre>	<pre><code>void PlayerTurn() {     var (row, column) = (0, 0);     bool moved = false;     while (!moved    !closeRequested)     {</code></pre>
---	---

2. Change == to != in ComputeTurn()

<pre><code>void ComputerTurn() {     var possibleMoves = new List&lt;(int X, int Y)&gt;;     for (int i = 0; i &lt; 3; i++)     {         for (int j = 0; j &lt; 3; j++)         {             if (board[i, j] == ' ')             {</code></pre>	<pre><code>void ComputerTurn() {     var possibleMoves = new List&lt;(int X, int Y)&gt;;     for (int i = 0; i &lt; 3; i++)     {         for (int j = 0; j &lt; 3; j++)         {             if (board[i, j] != ' ')             {</code></pre>
---	---

3. Changes winning logic by switching 2 with 3

<pre><code>112 113     bool CheckForThree(char c) =&gt; 114         board[0, 0] == c &amp;&amp; board[1, 0] == c &amp;&amp; board[2, 0] == c    115         board[0, 1] == c &amp;&amp; board[1, 1] == c &amp;&amp; board[2, 1] == c    116         board[0, 2] == c &amp;&amp; board[1, 2] == c &amp;&amp; board[2, 2] == c    117         board[0, 0] == c &amp;&amp; board[0, 1] == c &amp;&amp; board[0, 2] == c    118         board[1, 0] == c &amp;&amp; board[1, 1] == c &amp;&amp; board[1, 2] == c   </code></pre>
--

```

bool CheckForThree(char c) =>
    board[0, 0] == c && board[1, 0] == c && board[2, 0] == c || 
    board[0, 1] == c && board[1, 1] == c && board[2, 1] == c || 
    board[0, 2] == c && board[1, 2] == c && board[2, 2] == c || 
    board[0, 0] == c && board[0, 1] == c && board[0, 2] == c ||

```

4. Changed CloseRequested to !CloseRequested

```

};  
while (closeRequested)  
{  
};  
while (!closeRequested)  
{  


```

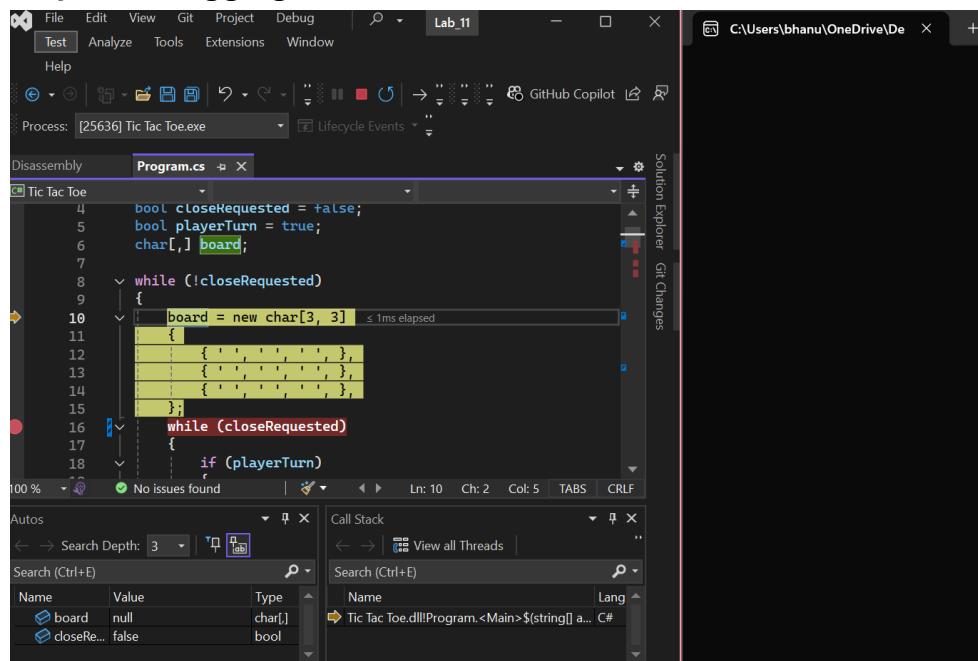
5. Checked for 'X' twice and did not check 'O'

```

PlayerTurn();  
if (CheckForThree('X'))  
{  
    EndGame(" You Win.");  
    break;  
}  
else  
{  
    ComputerTurn();  
    if (CheckForThree('O'))  
    {  
        if (CheckForThree('X'))  


```

### Step 3: Debugging Tic Tac Toe



The screenshot shows the Visual Studio interface with the title bar "Lab\_11". The main window displays the "Disassembly" view of the file "Program.cs". The code is as follows:

```
39     EndGame(" Draw.");
40     break;
41 }
42 if (!closeRequested)
43 {
44     Console.WriteLine();
45     Console.WriteLine(" Play Again [enter], or quit [es");
46     GetInput();
47     Console.CursorVisible = false;
48     switch (Console.ReadKey(true).Key)
49     {
50         case ConsoleKey.Enter: break;
51         case ConsoleKey.Escape:
52             closeRequested = true;
53     }
}
```

The line "Console.WriteLine();" at line 45 is highlighted. The status bar at the bottom indicates "No issues found".

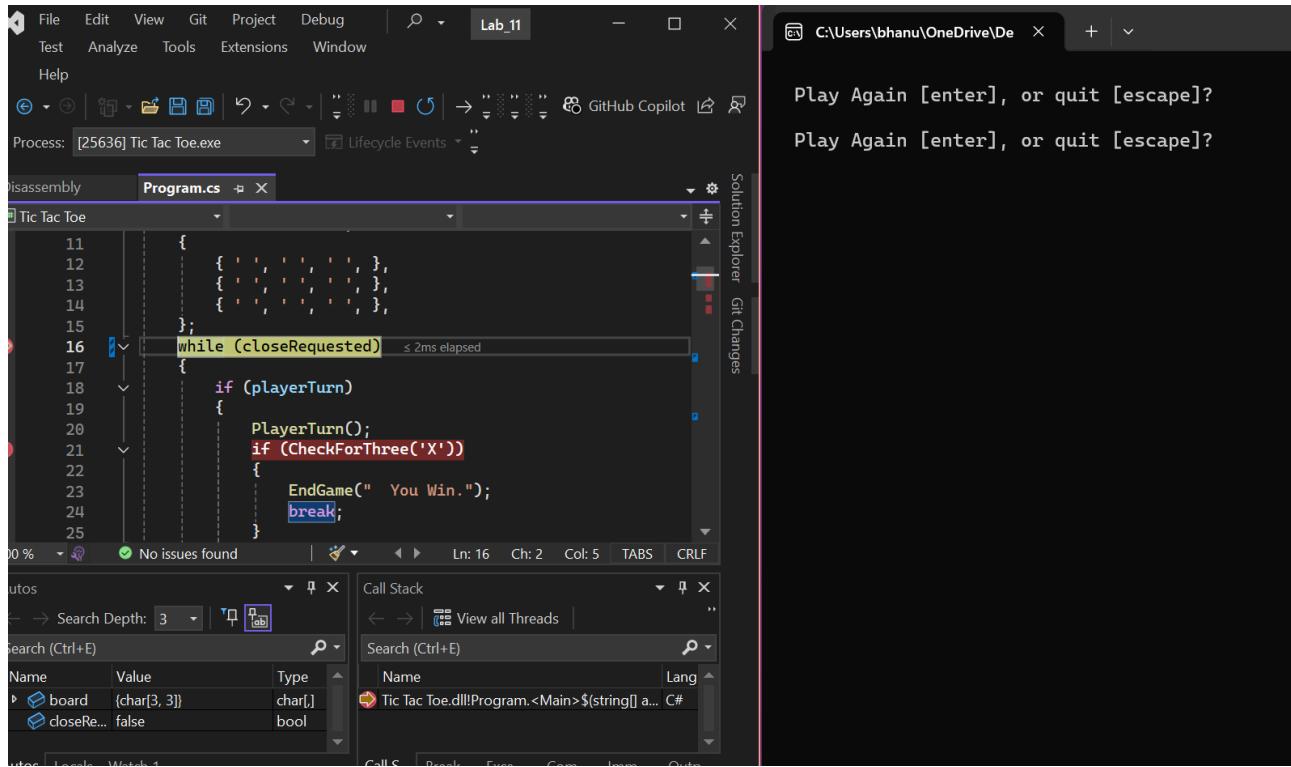
The screenshot shows the Visual Studio interface with the title bar "Lab\_11". The main window displays the "Disassembly" view of the file "Program.cs". The code is identical to the previous screenshot.

A breakpoint is set on the line "case ConsoleKey.Enter: break;". When the program reaches this point, the status bar at the bottom shows "Break...". The output window on the right displays the message "Play Again [enter], or quit [escape]?".

## Bug 1:

After realising that I am going in a loop and reading the console output, it seems that the game is thinking that it has ended and is asking the player for another round. Without the player even playing 1 round. And It is going in a loop.

So I checked and loop and noticed that at this point, we need the game to enter this while loop but since it is set to closeRequested and not !closRequested, we are unable to enter, So our 1st bug is found and corrected.



The screenshot shows the Visual Studio IDE with the project "Lab\_11" open. The code editor displays "Program.cs" with the following code:

```
11  {
12    {
13      {
14        {
15    };
16    while (closeRequested) < 2ms elapsed
17    {
18      if (playerTurn)
19      {
20        PlayerTurn();
21        if (CheckForThree('X'))
22        {
23          EndGame(" You Win.");
24        }
25      }

```

The line `while (closeRequested)` is highlighted with a red arrow indicating a break point. The status bar at the bottom shows "No issues found". To the right of the editor, two terminal windows show the game loop:

```
Play Again [enter], or quit [escape]?
Play Again [enter], or quit [escape]?
```

After correcting, it entered the loop as expected.

