

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
    └── bin/           # Compiled output folder
        └── obj/         # Intermediate build files folder

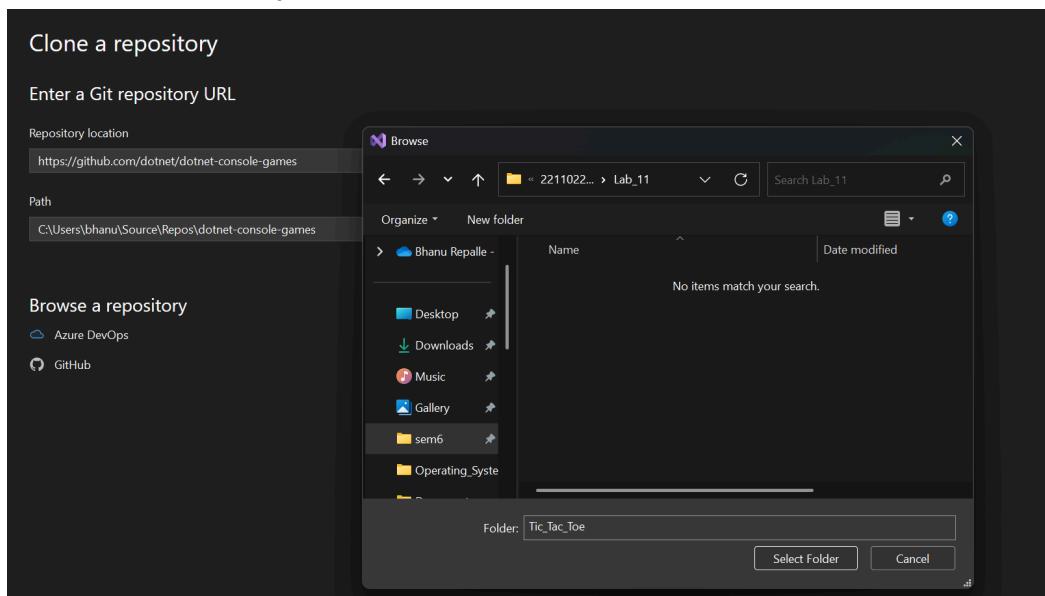
    └── Tic Tac Toe/   # Project folder for Tic Tac Toe
        ├── Program.cs    # Game logic and entry point
        ├── README.md      # Specific game info (optional)
        ├── Tic Tac Toe.csproj # Project build settings
        └── bin/           # Compiled output folder
            └── obj/         # Intermediate build files folder

```

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.

The image shows two side-by-side code editors in Visual Studio. The left editor shows the main loop of the game, and the right editor shows the input handling code. Breakpoints are visible in both sections. The game is currently running, as indicated by the console output and the state of the code.

```

58     }
59     }
60     Console.CursorVisible = true;
61 
62     void PlayerTurn()
63     {
64         var (row, column) = (0, 0);
65         bool moved = false;
66         while (!moved && !closeRequested)
67         {
68             Console.Clear();
69             RenderBoard();
70             Console.WriteLine(" Use the arrow and enter keys to select");
71             Console.SetCursorPosition(column * 4 + 4, row * 2 + 4);
72             Console.CursorVisible = true;
73             await Task.Delay(100);
74         }
75     }
76 
77     void ComputerTurn()
78     {
79         var (row, column) = (0, 0);
80         bool moved = false;
81         while (!moved && !closeRequested)
82         {
83             Console.Clear();
84             RenderBoard();
85             Console.WriteLine(" Use the arrow and enter keys to select");
86             Console.SetCursorPosition(column * 4 + 4, row * 2 + 4);
87             Console.CursorVisible = true;
88             await Task.Delay(100);
89         }
90     }
91 
92     void RenderBoard()
93     {
94         var board = new char[16];
95         for (int i = 0; i < 16; i++)
96         {
97             if (i % 4 == 0)
98             {
99                 board[i] = '|';
100            }
101        }
102        for (int i = 0; i < 16; i++)
103        {
104            if (i % 2 == 0)
105            {
106                board[i] = ' ';
107            }
108        }
109        for (int i = 0; i < 16; i++)
110        {
111            if (i % 4 == 0)
112            {
113                board[i] = ' ';
114            }
115        }
116        for (int i = 0; i < 16; i++)
117        {
118            if (i % 4 == 0)
119            {
120                board[i] = ' ';
121            }
122        }
123        for (int i = 0; i < 16; i++)
124        {
125            if (i % 4 == 0)
126            {
127                board[i] = ' ';
128            }
129        }
130        for (int i = 0; i < 16; i++)
131        {
132            if (i % 4 == 0)
133            {
134                board[i] = ' ';
135            }
136        }
137        for (int i = 0; i < 16; i++)
138        {
139            if (i % 4 == 0)
140            {
141                board[i] = ' ';
142            }
143        }
144        for (int i = 0; i < 16; i++)
145        {
146            if (i % 4 == 0)
147            {
148                board[i] = ' ';
149            }
150        }
151        for (int i = 0; i < 16; i++)
152        {
153            if (i % 4 == 0)
154            {
155                board[i] = ' ';
156            }
157        }
158        for (int i = 0; i < 16; i++)
159        {
160            if (i % 4 == 0)
161            {
162                board[i] = ' ';
163            }
164        }
165        for (int i = 0; i < 16; i++)
166        {
167            if (i % 4 == 0)
168            {
169                board[i] = ' ';
170            }
171        }
172        for (int i = 0; i < 16; i++)
173        {
174            if (i % 4 == 0)
175            {
176                board[i] = ' ';
177            }
178        }
179        for (int i = 0; i < 16; i++)
180        {
181            if (i % 4 == 0)
182            {
183                board[i] = ' ';
184            }
185        }
186        for (int i = 0; i < 16; i++)
187        {
188            if (i % 4 == 0)
189            {
190                board[i] = ' ';
191            }
192        }
193        for (int i = 0; i < 16; i++)
194        {
195            if (i % 4 == 0)
196            {
197                board[i] = ' ';
198            }
199        }
200        for (int i = 0; i < 16; i++)
201        {
202            if (i % 4 == 0)
203            {
204                board[i] = ' ';
205            }
206        }
207        for (int i = 0; i < 16; i++)
208        {
209            if (i % 4 == 0)
210            {
211                board[i] = ' ';
212            }
213        }
214        for (int i = 0; i < 16; i++)
215        {
216            if (i % 4 == 0)
217            {
218                board[i] = ' ';
219            }
220        }
221        for (int i = 0; i < 16; i++)
222        {
223            if (i % 4 == 0)
224            {
225                board[i] = ' ';
226            }
227        }
228        for (int i = 0; i < 16; i++)
229        {
230            if (i % 4 == 0)
231            {
232                board[i] = ' ';
233            }
234        }
235        for (int i = 0; i < 16; i++)
236        {
237            if (i % 4 == 0)
238            {
239                board[i] = ' ';
240            }
241        }
242        for (int i = 0; i < 16; i++)
243        {
244            if (i % 4 == 0)
245            {
246                board[i] = ' ';
247            }
248        }
249        for (int i = 0; i < 16; i++)
250        {
251            if (i % 4 == 0)
252            {
253                board[i] = ' ';
254            }
255        }
256        for (int i = 0; i < 16; i++)
257        {
258            if (i % 4 == 0)
259            {
260                board[i] = ' ';
261            }
262        }
263        for (int i = 0; i < 16; i++)
264        {
265            if (i % 4 == 0)
266            {
267                board[i] = ' ';
268            }
269        }
270        for (int i = 0; i < 16; i++)
271        {
272            if (i % 4 == 0)
273            {
274                board[i] = ' ';
275            }
276        }
277        for (int i = 0; i < 16; i++)
278        {
279            if (i % 4 == 0)
280            {
281                board[i] = ' ';
282            }
283        }
284        for (int i = 0; i < 16; i++)
285        {