<img alt="Screenshot of Microsoft Visual Studio showing the Disassembly and Program.cs tabs for a Tic Tac Toe application. The code is in C#. The disassembly tab shows assembly instructions corresponding to the C# code. The Program.cs tab shows the following code: 13-16: Board state assignment. 17: while (!closeRequested) { 18-27: if (playerTurn) { PlayerTurn(); if (CheckForThree('X')) { EndGame(" You Win."); break; } else { 28-30: // Commented out by a cursor } } } 31-32: } 33-34: } 35-36: } 37-38: } 39-40: } 41-42: } 43-44: } 45-46: } 47-48: } 49-50: } 51-52: } 53-54: } 55-56: } 57-58: } 59-60: } 61-62: } 63-64: } 65-66: } 67-68: } 69-70: } 71-72: } 73-74: } 75-76: } 77-78: } 79-80: } 81-82: } 83-84: } 85-86: } 87-88: } 89-90: } 91-92: } 93-94: } 95-96: } 97-98: } 99-100: } 101-102: } 103-104: } 105-106: } 107-108: } 109-110: } 111-112: } 113-114: } 115-116: } 117-118: } 119-120: } 121-122: } 123-124: } 125-126: } 127-128: } 129-130: } 131-132: } 133-134: } 135-136: } 137-138: } 139-140: } 141-142: } 143-144: } 145-146: } 147-148: } 149-150: } 151-152: } 153-154: } 155-156: } 157-158: } 159-160: } 161-162: } 163-164: } 165-166: } 167-168: } 169-170: } 171-172: } 173-174: } 175-176: } 177-178: } 179-180: } 181-182: } 183-184: } 185-186: } 187-188: } 189-190: } 191-192: } 193-194: } 195-196: } 197-198: } 199-200: } 201-202: } 203-204: } 205-206: } 207-208: } 209-210: } 211-212: } 213-214: } 215-216: } 217-218: } 219-220: } 221-222: } 223-224: } 225-226: } 227-228: } 229-230: } 231-232: } 233-234: } 235-236: } 237-238: } 239-240: } 241-242: } 243-244: } 245-246: } 247-248: } 249-250: } 251-252: } 253-254: } 255-256: } 257-258: } 259-260: } 261-262: } 263-264: } 265-266: } 267-268: } 269-270: } 271-272: } 273-274: } 275-276: } 277-278: } 279-280: } 281-282: } 283-284: } 285-286: } 287-288: } 289-290: } 291-292: } 293-294: } 295-296: } 297-298: } 299-300: } 301-302: } 303-304: } 305-306: } 307-308: } 309-310: } 311-312: } 313-314: } 315-316: } 317-318: } 319-320: } 321-322: } 323-324: } 325-326: } 327-328: } 329-330: } 331-332: } 333-334: } 335-336: } 337-338: } 339-340: } 341-342: } 343-344: } 345-346: } 347-348: } 349-350: } 351-352: } 353-354: } 355-356: } 357-358: } 359-360: } 361-362: } 363-364: } 365-366: } 367-368: } 369-370: } 371-372: } 373-374: } 375-376: } 377-378: } 379-380: } 381-382: } 383-384: } 385-386: } 387-388: } 389-390: } 391-392: } 393-394: } 395-396: } 397-398: } 399-400: } 401-402: } 403-404: } 405-406: } 407-408: } 409-410: } 411-412: } 413-414: } 415-416: } 417-418: } 419-420: } 421-422: } 423-424: } 425-426: } 427-428: } 429-429: } 430-431: } 432-433: } 434-435: } 436-437: } 438-439: } 440-441: } 442-443: } 444-445: } 446-447: } 448-449: } 450-451: } 452-453: } 454-455: } 456-457: } 458-459: } 460-461: } 462-463: } 464-465: } 466-467: } 468-469: } 470-471: } 472-473: } 474-475: } 476-477: } 478-479: } 480-481: } 482-483: } 484-485: } 486-487: } 488-489: } 490-491: } 492-493: } 494-495: } 496-497: } 498-499: } 499-500: } 500-501: } 501-502: } 502-503: } 503-504: } 504-505: } 505-506: } 506-507: } 507-508: } 508-509: } 509-510: } 510-511: } 511-512: } 512-513: } 513-514: } 514-515: } 515-516: } 516-517: } 517-518: } 518-519: } 519-520: } 520-521: } 521-522: } 522-523: } 523-524: } 524-525: } 525-526: } 526-527: } 527-528: } 528-529: } 529-530: } 530-531: } 531-532: } 532-533: } 533-534: } 534-535: } 535-536: } 536-537: } 537-538: } 538-539: } 539-540: } 540-541: } 541-542: } 542-543: } 543-544: } 544-545: } 545-546: } 546-547: } 547-548: } 548-549: } 549-550: } 550-551: } 551-552: } 552-553: } 553-554: } 554-555: } 555-556: } 556-557: } 557-558: } 558-559: } 559-560: } 560-561: } 561-562: } 562-563: } 563-564: } 564-565: } 565-566: } 566-567: } 567-568: } 568-569: } 569-570: } 570-571: } 571-572: } 572-573: } 573-574: } 574-575: } 575-576: } 576-577: } 577-578: } 578-579: } 579-580: } 580-581: } 581-582: } 582-583: } 583-584: } 584-585: } 585-586: } 586-587: } 587-588: } 588-589: } 589-590: } 590-591: } 591-592: } 592-593: } 593-594: } 594-595: } 595-596: } 596-597: } 597-598: } 598-599: } 599-599: } 599-600: } 600-601: } 601-602: } 602-603: } 603-604: } 604-605: } 605-606: } 606-607: } 607-608: } 608-609: } 609-610: } 610-611: } 611-612: } 612-613: } 613-614: } 614-615: } 615-616: } 616-617: } 617-618: } 618-619: } 619-620: } 620-621: } 621-622: } 622-623: } 623-624: } 624-625: } 625-626: } 626-627: } 627-628: } 628-629: } 629-630: } 630-631: } 631-632: } 632-633: } 633-634: } 634-635: } 635-636: } 636-637: } 637-638: } 638-639: } 639-640: } 640-641: } 641-642: } 642-643: } 643-644: } 644-645: } 645-646: } 646-647: } 647-648: } 648-649: } 649-650: } 650-651: } 651-652: } 652-653: } 653-654: } 654-655: } 655-656: } 656-657: } 657-658: } 658-659: } 659-660: } 660-661: } 661-662: } 662-663: } 663-664: } 664-665: } 665-666: } 666-667: } 667-668: } 668-669: } 669-670: } 670-671: } 671-672: } 672-673: } 673-674: } 674-675: } 675-676: } 676-677: } 677-678: } 678-679: } 679-680: } 680-681: } 681-682: } 682-683: } 683-684: } 684-685: } 685-686: } 686-687: } 687-688: } 688-689: } 689-690: } 690-691: } 691-692: } 692-693: } 693-694: } 694-695: } 695-696: } 696-697: } 697-698: } 698-699: } 699-699: } 699-700: } 700-701: } 701-702: } 702-703: } 703-704: } 704-705: } 705-706: } 706-707: } 707-708: } 708-709: } 709-710: } 710-711: } 711-712: } 712-713: } 713-714: } 714-715: } 715-716: } 716-717: } 717-718: } 718-719: } 719-720: } 720-721: } 721-722: } 722-723: } 723-724: } 724-725: } 725-726: } 726-727: } 727-728: } 728-729: } 729-729: } 729-730: } 730-731: } 731-732: } 732-733: } 733-734: } 734-735: } 735-736: } 736-737: } 737-738: } 738-739: } 739-740: } 740-741: } 741-742: } 742-743: } 743-744: } 744-745: } 745-746: } 746-747: } 747-748: } 748-749: } 749-749: } 749-750: } 750-751: } 751-752: } 752-753: } 753-754: } 754-755: } 755-756: } 756-757: } 757-758: } 758-759: } 759-759: } 759-760: } 760-761: } 761-762: } 762-763: } 763-764: } 764-765: } 765-766: } 766-767: } 767-768: } 768-769: } 769-769: } 769-770: } 770-771: } 771-772: } 772-773: } 773-774: } 774-775: } 775-776: } 776-777: } 777-778: } 778-779: } 779-779: } 779-780: } 780-781: } 781-782: } 782-783: } 783-784: } 784-785: } 785-786: } 786-787: } 787-788: } 788-789: } 789-789: } 789-790: } 790-791: } 791-792: } 792-793: } 793-794: } 794-795: } 795-796: } 796-797: } 797-798: } 798-799: } 799-799: } 799-800: } 800-801: } 801-802: } 802-803: } 803-804: } 804-805: } 805-806: } 806-807: } 807-808: } 808-809: } 809-809: } 809-810: } 810-811: } 811-812: } 812-813: } 813-814: } 814-815: } 815-816: } 816-817: } 817-818: } 818-819: } 819-819: } 819-820: } 820-821: } 821-822: } 822-823: } 823-824: } 824-825: } 825-826: } 826-827: } 827-828: } 828-829: } 829-829: } 829-830: } 830-831: } 831-832: } 832-833: } 833-834: } 834-835: } 835-836: } 836-837: } 837-838: } 838-839: } 839-839: } 839-840: } 840-841: } 841-842: } 842-843: } 843-844: } 844-845: } 845-846: } 846-847: } 847-848: } 848-849: } 849-849: } 849-850: } 850-851: } 851-852: } 852-853: } 853-854: } 854-855: } 855-856: } 856-857: } 857-858: } 858-859: } 859-859: } 859-860: } 860-861: } 861-862: } 862-863: } 863-864: } 864-865: } 865-866: } 866-867: } 867-868: } 868-869: } 869-869: } 869-870: } 870-871: } 871-872: } 872-873: } 873-874: } 874-875: } 875-876: } 876-877: } 877-878: } 878-879: } 879-879: } 879-880: } 880-881: } 881-882: } 882-883: } 883-884: } 884-885: } 885-886: } 886-887: } 887-888: } 888-889: } 889-889: } 889-890: } 890-891: } 891-892: } 892-893: } 893-894: } 894-895: } 895-896: } 896-897: } 897-898: } 898-899: } 899-899: } 899-900: } 900-901: } 901-902: } 902-903: } 903-904: } 904-905: } 905-906: } 906-907: } 907-908: } 908-909: } 909-909: } 909-910: } 910-911: } 911-912: } 912-913: } 913-914: } 914-915: } 915-916: } 916-917: } 917-918: } 918-919: } 919-919: } 919-920: } 920-921: } 921-922: } 922-923: } 923-924: } 924-925: } 925-926: } 926-927: } 927-928: } 928-929: } 929-929: } 929-930: } 930-931: } 931-932: } 932-933: } 933-934: } 934-935: } 935-936: } 936-937: } 937-938: } 938-939: } 939-939: } 939-940: } 940-941: } 941-942: } 942-943: } 943-944: } 944-945: } 945-946: } 946-947: } 947-948: } 948-949: } 949-949: } 949-950: } 950-951: } 951-952: } 952-953: } 953-954: } 954-955: } 955-956: } 956-957: } 957-958: } 958-959: } 959-959: } 959-960: } 960-961: } 961-962: } 962-963: } 963-964: } 964-965: } 965-966: } 966-967: } 967-968: } 968-969: } 969-969: } 969-970: } 970-971: } 971-972: } 972-973: } 973-974: } 974-975: } 975-976: } 976-977: } 977-978: } 978-979: } 979-979: } 979-980: } 980-981: } 981-982: } 982-983: } 983-984: } 984-985: } 985-986: } 986-987: } 987-988: } 988-989: } 989-989: } 989-990: } 990-991: } 991-992: } 992-993: } 993-994: } 994-995: } 995-996: } 996-997: } 997-998: } 998-999: } 999-999: } 999-1000: } 1000-1001: } 1001-1002: } 1002-1003: } 1003-1004: } 1004-1005: } 1005-1006: } 1006-1007: } 1007-1008: } 1008-1009: } 1009-1009: } 1009-1010: } 1010-1011: } 1011-1012: } 1012-1013: } 1013-1014: } 1014-1015: } 1015-1016: } 1016-1017: } 1017-1018: } 1018-1019: } 1019-1019: } 1019-1020: } 1020-1021: } 1021-1022: } 1022-1023: } 1023-1024: } 1024-1025: } 1025-1026: } 1026-1027: } 1027-1028: } 1028-1029: } 1029-1029: } 1029-1030: } 1030-1031: } 1031-1032: } 1032-1033: } 1033-1034: } 1034-1035: } 1035-1036: } 1036-1037: } 1037-1038: } 1038-1039: } 1039-1039: } 1039-1040: } 1040-1041: } 1041-1042: } 1042-1043: } 1043-1044: } 1044-1045: } 1045-1046: } 1046-1047: } 1047-1048: } 1048-1049: } 1049-1049: } 1049-1050: } 1050-1051: } 1051-1052: } 1052-1053: } 1053-1054: } 1054-1055: } 1055-1056: } 1056-1057: } 1057-1058: } 1058-1059: } 1059-1059: } 1059-1060: } 1060-1061: } 1061-1062: } 1062-1063: } 1063-1064: } 1064-1065: } 1065-1066: } 1066-1067: } 1067-1068: } 1068-1069: } 1069-1069: } 1069-1070: } 1070-1071: } 1071-1072: } 1072-1073: } 1073-1074: } 1074-1075: } 1075-1076: } 1076-1077: } 1077-1078: } 1078-1079: } 1079-1079: } 1079-1080: } 1080-1081: } 1081-1082: } 1082-1083: } 1083-1084: } 1084-1085: } 1085-1086: } 1086-1087: } 1087-1088: } 1088-1089: } 1089-1089: } 1089-1090: } 1090-1091: } 1091-1092: } 1092-1093: } 1093-1094: } 1094-1095: } 1095-1096: } 1096-1097: } 1097-1098: } 1098-1099: } 1099-1099: } 1099-1100: } 1100-1101: } 1101-1102: } 1102-1103: } 1103-1104: } 1104-1105: } 1105-1106: } 1106-1107: } 1107-1108: } 1108-1109: } 1109-1109: } 1109-1110: } 1110-1111: } 1111-1112: } 1112-1113: } 1113-1114: } 1114-1115: } 1115-1116: } 1116-1117: } 1117-1118: } 1118-1119: } 1119-1119: } 1119-1120: } 1120-1121: } 1121-1122: } 1122-1123: } 1123-1124: } 1124-1125: } 1125-1126: } 1126-1127: } 1127-1128: } 1128-1129: } 1129-1129: } 1129-1130: } 1130-1131: } 1131-1132: } 1132-1133: } 1133-1134: } 1134-1135: } 1135-1136: } 1136-1137: } 1137-1138: } 1138-1139: } 1139-1139: } 1139-1140: } 1140-1141: } 1141-1142: } 1142-1143: } 1143-1144: } 1144-1145: } 1145-1146: } 1146-1147: } 1147-1148: } 1148-1149: } 1149-1149: } 1149-1150: } 1150-1151: } 1151-1152: } 1152-1153: } 1153-1154: } 1154-1155: } 1155-1156: } 1156-1157: } 1157-1158: } 1158-1159: } 1159-1159: } 1159-1160: } 1160-1161: } 1161-1162: } 1162-1163: } 1163-1164: } 1164-1165: } 1165-1166: } 1166-1167: } 1167-1168: } 1168-1169: } 1169-1169: } 1169-1170: } 1170-1171: } 1171-1172: } 1172-1173: } 1173-1174: } 1174-1175: } 1175-1176: } 1176-1177: } 1177-1178: } 1178-1179: } 1179-1179: } 1179-1180: } 1180-1181: } 1181-1182: } 1182-1183: } 1183-1184: } 1184-1185: } 1185-1186: } 1186-1187: } 1187-1188: } 1188-1189: } 1189-1189: } 1189-1190: } 1190-1191: } 1191-1192: } 1192-1193: } 1193-1194: } 1194-1195: } 1195-1196: } 1196-1197: } 1197-1198: } 1198-1199: } 1199-1199: } 1199-1200: } 1200-1201: } 1201-1202: } 1202-1203: } 1203-1204: } 1204-1205: } 1205-1206: } 1206-1207: } 1207-1208: } 1208-1209: } 1209-1209: } 1209-1210: } 1210-1211: } 1211-1212: } 1212-1213: } 1213-1214: } 1214-1215: } 1215-1216: } 1216-1217: } 1217-1218: } 1218-1219: } 1219-1219: } 1219-1220: } 1220-1221: } 1221-1222: } 1222-1223: } 1223-1224: } 1224-1225: } 1225-1226: } 1226-1227: } 1227-1228: } 1228-1229: } 1229-1229: } 1229-1230: } 1230-1231: } 1231-1232: } 1232-1233: } 1233-1234: } 1234-1235: } 1235-1236: } 1236-1237: } 1237-1238: } 1238-1239: } 1239-1239: } 1239-1240: } 1240-1241: } 1241-1242: } 1242-1243: } 1243-1244: } 1244-1245: } 1245-1246: } 1246-1247: } 1247-1248: } 1248-1249: } 1249-1249: } 1249-1250: } 1250-1251: } 1251-1252: } 1252-1253: } 1253-1254: } 1254-1255: } 1255-1256: } 1256-1257: } 1257-1258: } 1258-1259: } 1259-1259: } 1259-1260: } 1260-1261: } 1261-1262: } 1262-1263: } 1263-1264: } 1264-1265: } 1265-1266: } 1266-1267: } 1267-1268: } 1268-1269: } 1269-1269: } 1269-1270: } 1270-1271: } 1271-1272: } 1272-1273: } 1273-1274: } 1274-1275: } 1275-1276: } 1276-1277: } 1277-127

File Edit View Git Project Debug | 🔎 | Lab\_11 | - □ X

Test Analyze Tools Extensions Window Help

Process: [25268] Tic Tac Toe.exe | Lifecycle Events

Program.cs

```
1 using System;
2 using System.Collections.Generic;
3
4 bool closeRequested = false;
5 bool playerTurn = true;
6 char[,] board;
7
8 while (!closeRequested)
9 {
10     board = new char[3, 3]
11     {
12         { ' ', ' ', ' ' },
13         { ' ', ' ', ' ' },
14         { ' ', ' ', ' ' }
15     };
16 }
```

100 % No issues found | Autos | Locals | Watch 1 | Call Stack | Search (Ctrl+E) | Call S... Break... Exce... Com... Imm... Outp...

Solution Explorer | Git Changes

Tic Tac Toe

Use the arrow and enter keys to select a move.