286            if (i % 4 == 0)
287            {
288                board[i] = ' ';
289            }
290        }
291        for (int i = 0; i < 16; i++)
292        {
293            if (i % 4 == 0)
294            {
295                board[i] = ' ';
296            }
297        }
298        for (int i = 0; i < 16; i++)
299        {
300            if (i % 4 == 0)
301            {
302                board[i] = ' ';
303            }
304        }
305        for (int i = 0; i < 16; i++)
306        {
307            if (i % 4 == 0)
308            {
309                board[i] = ' ';
310            }
311        }
312        for (int i = 0; i < 16; i++)
313        {
314            if (i % 4 == 0)
315            {
316                board[i] = ' ';
317            }
318        }
319        for (int i = 0; i < 16; i++)
320        {
321            if (i % 4 == 0)
322            {
323                board[i] = ' ';
324            }
325        }
326        for (int i = 0; i < 16; i++)
327        {
328            if (i % 4 == 0)
329            {
330                board[i] = ' ';
331            }
332        }
333        for (int i = 0; i < 16; i++)
334        {
335            if (i % 4 == 0)
336            {
337                board[i] = ' ';
338            }
339        }
340        for (int i = 0; i < 16; i++)
341        {
342            if (i % 4 == 0)
343            {
344                board[i] = ' ';
345            }
346        }
347        for (int i = 0; i < 16; i++)
348        {
349            if (i % 4 == 0)
350            {
351                board[i] = ' ';
352            }
353        }
354        for (int i = 0; i < 16; i++)
355        {
356            if (i % 4 == 0)
357            {
358                board[i] = ' ';
359            }
360        }
361        for (int i = 0; i < 16; i++)
362        {
363            if (i % 4 == 0)
364            {
365                board[i] = ' ';
366            }
367        }
368        for (int i = 0; i < 16; i++)
369        {
370            if (i % 4 == 0)
371            {
372                board[i] = ' ';
373            }
374        }
375        for (int i = 0; i < 16; i++)
376        {
377            if (i % 4 == 0)
378            {
379                board[i] = ' ';
380            }
381        }
382        for (int i = 0; i < 16; i++)
383        {
384            if (i % 4 == 0)
385            {
386                board[i] = ' ';
387            }
388        }
389        for (int i = 0; i < 16; i++)
390        {
391            if (i % 4 == 0)
392            {
393                board[i] = ' ';
394            }
395        }
396        for (int i = 0; i < 16; i++)
397        {
398            if (i % 4 == 0)
399            {
400                board[i] = ' ';
401            }
402        }
403        for (int i = 0; i < 16; i++)
404        {
405            if (i % 4 == 0)
406            {
407                board[i] = ' ';
408            }
409        }
410        for (int i = 0; i < 16; i++)
411        {
412            if (i % 4 == 0)
413            {
414                board[i] = ' ';
415            }
416        }
417        for (int i = 0; i < 16; i++)
418        {
419            if (i % 4 == 0)
420            {
421                board[i] = ' ';
422            }
423        }
424        for (int i = 0; i < 16; i++)
425        {
426            if (i % 4 == 0)
427            {
428                board[i] = ' ';
429            }
430        }
431        for (int i = 0; i < 16; i++)
432        {
433            if (i % 4 == 0)
434            {
435                board[i] = ' ';
436            }
437        }
438        for (int i = 0; i < 16; i++)
439        {
440            if (i % 4 == 0)
441            {
442                board[i] = ' ';
443            }
444        }
445        for (int i = 0; i < 16; i++)
446        {
447            if (i % 4 == 0)
448            {
449                board[i] = ' ';
450            }
451        }
452        for (int i = 0; i < 16; i++)
453        {
454            if (i % 4 == 0)
455            {
456                board[i] = ' ';
457            }
458        }
459        for (int i = 0; i < 16; i++)
460        {
461            if (i % 4 == 0)
462            {
463                board[i] = ' ';
464            }
465        }
466        for (int i = 0; i < 16; i++)
467        {
468            if (i % 4 == 0)
469            {
470                board[i] = ' ';
471            }
472        }
473        for (int i = 0; i < 16; i++)
474        {
475            if (i % 4 == 0)
476            {
477                board[i] = ' ';
478            }
479        }
480        for (int i = 0; i < 16; i++)
481        {
482            if (i % 4 == 0)
483            {
484                board[i] = ' ';
485            }
486        }
487        for (int i = 0; i < 16; i++)
488        {
489            if (i % 4 == 0)
490            {
491                board[i] = ' ';
492            }
493        }
494        for (int i = 0; i < 16; i++)
495        {
496            if (i % 4 == 0)
497            {
498                board[i] = ' ';
499            }
500        }
501        for (int i = 0; i < 16; i++)
502        {
503            if (i % 4 == 0)
504            {
505                board[i] = ' ';
506            }
507        }
508        for (int i = 0; i < 16; i++)
509        {
510            if (i % 4 == 0)
511            {
512                board[i] = ' ';
513            }
514        }
515        for (int i = 0; i < 16; i++)
516        {
517            if (i % 4 == 0)
518            {
519                board[i] = ' ';
520            }
521        }
522        for (int i = 0; i < 16; i++)
523        {
524            if (i % 4 == 0)
525            {
526                board[i] = ' ';
527            }
528        }
529        for (int i = 0; i < 16; i++)
530        {
531            if (i % 4 == 0)
532            {
533                board[i] = ' ';
534            }
535        }
536        for (int i = 0; i < 16; i++)
537        {
538            if (i % 4 == 0)
539            {
540                board[i] = ' ';
541            }
542        }
543        for (int i = 0; i < 16; i++)
544        {
545            if (i % 4 == 0)
546            {
547                board[i] = ' ';
548            }
549        }
550        for (int i = 0; i < 16; i++)
551        {
552            if (i % 4 == 0)
553            {
554                board[i] = ' ';
555            }
556        }
557        for (int i = 0; i < 16; i++)
558        {
559            if (i % 4 == 0)
560            {
561                board[i] = ' ';
562            }
563        }
564        for (int i = 0; i < 16; i++)
565        {
566            if (i % 4 == 0)
567            {
568                board[i] = ' ';
569            }
570        }
571        for (int i = 0; i < 16; i++)
572        {
573            if (i % 4 == 0)
574            {
575                board[i] = ' ';
576            }
577        }
578        for (int i = 0; i < 16; i++)
579        {
580            if (i % 4 == 0)
581            {
582                board[i] = ' ';
583            }
584        }
585        for (int i = 0; i < 16; i++)
586        {
587            if (i % 4 == 0)
588            {
589                board[i] = ' ';
590            }
591        }
592        for (int i = 0; i < 16; i++)
593        {
594            if (i % 4 == 0)
595            {
596                board[i] = ' ';
597            }
598        }
599        for (int i = 0; i < 16; i++)
600        {
601            if (i % 4 == 0)
602            {
603                board[i] = ' ';
604            }
605        }
606        for (int i = 0; i < 16; i++)
607        {
608            if (i % 4 == 0)
609            {
610                board[i] = ' ';
611            }
612        }
613        for (int i = 0; i < 16; i++)
614        {
615            if (i % 4 == 0)
616            {
617                board[i] = ' ';
618            }
619        }
620        for (int i = 0; i < 16; i++)
621        {
622            if (i % 4 == 0)
623            {
624                board[i] = ' ';
625            }
626        }
627        for (int i = 0; i < 16; i++)
628        {
629            if (i % 4 == 0)
630            {
631                board[i] = ' ';
632            }
633        }
634        for (int i = 0; i < 16; i++)
635        {
636            if (i % 4 == 0)
637            {
638                board[i] = ' ';
639            }
640        }
641        for (int i = 0; i < 16; i++)
642        {
643            if (i % 4 == 0)
644            {
645                board[i] = ' ';
646            }
647        }
648        for (int i = 0; i < 16; i++)
649        {
650            if (i % 4 == 0)
651            {
652                board[i] = ' ';
653            }
654        }
655        for (int i = 0; i < 16; i++)
656        {
657            if (i % 4 == 0)
658            {
659                board[i] = ' ';
660            }
661        }
662        for (int i = 0; i < 16; i++)
663        {
664            if (i % 4 == 0)
665            {
666                board[i] = ' ';
667            }
668        }
669        for (int i = 0; i < 16; i++)
670        {
671            if (i % 4 == 0)
672            {
673                board[i] = ' ';
674            }
675        }
676        for (int i = 0; i < 16; i++)
677        {
678            if (i % 4 == 0)
679            {
680                board[i] = ' ';
681            }
682        }
683        for (int i = 0; i < 16; i++)
684        {
685            if (i % 4 == 0)
686            {
687                board[i] = ' ';
688            }
689        }
690        for (int i = 0; i < 16; i++)
691        {
692            if (i % 4 == 0)
693            {
694                board[i] = ' ';
695            }
696        }
697        for (int i = 0; i < 16; i++)
698        {
699            if (i % 4 == 0)
700            {
701                board[i] = ' ';
702            }
703        }
704        for (int i = 0; i < 16; i++)
705        {
706            if (i % 4 == 0)
707            {
708                board[i] = ' ';
709            }
710        }
711        for (int i = 0; i < 16; i++)
712        {
713            if (i % 4 == 0)
714            {
715                board[i] = ' ';
716            }
717        }
718        for (int i = 0; i < 16; i++)
719        {
720            if (i % 4 == 0)
721            {
722                board[i] = ' ';
723            }
724        }
725        for (int i = 0; i < 16; i++)
726        {
727            if (i % 4 == 0)
728            {
729                board[i] = ' ';
730            }
731        }
732        for (int i = 0; i < 16; i++)
733        {
734            if (i % 4 == 0)
735            {
736                board[i] = ' ';
737            }
738        }
739        for (int i = 0; i < 16; i++)
740        {
741            if (i % 4 == 0)
742            {
743                board[i] = ' ';
744            }
745        }
746        for (int i = 0; i < 16; i++)
747        {
748            if (i % 4 == 0)
749            {
750                board[i] = ' ';
751            }
752        }
753        for (int i = 0; i < 16; i++)
754        {
755            if (i % 4 == 0)
756            {
757                board[i] = ' ';
758            }
759        }
760        for (int i = 0; i < 16; i++)