File Edit View Git Project Debug | 🔎 | Lab\_11 | - □ X

Test Analyze Tools Extensions Window Help

Process: [21844] Tic Tac Toe.exe | Lifecycle Events

Program.cs

```
16     while (!closeRequested)
17     {
18         if (playerTurn)
19         {
20             PlayerTurn();
21             if (CheckForThree('X'))
22             {
23                 EndGame(" You Win.");
24                 break;
25             }
26         }
27         else
28         {
29             ComputerTurn();
30             if (CheckForThree('X'))
```

00 % No issues found | Autos | Locals | Watch 1 | Call Stack | Search (Ctrl+E) | Call S... Break... Exce... Com... Imm... Outp...

Solution Explorer | Git Changes

Tic Tac Toe

Use the arrow and enter keys to select a move.

So I checked if Computer Turn was being called or not. But this was happening because I am going in an infinite loop within playerTurn, because in the below line it should be `&&`.

The screenshot shows the Visual Studio IDE with the project "Lab\_11" open. The code editor displays `Program.cs` with the following code snippet:

```

59     }
60     Console.CursorVisible = true;
61
62     void PlayerTurn()
63     {
64         var (row, column) = (0, 0);
65         bool moved = false;
66         while (!moved || !closeRequested) < 1ms elapsed
67         {
68             Console.Clear();
69             RenderBoard();
70             Console.WriteLine();
71             Console.WriteLine(" Use the arrow and enter keys to select a move.");
72             Console.SetCursorPosition(column * 4 + 4, row * 2 + 1);
73             Console.CursorVisible = true;
    
```

The call stack window shows two frames:

- `Tic Tac Toe.dll!Program.<Main>$_PlayerT... C#`
- `Tic Tac Toe.dll!Program.<Main>$<string[] ... C#`

The status bar at the bottom indicates "No issues found".

### Bug 3:

I continued playing and I noticed that even after 'O' has won the game is still continuing.

The screenshot shows the Visual Studio IDE with the project "Lab\_11" open. The code editor displays `Program.cs` with the following code snippet:

```

9     {
10        board = new char[3, 3];
11        {
12            { ' ', ' ', ' ' },
13            { ' ', ' ', ' ' },
14            { ' ', ' ', ' ' },
15        };
16        while (!closeRequested)
17        {
18            if (playerTurn)
19            {
20                PlayerTurn();
21                if (CheckForThree('X'))
22                {
23                    EndGame(" You Win.");
    
```

The call stack window shows one frame:

- `Tic Tac Toe.dll!Program.<Main>$_PlayerT... C#`

The status bar at the bottom indicates "No issues found".

So I checked the winning conditions and the winning function for 'O' and there is no checking done for 'O' winning. So I changed it.

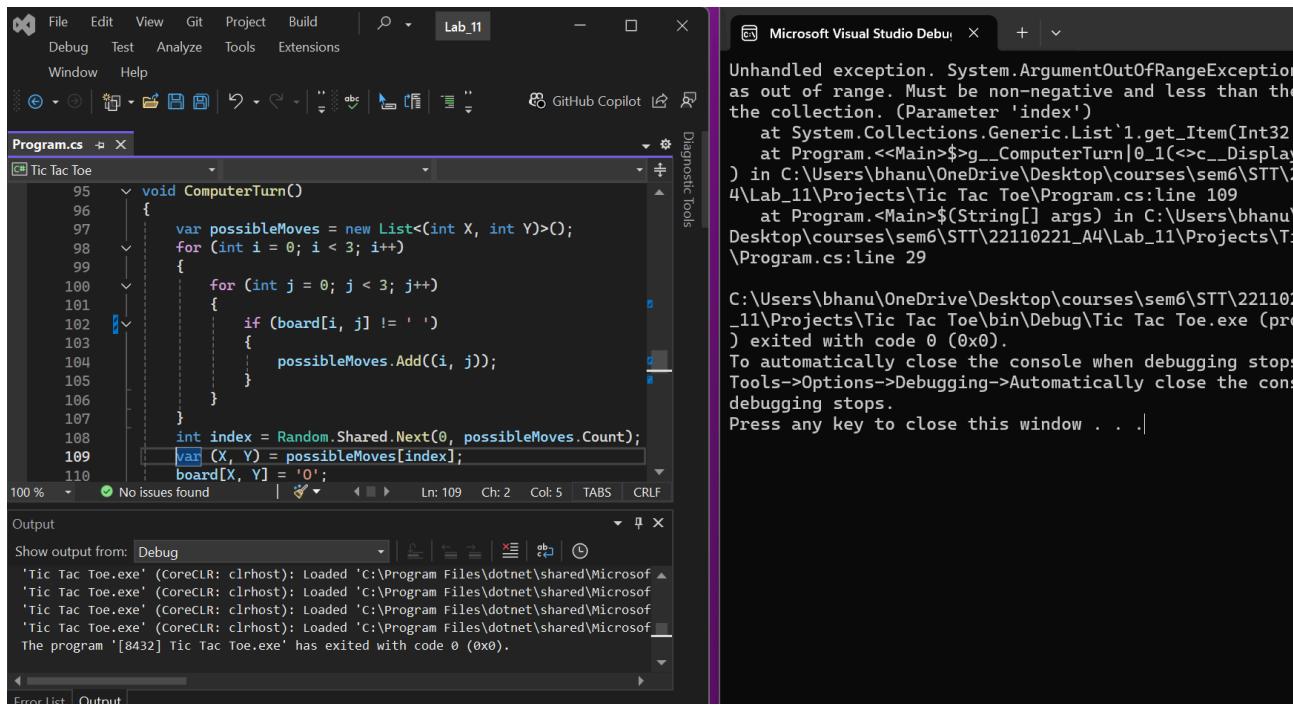
```

        if (playerTurn)
    {
        PlayerTurn();
        if (CheckForThree('X'))
        {
            EndGame(" You Win.");
            break;
        }
    }
    else
    {
        ComputerTurn();
        if (CheckForThree('X'))
        {
            EndGame(" You Lose.");
        }
    }
}

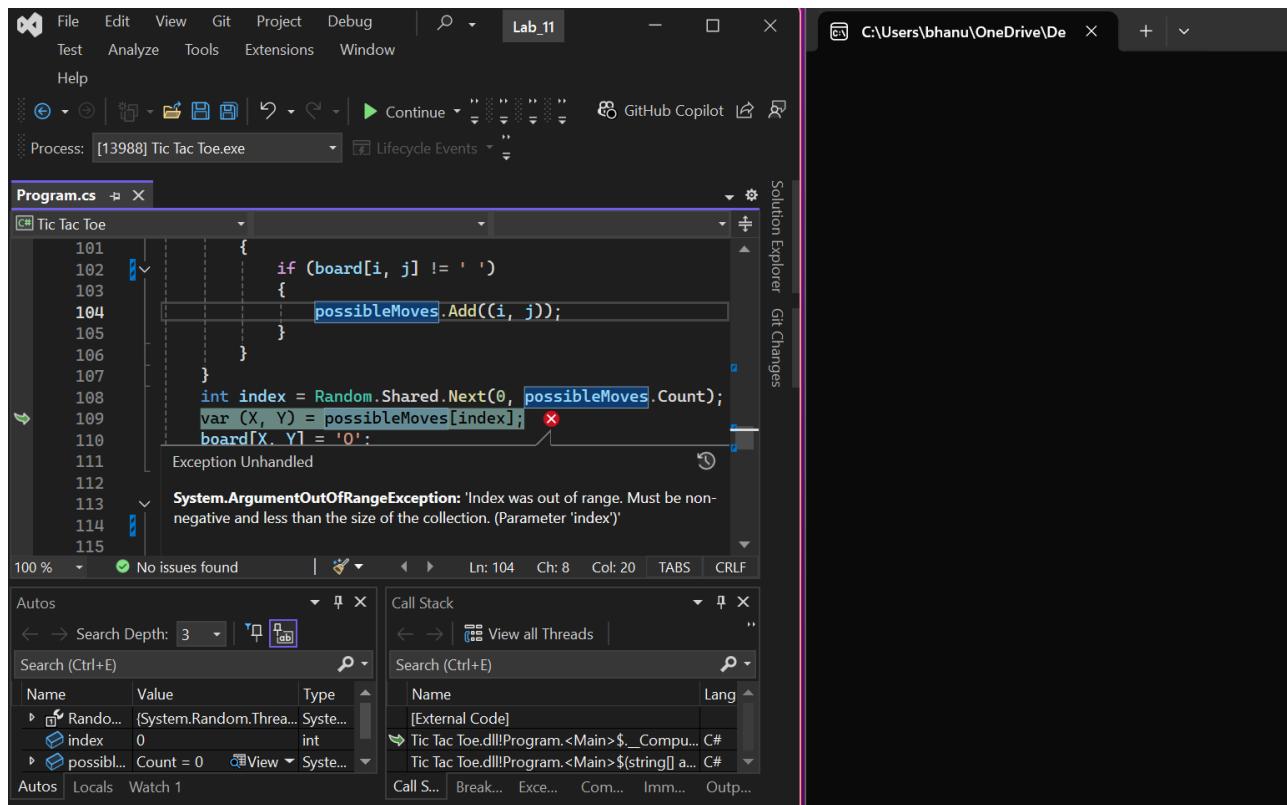
```

#### Bug 4:

As I continued playing the game, I got the error that the index is out of range. This arised in ComputerTurn().



I couldn't find anything using step over so I used step-into. Then I found this



In the bottom, it says possible moves **count = 0**. As we are trying to read from an empty array this error is arising but why is possible moves empty before the game is played.

Computer was not able to find any turn because it is not adding the possible moves which it found into the array

```

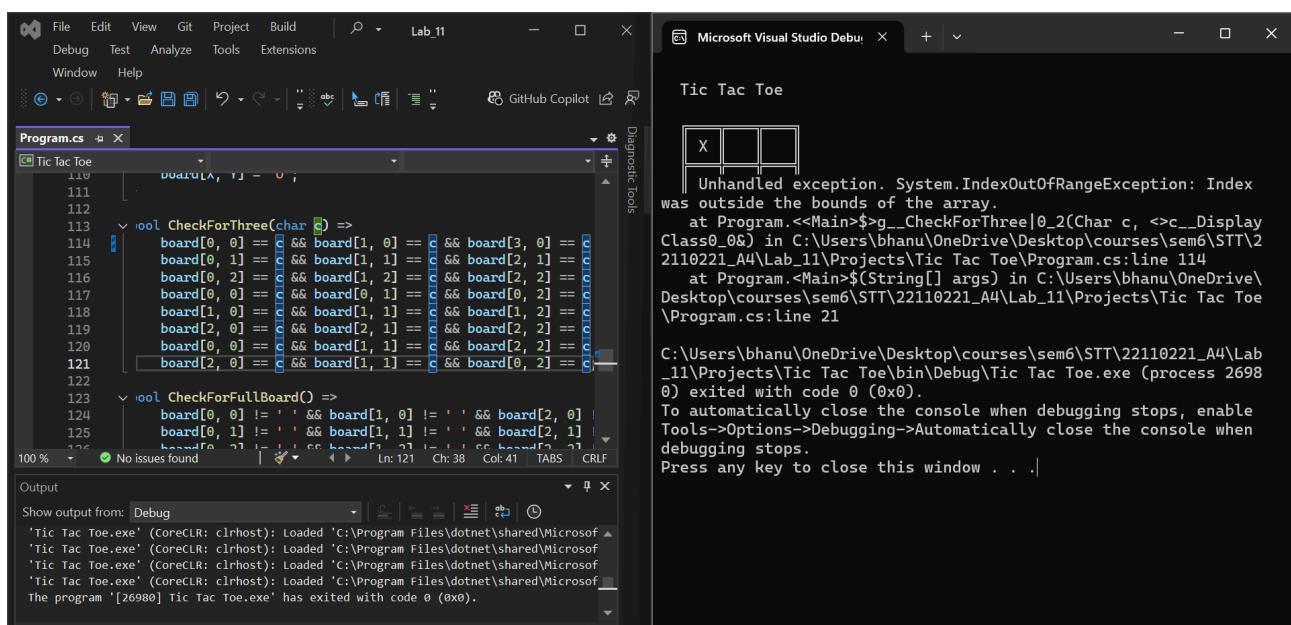
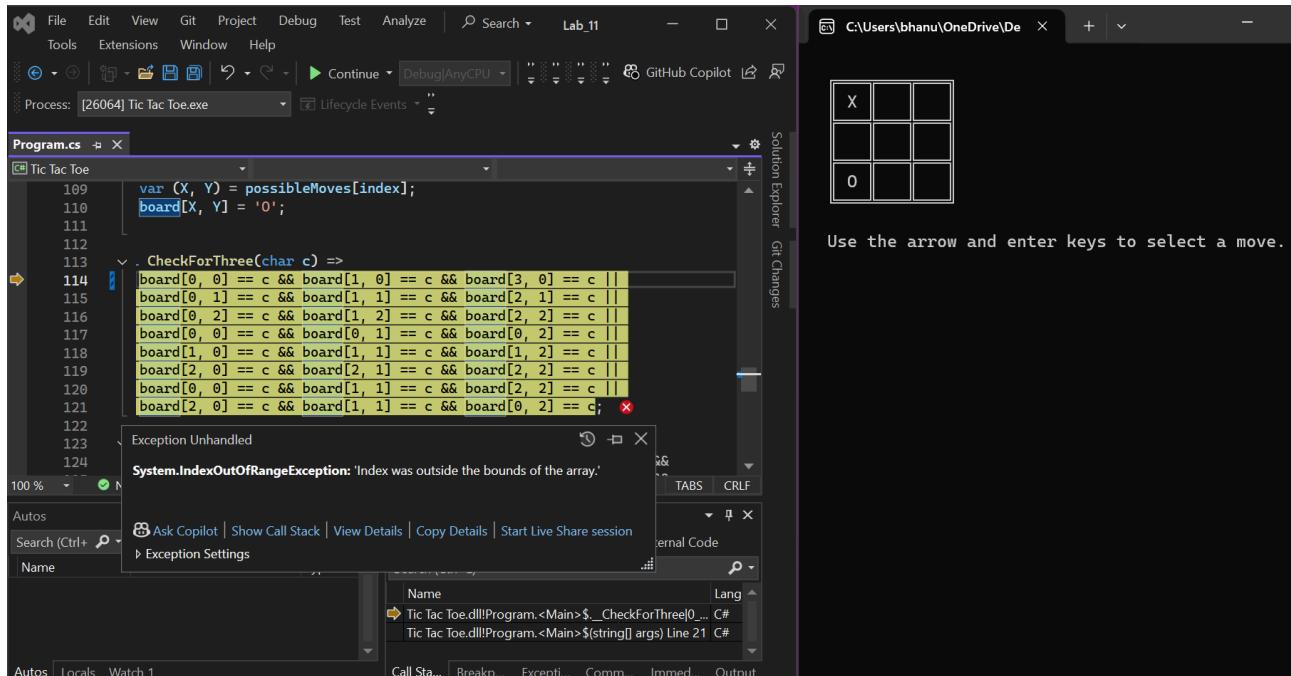
void ComputerTurn()
{
    var possibleMoves = new List<(int X, int Y)>();
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (board[i, j] != ' ')
            {
                possibleMoves.Add((i, j));
            }
        }
    }
}

```

So I corrected it. And continued playing the game.