761        {
762            if (i % 4 == 0)
763            {
764                board[i] = ' ';
765            }
766        }
767        for (int i = 0; i < 16; i++)
768        {
769            if (i % 4 == 0)
770            {
771                board[i] = ' ';
772            }
773        }
774        for (int i = 0; i < 16; i++)
775        {
776            if (i % 4 == 0)
777            {
778                board[i] = ' ';
779            }
780        }
781        for (int i = 0; i < 16; i++)
782        {
783            if (i % 4 == 0)
784            {
785                board[i] = ' ';
786            }
787        }
788        for (int i = 0; i < 16; i++)
789        {
790            if (i % 4 == 0)
791            {
792                board[i] = ' ';
793            }
794        }
795        for (int i = 0; i < 16; i++)
796        {
797            if (i % 4 == 0)
798            {
799                board[i] = ' ';
800            }
801        }
802        for (int i = 0; i < 16; i++)
803        {
804            if (i % 4 == 0)
805            {
806                board[i] = ' ';
807            }
808        }
809        for (int i = 0; i < 16; i++)
810        {
811            if (i % 4 == 0)
812            {
813                board[i] = ' ';
814            }
815        }
816        for (int i = 0; i < 16; i++)
817        {
818            if (i % 4 == 0)
819            {
820                board[i] = ' ';
821            }
822        }
823        for (int i = 0; i < 16; i++)
824        {
825            if (i % 4 == 0)
826            {
827                board[i] = ' ';
828            }
829        }
830        for (int i = 0; i < 16; i++)
831        {
832            if (i % 4 == 0)
833            {
834                board[i] = ' ';
835            }
836        }
837        for (int i = 0; i < 16; i++)
838        {
839            if (i % 4 == 0)
840            {
841                board[i] = ' ';
842            }
843        }
844        for (int i = 0; i < 16; i++)
845        {
846            if (i % 4 == 0)
847            {
848                board[i] = ' ';
849            }
850        }
851        for (int i = 0; i < 16; i++)
852        {
853            if (i % 4 == 0)
854            {
855                board[i] = ' ';
856            }
857        }
858        for (int i = 0; i < 16; i++)
859        {
860            if (i % 4 == 0)
861            {
862                board[i] = ' ';
863            }
864        }
865        for (int i = 0; i < 16; i++)
866        {
867            if (i % 4 == 0)
868            {
869                board[i] = ' ';
870            }
871        }
872        for (int i = 0; i < 16; i++)
873        {
874            if (i % 4 == 0)
875            {
876                board[i] = ' ';
877            }
878        }
879        for (int i = 0; i < 16; i++)
880        {
881            if (i % 4 == 0)
882            {
883                board[i] = ' ';
884            }
885        }
886        for (int i = 0; i < 16; i++)
887        {
888            if (i % 4 == 0)
889            {
890                board[i] = ' ';
891            }
892        }
893        for (int i = 0; i < 16; i++)
894        {
895            if (i % 4 == 0)
896            {
897                board[i] = ' ';
898            }
899        }
900        for (int i = 0; i < 16; i++)
901        {
902            if (i % 4 == 0)
903            {
904                board[i] = ' ';
905            }
906        }
907        for (int i = 0; i < 16; i++)
908        {
909            if (i % 4 == 0)
910            {
911                board[i] = ' ';
912            }
913        }
914        for (int i = 0; i < 16; i++)
915        {
916            if (i % 4 == 0)
917            {
918                board[i] = ' ';
919            }
920        }
921        for (int i = 0; i < 16; i++)
922        {
923            if (i % 4 == 0)
924            {
925                board[i] = ' ';
926            }
927        }
928        for (int i = 0; i < 16; i++)
929        {
930            if (i % 4 == 0)
931            {
932                board[i] = ' ';
933            }
934        }
935        for (int i = 0; i < 16; i++)
936        {
937            if (i % 4 == 0)
938            {
939                board[i] = ' ';
940            }
941        }
942        for (int i = 0; i < 16; i++)
943        {
944            if (i % 4 == 0)
945            {
946                board[i] = ' ';
947            }
948        }
949        for (int i = 0; i < 16; i++)
950        {
951            if (i % 4 == 0)
952            {
953                board[i] = ' ';
954            }
955        }
956        for (int i = 0; i < 16; i++)
957        {
958            if (i % 4 == 0)
959            {
960                board[i] = ' ';
961            }
962        }
963        for (int i = 0; i < 16; i++)
964        {
965            if (i % 4 == 0)
966            {
967                board[i] = ' ';
968            }
969        }
970        for (int i = 0; i < 16; i++)
971        {
972            if (i % 4 == 0)
973            {
974                board[i] = ' ';
975            }
976        }
977        for (int i = 0; i < 16; i++)
978        {
979            if (i % 4 == 0)
980            {
981                board[i] = ' ';
982            }
983        }
984        for (int i = 0; i < 16; i++)
985        {
986            if (i % 4 == 0)
987            {
988                board[i] = ' ';
989            }
990        }
991        for (int i = 0; i < 16; i++)
992        {
993            if (i % 4 == 0)
994            {
995                board[i] = ' ';
996            }
997        }