**Bug 5:**

For a few cases, I faced index out of range exception, so I kept playing to notice the pattern.



```

    board = new char[3, 3]
    {
        { ' ', ' ', ' ' },
        { ' ', ' ', ' ' },
        { ' ', ' ', ' ' },
    };
    while (!closeRequested)
    {
        if (playerTurn)
        {
            PlayerTurn();
            if (CheckForThree('X'))
            {
                EndGame(" You Win.");
                break;
            }
        }
    }

```

```

111     }
112
113     < bool CheckForThree(char c) =>
114         board[0, 0] == c && board[1, 0] == c && board[3, 0] == c
115         board[0, 1] == c && board[1, 1] == c && board[2, 1] == c
116         board[0, 2] == c && board[1, 2] == c && board[2, 2] == c
117         board[0, 0] == c && board[0, 1] == c && board[0, 2] == c
118         board[1, 0] == c && board[1, 1] == c && board[1, 2] == c
119         board[2, 0] == c && board[2, 1] == c && board[2, 2] == c
120         board[0, 0] == c && board[1, 0] == c && board[2, 0] == c
121         board[2, 0] == c && board[1, 1] == c && board[0, 2] == c
122
123     < bool CheckForFullboard() =>
124         board[0, 0] != ' ' && board[1, 0] != ' ' && board[2, 0]
125         board[0, 1] != ' ' && board[1, 1] != ' ' && board[2, 1]
126         board[0, 2] != ' ' && board[1, 2] != ' ' && board[2, 2]

```

Unhandled exception. System.IndexOutOfRangeException: Index was outside the bounds of the array.  
at Program.<Main>\$>g\_\_CheckForThree|0\_2(Char c, <>c\_\_DisplayClass0\_0&)  
at C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221\_A4\Lab\_11\Projects\Tic Tac Toe\Program.cs:line 114  
at Program.<Main>\$>{String[] args} in C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221\_A4\Lab\_11\Projects\Tic Tac Toe\Program.cs:line 21

C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221\_A4\Lab\_11\Projects\Tic Tac Toe\bin\Debug\Tic Tac.exe (process 6576) exited with code 0 (0x0).  
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.  
Press any key to close this window . . .

This error is arising only in the case when both board[0][0] and board[1][0] are filled (with any symbol). This is because the bug here is only identified when the first two statements are true then it needs to check the 3rd statement which is out of range.

```

< bool CheckForThree(char c) =>
    board[0, 0] == c && board[1, 0] == c && board[3, 0] == c ||
    board[0, 1] == c && board[1, 1] == c && board[2, 1] ==

```

All the bugs are found!

## Step 4: Analyzing Rock Paper Scissors

I started by playing a few rounds without debugging.

I lost

```
Rock, Paper, Scissors  
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:  
The computer chose Paper.  
You lose.  
Score: 0 wins, 1 losses, 0 draws  
Press Enter To Continue...
```

I won

```
Rock, Paper, Scissors  
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:  
The computer chose Rock.  
You win.  
Score: 1 wins, 1 losses, 0 draws  
Press Enter To Continue...
```

Draw

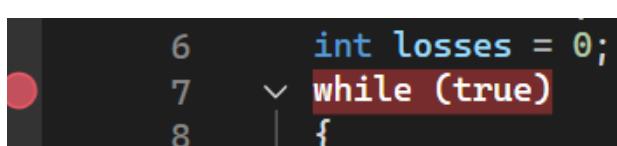
```
Rock, Paper, Scissors  
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:  
The computer chose Scissors.  
This game was a draw.  
Score: 3 wins, 3 losses, 1 draws  
Press Enter To Continue...
```

Invalid

```
Rock, Paper, Scissors  
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:  
Invalid Input. Try Again...  
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:
```

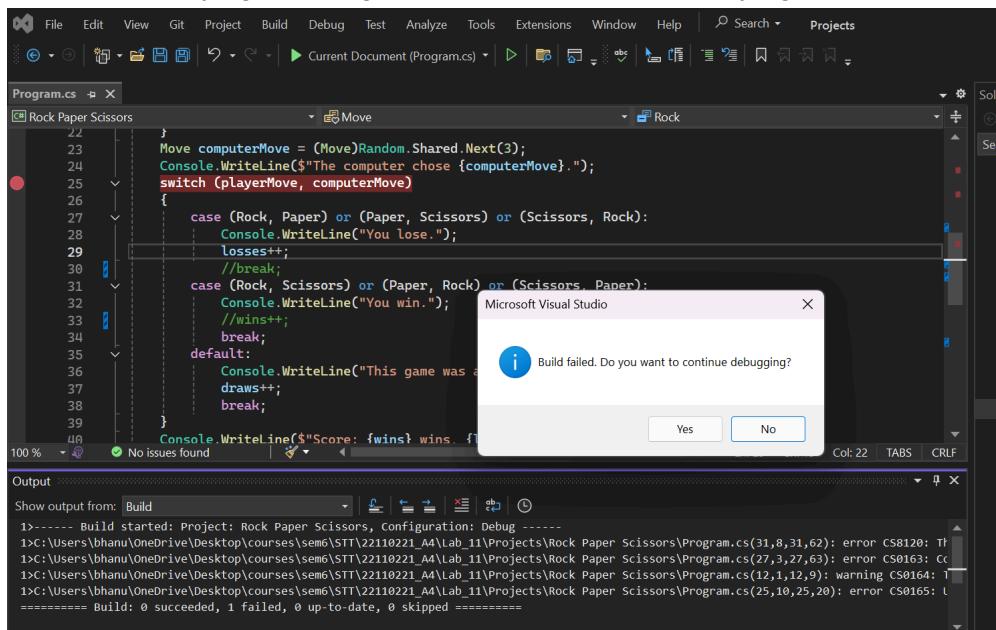
Then, added break points.

Breakpoints



```
Move computerMove = (Move)Random.Shared.Next(3);  
Console.WriteLine($"The computer chose {computerMove}.")  
switch (playerMove, computerMove)  
{
```

And started playing in debug mode but before I could play I got these errors.



```
9): warning CS0164: This label has not been referenced  
,20): error CS0165: Use of unassigned local variable 'playerMove'
```

These errors refer to not using goto for a label, hence I added it. And the other one says playerMove is not initialized so I initialised it to -1 (invalid).

```

    Console.WriteLine("Choose your move");
    goto GetInput;
    Console.Write("Computer move: ");
    Move computerMove;
}

Console.WriteLine("Choose your move");
Move playerMove = -1;
switch ((Console.ReadLine().ToLower())

```

Everything is working as expected so I inserted 5 bugs

1. I replace wins with losses

```

case (Rock, Scissors) or (Paper, Rock) or (Scissors, Paper):
    Console.WriteLine("You win.");
    wins++;
    break;

    break;
case (Rock, Scissors) or (Paper, Rock) or (Scissors, Paper):
    Console.WriteLine("You win.");
    losses++;
    break;
}

```

2. Removing one of the win conditions

```

case (Rock, Scissors) or (Paper, Rock) or (Scissors, Paper):
    Console.WriteLine("You win.");
    losses++;
    break;

case (Rock, Scissors) or (Paper, Rock):
    Console.WriteLine("You win.");
    losses++;
    break;
}

```

3. I replaced case 'e' return with break;

```

case "p" or "paper": playerMove = Paper; break;
case "s" or "scissors": playerMove = Scissors; break;
case "e" or "exit": Console.Clear(); return;
default:Console.WriteLine("Invalid Input. Try Again..."); 
goto GetInput;

case "r" or "rock": playerMove = Rock; break;
case "p" or "paper": playerMove = Paper; break;
case "s" or "scissors": playerMove = Scissors; break;
case "e" or "exit": Console.Clear(); break;
default:Console.WriteLine("Invalid Input. Try Again..."); 
goto GetInput;

```

4. Make the computer generate an invalid move

```

}

Move computerMove = (Move)Random.Shared.Next(3);
Console.WriteLine($"The computer chose {computerMove}.");

```

```
Move computerMove = (Move)Random.Shared.Next(4);
Console.WriteLine($"The computer chose {computerMove}.");
```

5. Consider Rock vs Rock as a lose

```
case (Rock, Paper) or (Paper, Scissors) or (Scissors, Rock):
    Console.WriteLine("You lose.");
    losses++;
    break;

case (Rock, Rock) or (Paper, Scissors) or (Scissors, Rock):
    Console.WriteLine("You lose.");
    losses++;
    break;
```

## Step 5: Debugging Rock Paper Scissors

Before I could start debugging, the build failed.

First I started with step-over debugging.

### Bug 1:

And as I played the game. It showed a draw when I chose rock and the computer chose paper.  
This is incorrect.

Upon checking the draw or default case i.e by checking the winning and losing condition, I noticed that there is no (Rock,Paper) condition. It should be in the losing condition. 1st bug detected.

### Bug 2:

Then by playing a few rounds, I noticed another issue. It is saying Rock vs Rock as a lose.  
Reviewing the winning and losing condition once again I noticed the issue and removed the Rock vs Rock condition as the lose condition.

```
case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock) or (Rock, Rock):
    Console.WriteLine("You lose.");
    losses++;
    break;

case (Rock, Scissors) or (Paper, Rock):
switch (playerMove, computerMove)
{
    case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock):
        Console.WriteLine("You lose.");
        losses++;
        break;
```

Finally I made sure all the condition are correct

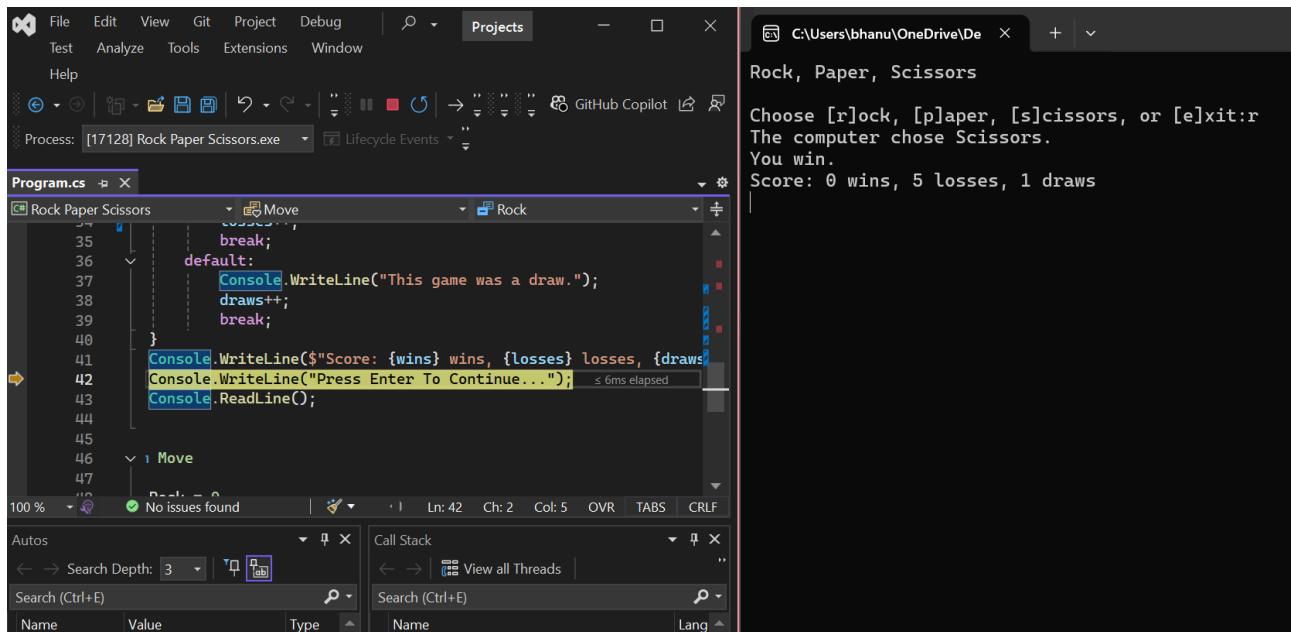
```

switch (playerMove, computerMove)
{
    case (Scissors, Rock) or (Paper, Scissors) or (Rock, Paper):
        Console.WriteLine("You lose.");
        losses++;
        break;
    case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock):
        Console.WriteLine("You win.");
        wins++;
        break;
    default:
        Console.WriteLine("This game was a draw.");
        draws++;
        break;
}

```

### Bug 3:

After playing several rounds, I noticed that even after winning a few rounds the computer is viewing no wins by player. But it acknowledged that I won that round.



Upon checking the case, It seems that no matter who wins, the score is being added to the computer i.e losses are getting incremented every time. So I corrected it.

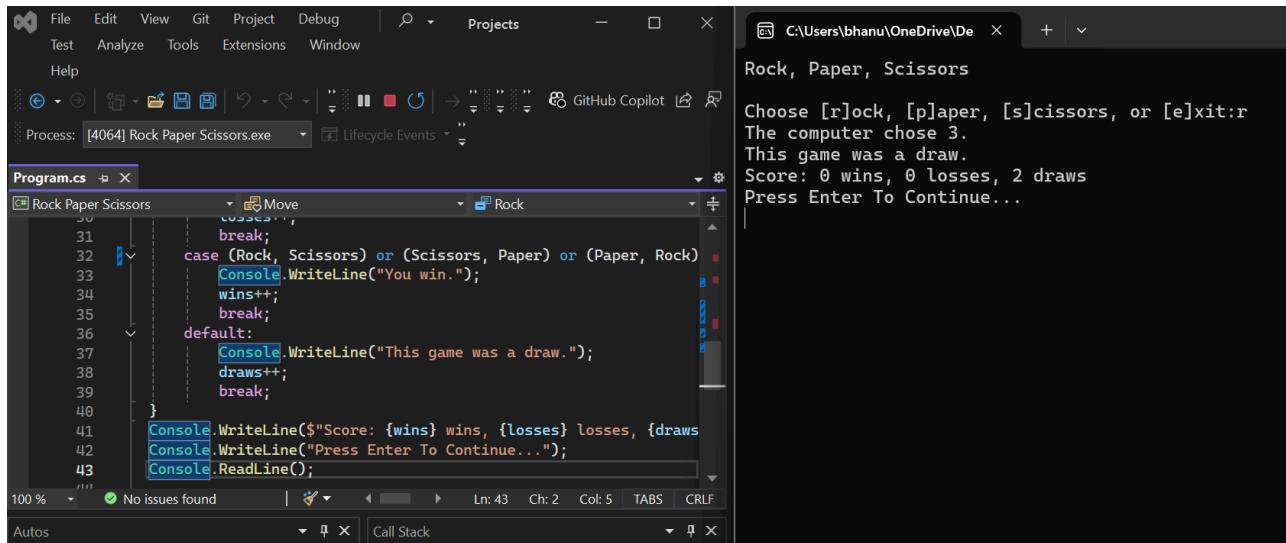
```

switch (playerMove, computerMove)
{
    case (Scissors, Rock) or (Paper, Scissors):
        Console.WriteLine("You lose.");
        losses++;
        break;
    case (Rock, Scissors) or (Scissors, Paper):
        Console.WriteLine("You win.");
        wins++;
        break;
    default:

```

## Bug 4:

When I was playing I found that the computer is sometimes showing a valid move and sometimes as invalid move like 3.



```
File Edit View Git Project Debug | 🔍 Projects - X
Test Analyze Tools Extensions Window Help
Process: [4064] Rock Paper Scissors.exe Lifecycle Events
Program.cs ✘ X
Rock Paper Scissors
    ...
31     break;
32     case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock)
33         Console.WriteLine("You win.");
34         wins++;
35         break;
36     default:
37         Console.WriteLine("This game was a draw.");
38         draws++;
39         break;
40     }
41     Console.WriteLine($"Score: {wins} wins, {losses} losses, {draws}
42     Console.WriteLine("Press Enter To Continue...");
43     Console.ReadLine();
100 % No issues found | 🔍 Autos Call Stack
```

Rock, Paper, Scissors

Choose [r]ock, [p]aper, [s]cissors, or [e]xit:r  
The computer chose 3.  
This game was a draw.  
Score: 0 wins, 0 losses, 2 draws  
Press Enter To Continue...

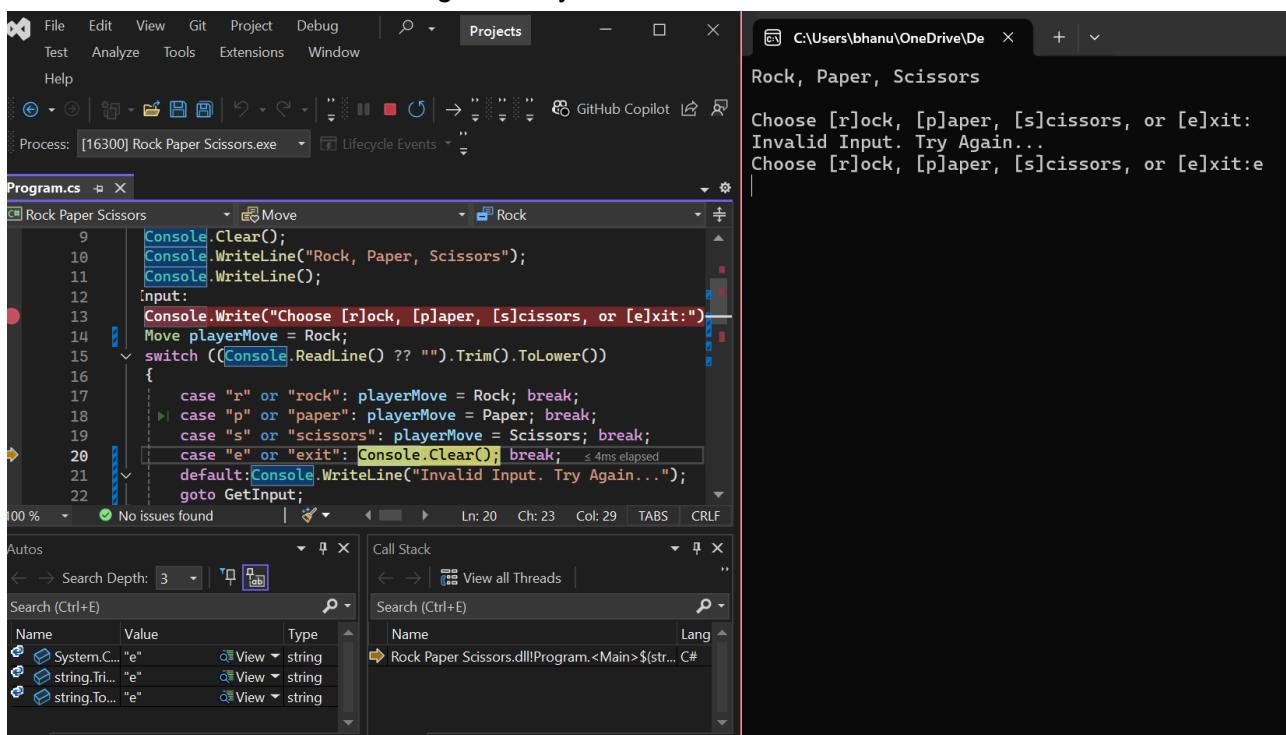
This is because of :

`Move computerMove = (Move)Random.Shared.Next(4);`

So I set it to 3.

## Bug 5:

This time all the cases are working correctly so I wanted to exit



```
File Edit View Git Project Debug | 🔍 Projects - X
Test Analyze Tools Extensions Window Help
Process: [16300] Rock Paper Scissors.exe Lifecycle Events
Program.cs ✘ X
Rock Paper Scissors
    ...
9     Console.Clear();
10    Console.WriteLine("Rock, Paper, Scissors");
11    Console.WriteLine();
12    input:
13    Console.Write("Choose [r]ock, [p]aper, [s]cissors, or [e]xit:");
14    Move playerMove = Rock;
15    switch ((Console.ReadLine() ?? "").Trim().ToLower())
16    {
17        case "r" or "rock": playerMove = Rock; break;
18        case "p" or "paper": playerMove = Paper; break;
19        case "s" or "scissors": playerMove = Scissors; break;
20        case "e" or "exit": Console.Clear(); break;
21        default: Console.WriteLine("Invalid Input. Try Again..."); goto GetInput;
22    }
100 % No issues found | 🔍 Autos Call Stack
<- → Search Depth: 3 🔍 Search (Ctrl+E)
Name Value Type
System.C... "e" string
string.Iri... "e" string
string.To... "e" string
```

Rock, Paper, Scissors

Choose [r]ock, [p]aper, [s]cissors, or [e]xit:  
Invalid Input. Try Again...  
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:e

The screenshot shows the Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Debug, Test, Analyze, Tools, Extensions, and Window. The title bar indicates the project is "Rock Paper Scissors" and the file is "Program.cs". The code editor displays the following C# code:

```
    losses++;  
    break;  
    case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock)  
        Console.WriteLine("You win.");  
        wins++;  
        break;  
    default:  
        Console.WriteLine("This game was a draw.");  
        draws++;  
        break;  
    }  
    Console.WriteLine($"Score: {wins} wins, {losses} losses, {draws}  
    Console.WriteLine("Press Enter To Continue...");  
    Console.ReadLine();
```

The status bar at the bottom shows "100 % No issues found". On the right side, there are several tool windows: "Call Stack", "Search (Ctrl+E)", and "Output". The "Output" window shows the message "The computer chose Rock.".

But the game kept on going, the computer still chose a player.

The screenshot shows the Visual Studio IDE interface with the same project and file as the previous screenshot. The code editor now displays the corrected C# code:

```
    losses++;  
    break;  
    case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock)  
        Console.WriteLine("You win.");  
        wins++;  
        break;  
    default:  
        Console.WriteLine("This game was a draw.");  
        draws++;  
        break;  
    }  
    Console.WriteLine($"Score: {wins} wins, {losses} losses, {draws}  
    Console.WriteLine("Press Enter To Continue...");  
    Console.ReadLine();
```

The status bar at the bottom shows "100 % No issues found". The "Output" window now displays the messages "The computer chose Rock.", "This game was a draw.", "Score: 0 wins, 1 losses, 3 draws", and "Press Enter To Continue...".

Upon checking the exit condition. It is set to break rather than return.

All bugs are found!

### 3. Results and Analysis

The debugging activities were successfully conducted on the selected C# console games (e.g., Tic Tac Toe, Rock Paper Scissors) using Visual Studio 2022. Program execution flow was visually traced using the debugger by setting breakpoints at key logic points (start of loops, function calls, conditional checks) and utilizing step operations (Step Into, Step Over, Step Out) to observe the sequence. This process provided clear insight into how the applications handled user input, updated game state, and evaluated conditions.

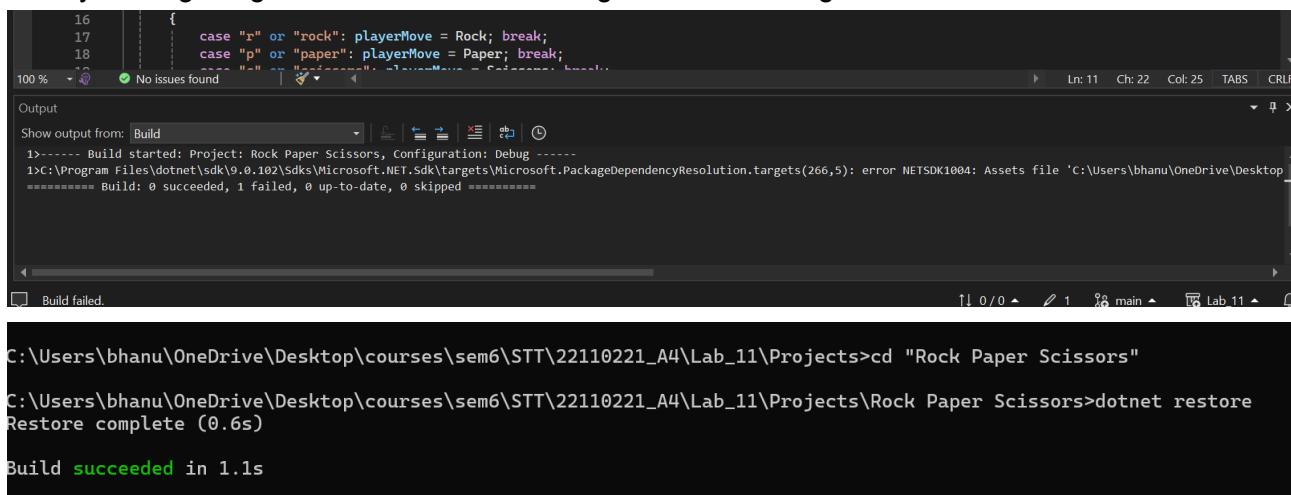
A total of five distinct bugs were successfully injected and subsequently resolved. These involved both simulated runtime errors (e.g., incorrect enum casts leading to faulty comparisons, operator mutations causing logic flaws) and logical errors resulting in incorrect game behavior (e.g., flawed win/loss conditions, biased AI moves, incorrect scorekeeping). Analysis via the debugger, particularly by monitoring variable values in the Locals window and observing the execution path, allowed for precise pinpointing of the root cause of each bug. The code was then corrected (e.g., fixing enum values, reverting mutations, correcting conditional logic), rebuilt, and re-executed to confirm the successful resolution of each issue. This analysis confirms the debugger's critical role in identifying and understanding runtime/logic errors that are not detected during compile-time.

---

## 4. Discussion and Conclusion

### Challenges and Reflections

Initially I was getting a build failed error and it got resolved using restore dotnet



The screenshot shows the Visual Studio 2022 interface. The code editor displays a portion of a C# file with code related to a switch statement. The output window below shows a build log:

```
16
17     {
18         case "r" or "rock": playerMove = Rock; break;
19         case "p" or "paper": playerMove = Paper; break;
20         default: playerMove = Scissors; break;
}
100 % No issues found
Output
Show output from: Build
1>----- Build started: Project: Rock Paper Scissors, Configuration: Debug -----
1>C:\Program Files\dotnet\sdk\9.0.102\Sdks\Microsoft.NET.Sdk\targets\Microsoft.PackageDependencyResolution.targets(266,5): error NETSDK1004: Assets file 'C:\Users\bhanu\OneDrive\Desktop\Rock Paper Scissors\Rock Paper Scissors.csproj' not found.
=====
Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped
Build failed.
```

Below the output window, a terminal window shows the command line steps taken to resolve the issue:

```
C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221_A4\Lab_11\Projects>cd "Rock Paper Scissors"
C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221_A4\Lab_11\Projects\Rock Paper Scissors>dotnet restore
Restore complete (0.6s)

Build succeeded in 1.1s
```

Initial challenges included differentiating compiler errors from runtime bugs and understanding debugger interactions with console input (`Console.ReadKey()`). Intentionally creating non-compiler-breaking bugs required careful thought about runtime logic rather than just syntax. Effectively tracing complex conditional logic (like the game's win/loss switch) using the debugger took practice.

I created bugs like removing default statements or removing the body from the case statements and these were easily detected by the compiler itself.

The hands-on debugging process significantly clarified how code executes dynamically. Witnessing variable changes and control flow shifts in real-time was highly instructive. Injecting bugs proved to be a valuable exercise in understanding code fragility and the importance of thorough testing.

## Lessons Learned

Debugger Navigation: Proficient use of Step Into (F11), Step Over (F10), and Step Out (Shift+F11) for controlling execution flow.

State Inspection: Utilizing debugger windows (Locals, Autos, Watch) to monitor variable values during runtime.