998        for (int i = 0; i < 16; i++)
999        {
1000            if (i % 4 == 0)
1001            {
1002                board[i] = ' ';
1003            }
1004        }
1005        for (int i = 0; i < 16; i++)
1006        {
1007            if (i % 4 == 0)
1008            {
1009                board[i] = ' ';
1010            }
1011        }
1012        for (int i = 0; i < 16; i++)
1013        {
1014            if (i % 4 == 0)
1015            {
1016                board[i] = ' ';
1017            }
1018        }
1019        for (int i = 0; i < 16; i++)
1020        {
1021            if (i % 4 == 0)
1022            {
1023                board[i] = ' ';
1024            }
1025        }
1026        for (int i = 0; i < 16; i++)
1027        {
1028            if (i % 4 == 0)
1029            {
1030                board[i] = ' ';
1031            }
1032        }
1033        for (int i = 0; i < 16; i++)
1034        {
1035            if (i % 4 == 0)
1036            {
1037                board[i] = ' ';
1038            }
1039        }
1040        for (int i = 0; i < 16; i++)
1041        {
1042            if (i % 4 == 0)
1043            {
1044                board[i] = ' ';
1045            }
1046        }
1047        for (int i = 0; i < 16; i++)
1048        {
1049            if (i % 4 == 0)
1050            {
1051                board[i] = ' ';
1052            }
1053        }
1054        for (int i = 0; i < 16; i++)
1055        {
1056            if (i % 4 == 0)
1057            {
1058                board[i] = ' ';
1059            }
1060        }
1061        for (int i = 0; i < 16; i++)
1062        {
1063            if (i % 4 == 0)
1064            {
1065                board[i] = ' ';
1066            }
1067        }
1068        for (int i = 0; i < 16; i++)
1069        {
1070            if (i % 4 == 0)
1071            {
1072                board[i] = ' ';
1073            }
1074        }
1075        for (int i = 0; i < 16; i++)
1076        {
1077            if (i % 4 == 0)
1078            {
1079                board[i] = ' ';
1080            }
1081        }
1082        for (int i = 0; i < 16; i++)
1083        {
1084            if (i % 4 == 0)
1085            {
1086                board[i] = ' ';
1087            }
1088        }
1089        for (int i = 0; i < 16; i++)
1090        {
1091            if (i % 4 == 0)
1092            {
1093                board[i] = ' ';
1094            }
1095        }
1096        for (int i = 0; i < 16; i++)
1097        {
1098            if (i % 4 == 0)
1099            {
1100                board[i] = ' ';
1101            }
1102        }
1103        for (int i = 0; i < 16; i++)
1104        {
1105            if (i % 4 == 0)
1106            {
1107                board[i] = ' ';
1108            }
1109        }
1110        for (int i = 0; i < 16; i++)
1111        {
1112            if (i % 4 == 0)
1113            {
1114                board[i] = ' ';
1115            }
1116        }
1117        for (int i = 0; i < 16; i++)
1118        {
1119            if (i % 4 == 0)
1120            {
1121                board[i] = ' ';
1122            }
1123        }
1124        for (int i = 0; i < 16; i++)
1125        {
1126            if (i % 4 == 0)
1127            {
1128                board[i] = ' ';
1129            }
1130        }
1131        for (int i = 0; i < 16; i++)
1132        {
1133            if (i % 4 == 0)
1134            {
1135                board[i] = ' ';
1136            }
1137        }
1138        for (int i = 0; i < 16; i++)
1139        {
1140            if (i % 4 == 0)
1141            {
1142                board[i] = ' ';
1143            }
1144        }
1145        for (int i = 0; i < 16; i++)
1146        {
1147            if (i % 4 == 0)
1148            {
1149                board[i] = ' ';
1150            }
1151        }
1152        for (int i = 0; i < 16; i++)
1153        {
1154            if (i % 4 == 0)
1155            {
1156                board[i] = ' ';
1157            }
1158        }
1159        for (int i = 0; i < 16; i++)
1160        {
1161            if (i % 4 == 0)
1162            {
1163                board[i] = ' ';
1164            }
1165        }
1166        for (int i = 0; i < 16; i++)
1167        {
1168            if (i % 4 == 0)
1169            {
1170                board[i] = ' ';
1171            }
1172        }
1173        for (int i = 0; i < 16; i++)
1174        {
1175            if (i % 4 == 0)
1176            {
1177               
```

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.

A screenshot of the Visual Studio debugger interface. The code editor shows lines 35 through 44 of a C# file. Line 36 contains the assignment `playerTurn = !playerTurn;`. A yellow highlight covers the entire line. Below the code editor is a status bar with "No issues found". Underneath is the "Autos" window, which has a search bar and displays a single entry: `playerTurn` with a value of `true` and type `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

A screenshot of the Visual Studio debugger interface. The code editor shows lines 134 through 150. Line 142 contains the call to `EndGame("")`, with a yellow highlight covering the entire line. Below the code editor is a status bar with "No issues found". Underneath is the "Autos" window, which has a search bar and displays a single entry: `System.Runtime.CompilerServices...` with a value of `"\r\n Tic Tac Toe\r\n\r\n\r\n\r\n\r\n"`. To the right of the debugger, a terminal window titled "Tic Tac Toe" shows a 3x3 grid. The top-left cell contains an "X", the bottom-left cell contains a "0", and all other cells are empty.

This is observed at the `RenderBoard()` breakpoint