Breakpoint Strategy: Placing breakpoints effectively to isolate problems and analyze specific code sections.

Runtime vs. Compile-time: Clearly distinguishing between errors caught by the compiler (syntax, static analysis) and those needing the debugger (runtime exceptions, logic flaws).

Bug Causation: Analyzing how small code changes (mutations) can lead to significant runtime errors or incorrect behavior.

## Summary

This lab provided essential hands-on experience with the Visual Studio Debugger using a C# console game. Activities involved tracing program execution, identifying and fixing injected runtime bugs, and documenting the process. Key debugging techniques were practiced, enhancing understanding of code analysis, runtime behavior, and fault diagnosis in C# applications.

---

## 5. References

1. <https://github.com/dotnet/dotnet-console-games>
2. <https://visualstudio.microsoft.com>
3. <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/program-structure/top-level-statements>
4. <https://learn.microsoft.com/en-us/visualstudio/debugger>
5. Google AI studio for debugging, refining the grammar and formatting of this report

# Lab Assignment 12

**Lab Topic: Event-driven Programming for Windows Forms Apps. in C#**

## 1. Introduction

This project explores fundamental C# programming concepts through the development of two alarm clock applications. The first task utilizes a console interface to demonstrate core event handling and time monitoring using the publisher/subscriber pattern. The second task transitions to a graphical user interface (GUI) using Windows Forms, showcasing event-driven programming, UI responsiveness, and visual feedback common in desktop applications. Both tasks involve handling user input, validating data, working with date/time structures, and triggering actions based on time comparisons, providing practical experience in different C# application models.

### Environment Setup:

- Operating System: Windows
- Software: Visual Studio 2022 (Community Edition), Visual Studio with .NET SDK
- Programming Language: C# 13 or dotnet SDK 9.0.102

### Key Concepts:

**Windows Forms (WinForms):** A .NET GUI framework for building desktop applications with visual elements like forms, buttons, text boxes, etc. It operates on an event-driven model.

**Event-Driven Programming:** A programming paradigm where the flow of the program is determined by events (e.g., user clicks a button, a timer ticks). Code execution happens in response to these events.

**Events and Delegates (Action, EventHandler):** A mechanism in C# for enabling communication between objects (publisher/subscriber). Delegates define the required signature for event handler methods, and events allow classes to notify subscribers when something significant happens.

**Partial Classes:** A C# feature allowing a class definition to be split across multiple source files (e.g., Form1.cs and Form1.Designer.cs), commonly used by WinForms designer.

---

## 2. Methodology and Execution

This is the file structure in the folder Lab12.

### File Structure:

```
└── Lab12/      # Console Alarm Project Folder
    ├── Lab12.csproj  # Project build settings/dependencies
    ├── Lab12.sln    # Solution file (groups projects)
    ├── Program.cs   # Main application code/entry point
    └── bin/         # Compiled output binaries folder
        └── obj/       # Intermediate build files folder

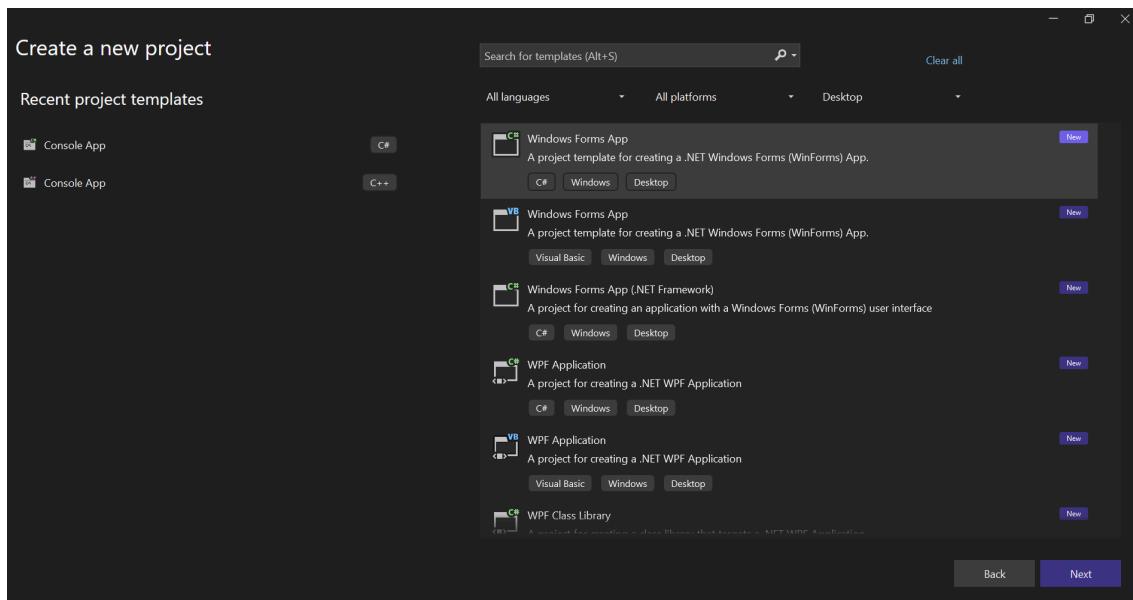
└── Lab12_2/      # WinForms Alarm Project Folder
    ├── Lab12_2.csproj  # Project build settings/dependencies
    ├── Lab12_2.sln    # Solution file (groups projects)
    ├── Form1.cs      # Form's user code-behind logic
    ├── Form1.Designer.cs # Designer-generated UI code
    ├── Form1.resx     # Form resources (images, strings)
    ├── Program.cs    # Main application entry point
    └── Lab12_2.csproj.user # User-specific project settings (optional)
        └── bin/         # Compiled output binaries folder
            └── obj/       # Intermediate build files folder
```

Below is the step-by-step procedure I followed.

### Step 1: Initial Setup

For task1: I created a new C# "Console App" project in Visual Studio, named the project Lab12 and ensured the target .NET framework was suitable. Visual Studio generated the initial project structure, including Program.cs.

For task2: I Created a new C# "Windows Forms App" project in Visual Studio, named the project Lab12\_2 and confirmed the target .NET framework. Visual Studio generated the initial project files (Program.cs, Form1.cs, Form1.Designer.cs).



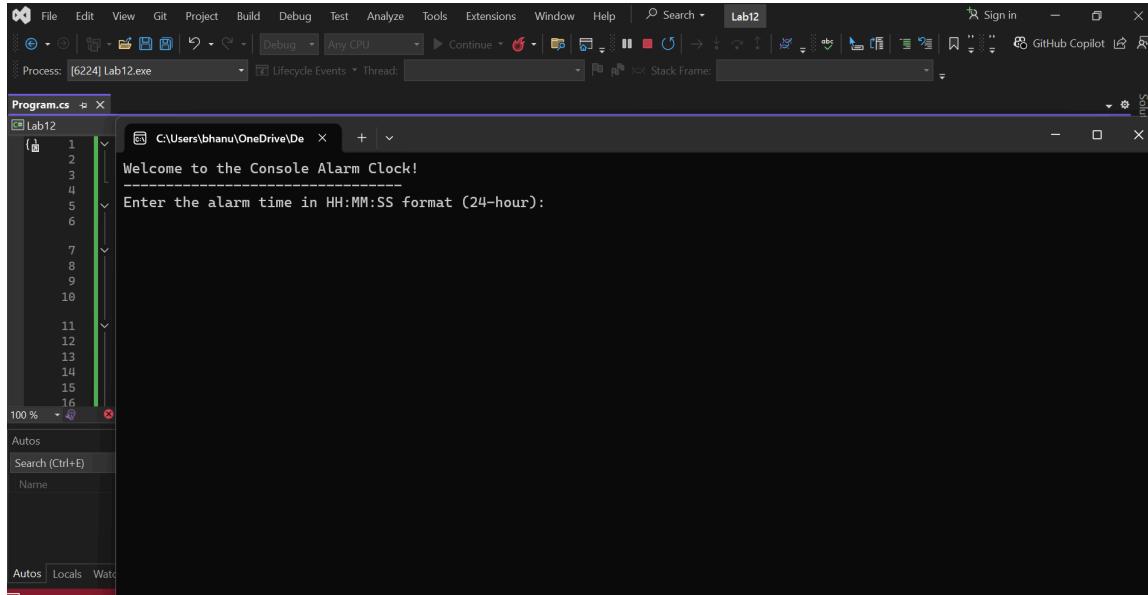
## Step 2: Implementing Input, Validation, and Core Logic (Program.cs)

Inside the Main method of Program.cs, I first set up the publisher/subscriber mechanism:

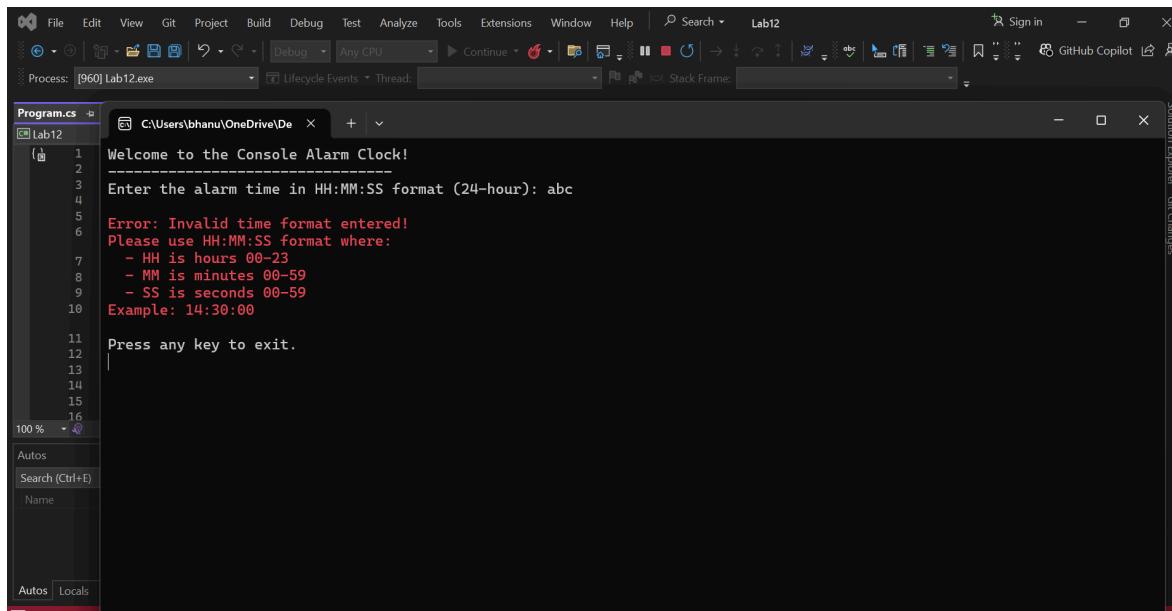
- Declared a static event: public static event Action RaiseAlarm; (Publisher).
- Defined the static event handler method: static void RingAlarm() which prints the alarm message (Subscriber logic).
- Subscribed the RingAlarm method to the RaiseAlarm event: RaiseAlarm += RingAlarm;.

Next, I implemented the user interaction and time handling:

- Prompted the user to enter the alarm time in HH:MM:SS format using Console.WriteLine.



- Validated the input string using a helper method (`IsValidTimeFormat`) which employed Regular Expressions to ensure the format was strictly HH:MM:SS (24-hour).
- If the input was invalid, I displayed a detailed error message explaining the correct format and exited the application.



The screenshot shows the Visual Studio IDE with the code editor open. The file is named `Program.cs` and contains the following code:

```

1 Welcome to the Console Alarm Clock!
2 -----
3 Enter the alarm time in HH:MM:SS format (24-hour): abc
4
5 Error: Invalid time format entered!
6 Please use HH:MM:SS format where:
7   - HH is hours 00-23
8   - MM is minutes 00-59
9   - SS is seconds 00-59
10 Example: 14:30:00
11
12 Press any key to exit.
13
14
15
16

```

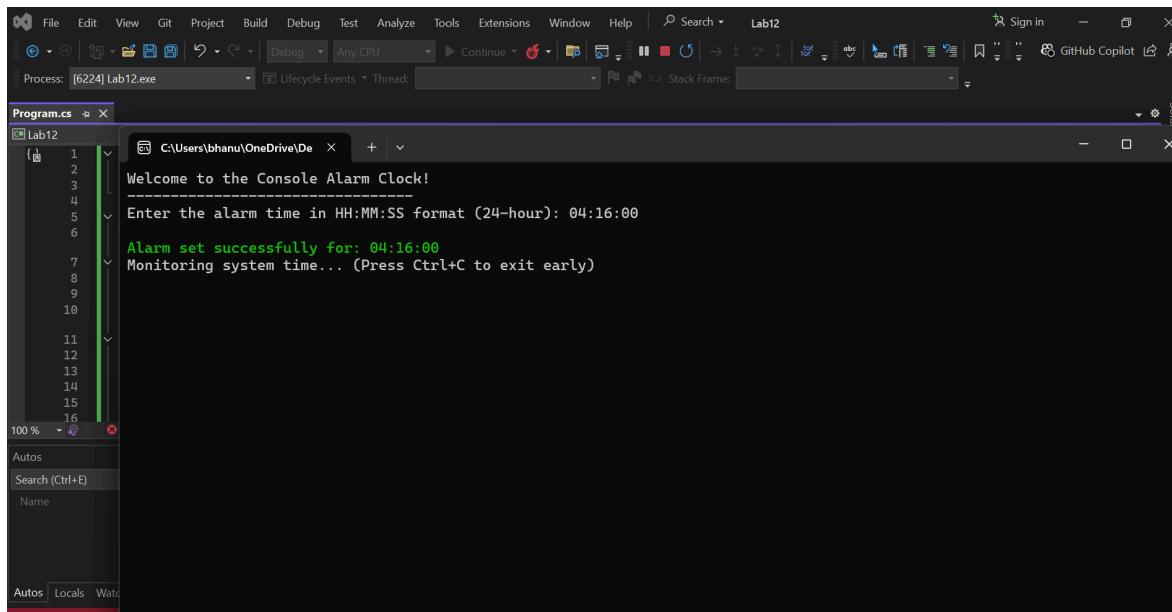
The output window shows the application's response to an invalid input ('abc'):

```

Process: [960] Lab12.exe
Lifecycle Events | Thread: Stack Frame: 

```

- If the input was valid, I parsed the string into a `TimeSpan` object (`alarmTime = TimeSpan.Parse(inputTime);`) and confirmed the set time back to the user.



The screenshot shows the Visual Studio IDE with the code editor open. The file is named `Program.cs` and contains the following code:

```

1 Welcome to the Console Alarm Clock!
2 -----
3 Enter the alarm time in HH:MM:SS format (24-hour): 04:16:00
4
5 Alarm set successfully for: 04:16:00
6 Monitoring system time... (Press Ctrl+C to exit early)
7
8
9
10
11
12
13
14
15
16

```

The output window shows the application's response to a valid input ('04:16:00'):

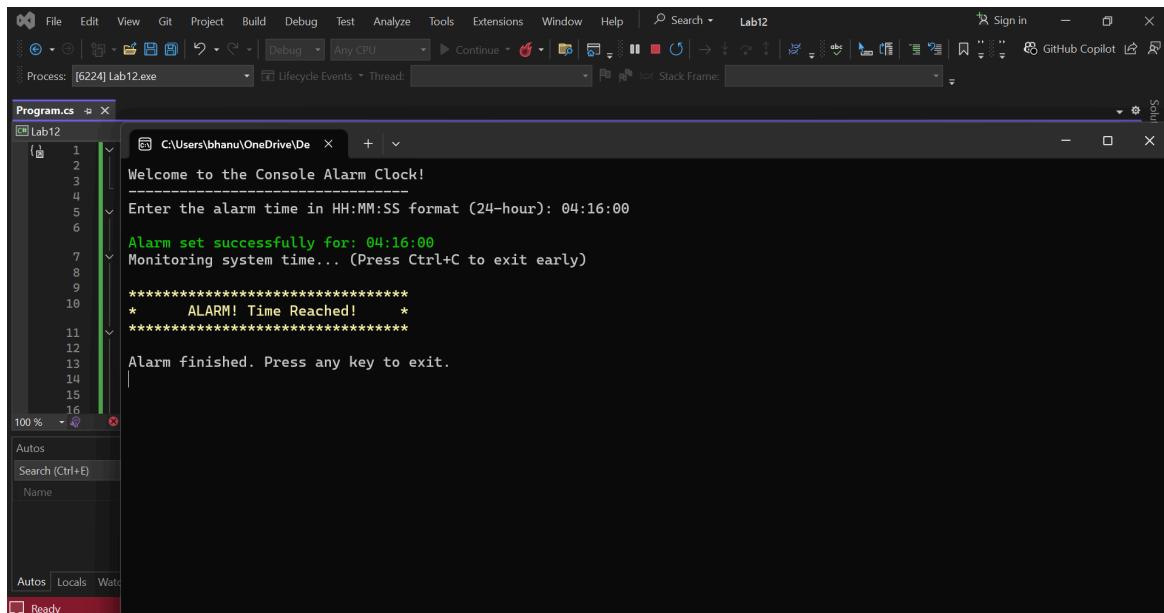
```

Process: [6224] Lab12.exe
Lifecycle Events | Thread: Stack Frame: 

```

Then, I implemented the time-checking loop by starting an infinite `while(true)` loop. Inside the loop, I retrieved the current system time's `TimeOfDay`. To ensure accurate comparison only down to the second (matching the input precision), I truncated the milliseconds/ticks from the current `TimeOfDay`. I compared the truncated current time (hours, minutes, seconds) with the `alarmTime`. If

they matched, I invoked the RaiseAlarm event (RaiseAlarm?.Invoke());, which triggered the subscribed RingAlarm method, and then used break; to exit the loop.

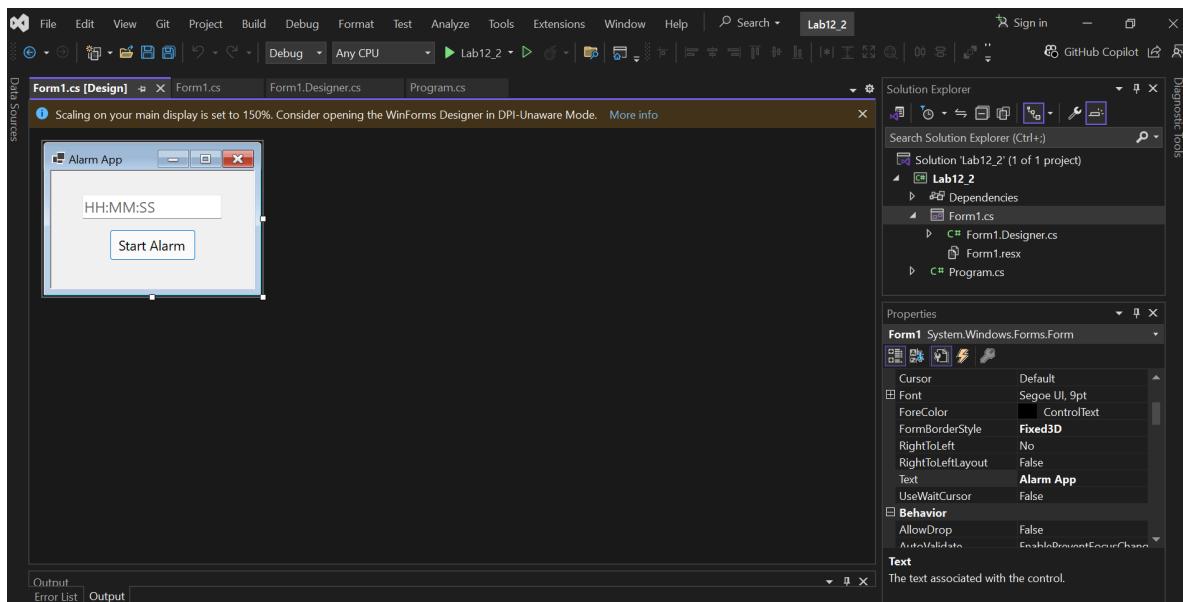


If they didn't match, I paused the execution for approximately one second using Thread.Sleep(1000) before the next iteration of the loop.

I tested various scenarios: entering invalid time formats (e.g., "10:5", "25:00:00", "abc"), entering a valid time a minute or two in the future, and waiting for the alarm to trigger to verify correct validation, event firing, message display, and loop termination.

### Step 3: Designing the User Interface (Form1)

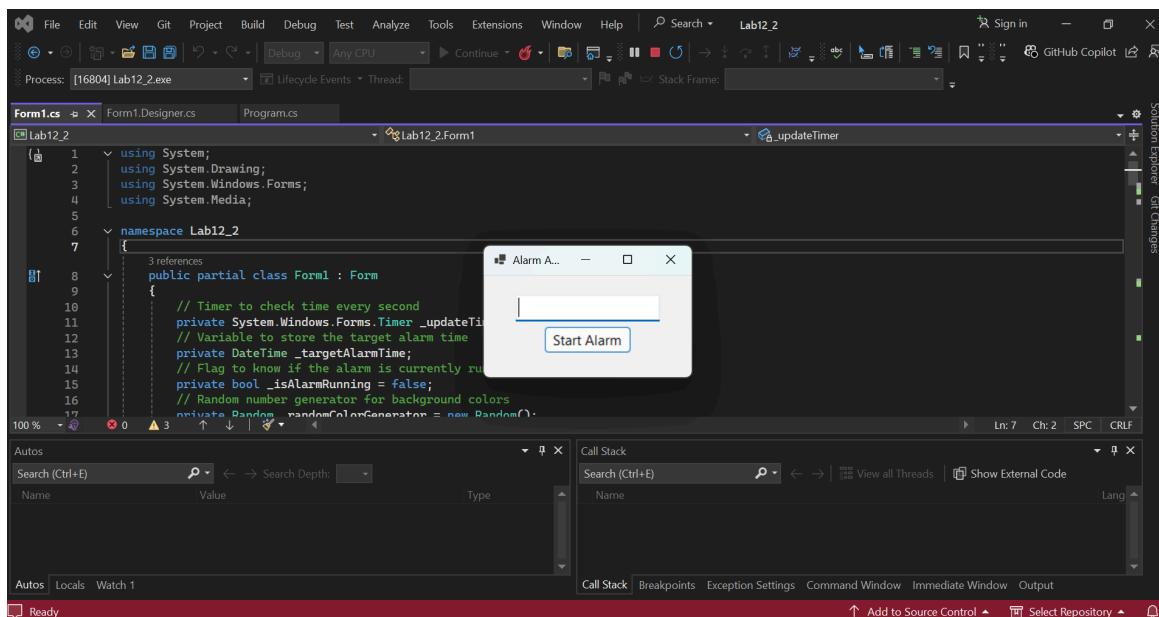
I opened Form1.cs in the Designer and added a TextBox (named txtTime) for time input and a Button (named btnStart) to initiate the alarm from the Toolbox. Then, configured control properties (PlaceholderText, Button Text, Font, Size, Location) via the Properties window for layout and usability. Set Form properties like Title (Text), FormBorderStyle, StartPosition, and AcceptButton.

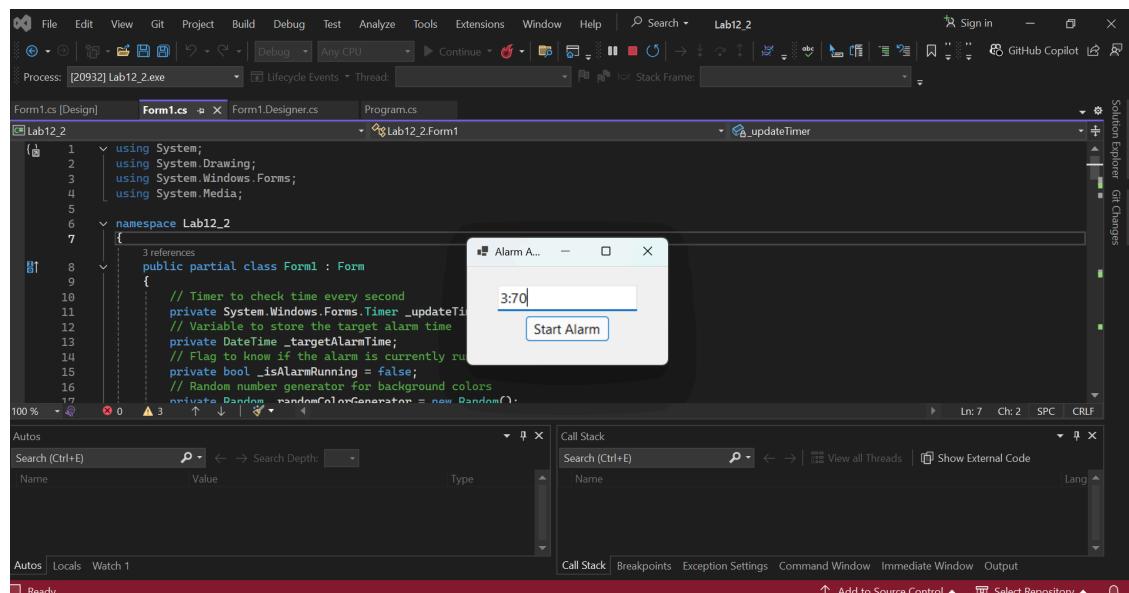
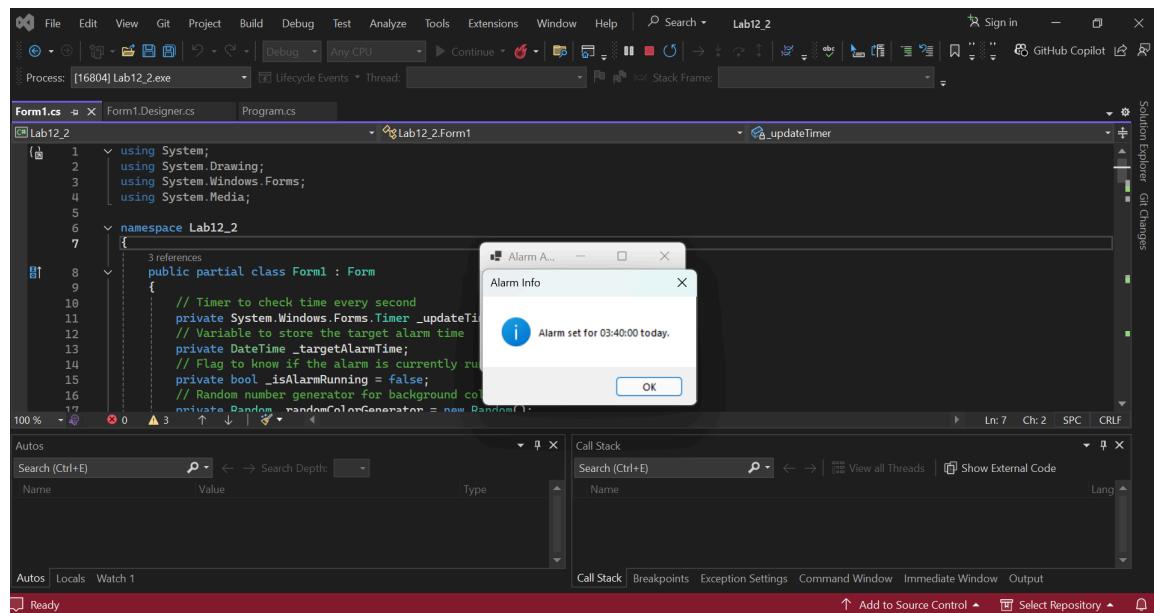


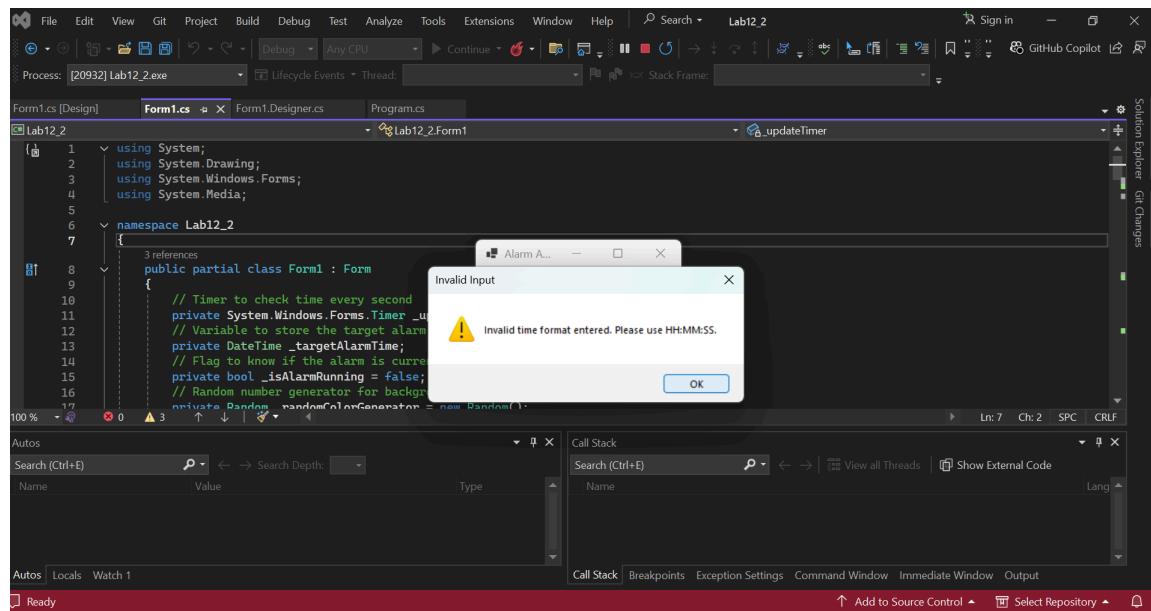
### Step 3: Implementing Core Logic (Form1.cs - Code-Behind)

After setting up the visual elements, I switched to the code view of Form1.cs. First, I declared the necessary class-level variables to manage the application's state, such as `_updateTimer`, `_targetAlarmTime`, and `_isAlarmRunning`.