```

C# Tic Tac Toe
61
62     | } PlayerTurn()
63
64     | var (row, column) = (0, 0);
65     | bool moved = 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 se
72         Console.SetCursorPosition(column * 4 + 4, row * 2 + 4);
73         Console.CursorVisible = true;
74         switch (Console.ReadKey(true).Key)

```

Debugging:

I set multiple bugs along the lines.

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.

Disassembly

```

62     <playerTurn>
63         (row, column) = (0, 0);
64         .moved = false;
65     <while (!moved && !closeRequested)>
66         Console.Clear();
67         RenderBoard();
68         Console.WriteLine();
69         Console.WriteLine(" Use the arrow and enter keys to select a move.");
70         Console.SetCursorPosition(column * 4 + 4, row * 2 + 4);
71         Console.CursorVisible = true;
72         switch (Console.ReadKey(true).Key)
73         {
74             case ConsoleKey.UpArrow:   row = row <= 0 ? 2 : row - 1;
75             case ConsoleKey.DownArrow: row = row >= 2 ? 0 : row + 1;
76             case ConsoleKey.LeftArrow: column = column <= 0 ? 2 : column - 1;
77             case ConsoleKey.RightArrow: column = column >= 2 ? 0 : column + 1;
78             case ConsoleKey.Enter:
79                 [row, column] = ' ';
80             default:
81                 [row, column] = 'X';
82             break;
83         }
84     }
85 }
```

Process: [5228] Tic Tac Toe.exe

Tic Tac Toe

Use the arrow and enter keys to select a move.

Disassembly

```

71     <playerTurn>
72         " Use the arrow and enter keys to select a move.";
73         position(column * 4 + 4, row * 2 + 4);
74         .ble = true;
75     <while (!key(true).Key)
76         y.UpArrow:   row = row <= 0 ? 2 : row - 1; break;
77         y.DownArrow: row = row >= 2 ? 0 : row + 1; break;
78         y.LeftArrow: column = column <= 0 ? 2 : column - 1; break;
79         y.RightArrow: column = column >= 2 ? 0 : column + 1; break;
80         y.Enter:
81         [row, column] = ' ';
```

Process: [5228] Tic Tac Toe.exe

Tic Tac Toe

Use the arrow and enter keys to

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()`

```

<void PlayerTurn()
{
    var (row, column) = (0, 0);
    bool moved = false;
    <while (!moved && !closeRequested)>
    {
        [row, column] = ' ';
        moved = true;
    }
}
```

```

<void PlayerTurn()
{
    var (row, column) = (0, 0);
    bool moved = false;
    <while (!moved || !closeRequested)>
    {
        [row, column] = ' ';
        moved = true;
    }
}
```

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

```

✓ 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));
        }
    }
}

```

3. Changes winning logic by switching 2 with 3

```

112
113     ✓ bool CheckForThree(char c) =>
114         board[0, 0] == c && board[1, 0] == c && board[2, 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
120
121     ✓ bool CheckForThree(char c) =>
122         board[0, 0] == c && board[1, 0] == c && board[3, 0] == c || 
123         board[0, 1] == c && board[1, 1] == c && board[2, 1] == c || 
124         board[0, 2] == c && board[1, 2] == c && board[2, 2] == c || 
125         board[0, 0] == c && board[0, 1] == c && board[0, 2] == c || 
126         board[1, 0] == c && board[1, 1] == c && board[1, 2] == c || 
127         board[2, 0] == c && board[2, 1] == c && board[2, 2] == c || 
128         board[3, 0] == c && board[3, 1] == c && board[3, 2] == c || 
129
130
131     ✓ void ComputerTurn()
132     {
133         var possibleMoves = new List<(int X, int Y)>;
134         for (int i = 0; i < 3; i++)
135         {
136             for (int j = 0; j < 3; j++)
137             {
138                 if (board[i, j] != ' ')
139                     possibleMoves.Add((i, j));
140             }
141         }
142     }
}

```

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