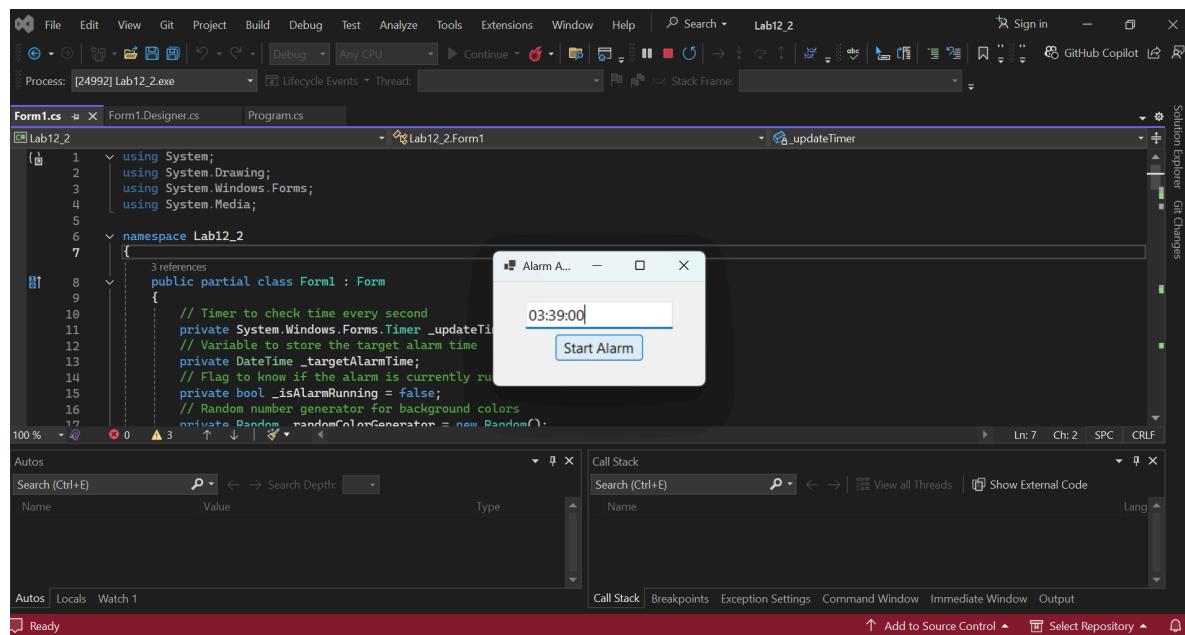
Then, I generated the `btnStart_Click` event handler by double-clicking the start button in the designer. Inside this `btnStart_Click` method, I implemented the logic to parse the time input from the `txtTime` textbox using `TimeSpan.TryParse`, making sure to handle any invalid input by showing a `MessageBox`.

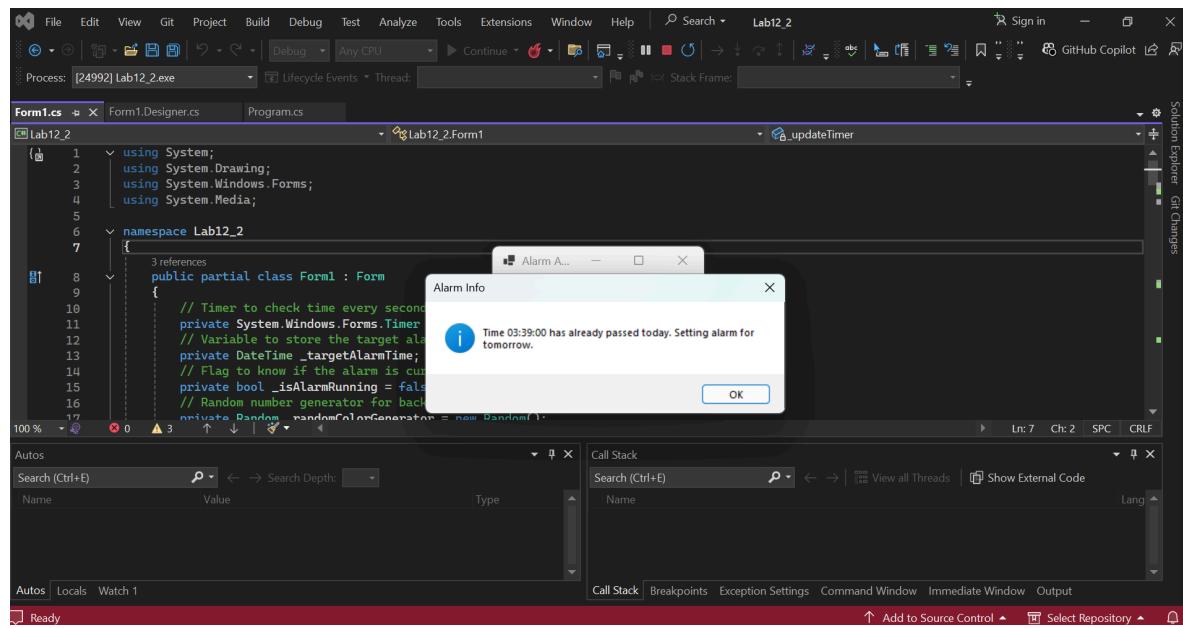






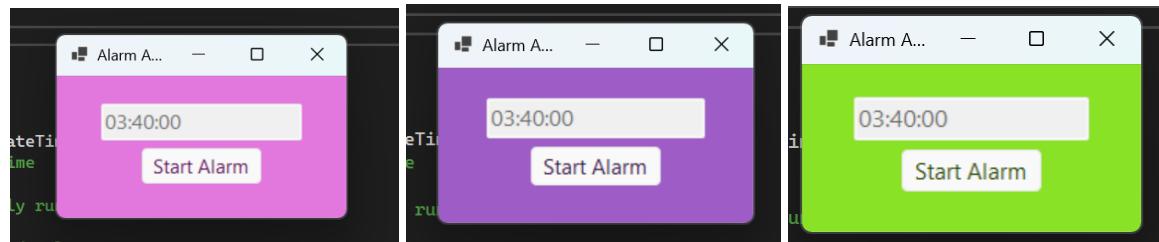
If the input was valid, I calculated the target alarm time (`_targetAlarmTime`), carefully checking if the time had already passed for today and setting it for tomorrow if needed, before showing a confirmation MessageBox.



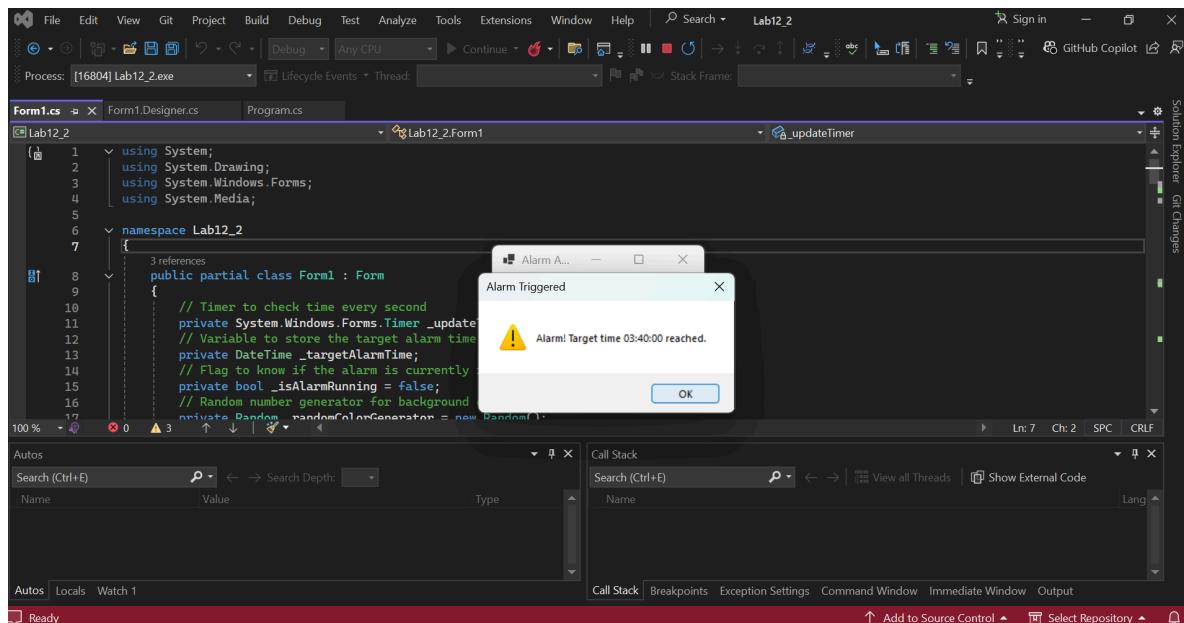


Next, I initialized and started the `_updateTimer`, set the `_isAlarmRunning` flag to true, and disabled the input controls. Following that, I created the `UpdateTimer_Tick` event handler method and assigned it to the timer's `Tick` event.

In this method, which runs every second, I checked the current time against the target: if the time hadn't been reached, I changed the form's `BackColor` to a random color.



Once the time was reached, I stopped the timer, reset the running flag, restored the original background color, showed the final "Alarm Triggered" MessageBox, re-enabled the controls, and disposed of the timer.



Lastly, I verified that Program.cs contained the correct application initialization code for the .NET framework I was using and confirmed the Application.Run(new Form1()); call was present to launch the form.

Tested various scenarios: invalid input, valid future time, time already passed today, attempting to restart while running, and reaching the alarm time to verify correct behavior, color changes, messages, and control state changes.

---

### 3. Results and Analysis

The project successfully resulted in the development of two distinct C# applications fulfilling the specified alarm clock requirements:

**Task 1 (Console Application):** A functional console application was produced. It correctly prompts the user for a time in HH:MM:SS format, validates the input using regular expressions, and continuously monitors the system time. Upon matching the target time, it successfully raises a custom event (RaiseAlarm). This event triggers the subscribed RingAlarm method, which displays the specified alarm message in the console.

**Task 2 (Windows Forms Application):** A functional GUI application using Windows Forms was developed. The application presents a user-friendly interface with a TextBox for time input and a Button to initiate the alarm. Input validation (TimeSpan.TryParse) occurs upon button click. When started, the application correctly disables input controls and utilizes a System.Windows.Forms.Timer to trigger an event every second. This event handler changes the form's background color randomly, providing continuous visual feedback. The timer's event handler also checks the current time against the target time. Upon reaching the target, the timer stops, the background color reverts to its original state, controls are re-enabled, and a confirmation

MessageBox is displayed. It also handles the edge case of the entered time having already passed for the current day.

---

## 4. Discussion and Conclusion

### Challenges and Reflections

- Initially, in Task 2, configuring the WinForms project correctly involved resolving compiler errors related to missing method definitions (btnStart\_Click) and ensuring the Form1 class properly inherited from System.Windows.Forms.Form. Differences between .NET Framework and modern .NET (.NET 6+) initialization (ApplicationConfiguration) also required clarification.
- Implementing accurate time comparison, especially handling the case where the target time might have already passed today and needed to be set for the next day, required careful logic.
- Ensuring the GUI remained responsive in Task 2 necessitated understanding the difference between Thread.Sleep (unsuitable for GUI) and the System.Windows.Forms.Timer. Correctly implementing the event wiring (both custom events in Task 1 and control/timer events in Task 2) was also a key learning point.

### Lessons Learned

The process highlighted the importance of understanding the underlying framework and programming paradigm (console vs. event-driven GUI). Debugging compile-time errors, particularly in the WinForms designer interaction, emphasized the need for careful code structure and understanding how partial classes work. The transition from the direct execution flow of the console app to the asynchronous, event-based nature of the WinForms app was a significant conceptual step. It became clear how crucial appropriate background processing techniques (like the WinForms Timer) are for creating usable GUI applications.

### Summary

This project involved the development of two C# alarm clock applications using different approaches: a console application and a Windows Forms GUI application. Both applications successfully accepted user-defined time input, monitored system time, and triggered an alarm notification upon reaching the target time. Key C# concepts including event handling (custom events and control events), time manipulation (DateTime, TimeSpan), input validation (Regex, TryParse), and timing mechanisms (Thread.Sleep, System.Windows.Forms.Timer) were utilized. The project effectively demonstrated the implementation of time-based, event-driven logic in both console and graphical environments, meeting all specified objectives.

---

## 5. References

- Lecture 11 Slides
- <https://learn.microsoft.com/en-us/dotnet/csharp>

- [Google ai studio](#)