```

Step 3: Debugging Tic Tac Toe

This screenshot shows the Visual Studio interface during the debugging process of a Tic Tac Toe application. The main window displays the source code of `Program.cs`. A break point is set at line 10, where a local variable `board` is being initialized. The Autos window shows the current values of variables: `board` is null (char[]) and `closeRequested` is false (bool). The Call Stack window shows the current thread is running in `Tic Tac Toe.dll!Program.<Main>$({string[]} a... C#)`.

```
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        while (closeRequested)
17        {
18            if (playerTurn)
19            {
20                ...
21            }
22        }
23    }
24 }
```

Name	Value	Type
board	null	char[]
closeRe...	false	bool

Name	Lang
Tic Tac Toe.dll!Program.<Main>\$({string[]} a... C#)	

This screenshot continues the debugging session of the Tic Tac Toe application. The code execution has moved to line 45, where the application outputs a draw message and asks the user if they want to play again or quit. The Autos window now shows that `closeRequested` is true (bool). The Call Stack window remains the same.

```
39         EndGame(" Draw.");
40         break;
41     }
42 }
43 if (!closeRequested)
44 {
45     Console.WriteLine();
46     Console.WriteLine(" Play Again [enter], or quit [es");
47     GetInput();
48     Console.CursorVisible = false;
49     switch (Console.ReadKey(true).Key)
50     {
51         case ConsoleKey.Enter: break;
52         case ConsoleKey.Escape:
53             closeRequested = true;
54     }
55 }
```

Name	Value	Type
closeRe...	false	bool

Name	Lang
Tic Tac Toe.dll!Program.<Main>\$({string[]} a... C#)	

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

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 interface with the 'Lab_11' project open. The 'Program.cs' tab is selected, displaying the same code as before:

```
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                     break;
25                 }
26             }
27         }
28     }
29 }
```

The 'Call Stack' and 'Search (Ctrl+E)' windows are also visible. The status bar at the bottom indicates 'Ln: 16 Ch: 2 Col: 5 TABS CRLF'.

After correcting, it entered the loop as expected.

The screenshot shows the Visual Studio interface with the 'Lab_11' project open. The 'Program.cs' tab is selected, displaying the corrected code:

```
13     {
14         {
15             {
16         };
17     while (!closeRequested)
18     {
19         if (playerTurn)
20         {
21             PlayerTurn();
22             if (CheckForThree('X'))
23             {
24                 EndGame(" You Win.");
25                 break;
26             }
27         }
28     }
29 }
```

The 'Call Stack' and 'Search (Ctrl+E)' windows are also visible. The status bar at the bottom indicates 'Ln: 17 Ch: 2 Col: 5 TABS CRLF'.

Bug 2:

After fixing the bug, I went to play another round and found that the computer was not playing any input.

The screenshot shows the Visual Studio interface with the project "Lab_11" open. The code editor displays `Program.cs` with the following code:

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

The Solution Explorer and Git Changes panes are visible on the right. In the background, a terminal window shows the output of the program: a 3x3 grid with 'X' in the top-left and top-right positions. A message at the bottom of the terminal says "Use the arrow and enter keys to select a move."

The screenshot shows the Visual Studio interface with the project "Lab_11" open. The code editor displays `Program.cs` with the following code:

```
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    }
```

The Solution Explorer and Git Changes panes are visible on the right. In the background, a terminal window shows the output of the program: a 3x3 grid with 'X' in the top-left, middle-right, and bottom-left positions. A message at the bottom of the terminal says "Use the arrow and enter keys to select a move." The Autos and Locals toolbars are open at the bottom of the interface.

The screenshot shows the Visual Studio IDE with the project "Lab_11" open. The main window displays the code for "Program.cs". A bug is present in the `PlayerTurn()` method where the condition `while (!moved || !closeRequested)` is always true, causing an infinite loop. The Autos window shows that `playerTurn` is set to `true`, which is the reason for the loop. The Solution Explorer and Call Stack windows are also visible.

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 main window displays the code for "Program.cs". The bug has been fixed by changing the condition in the `PlayerTurn()` method from `while (!moved || !closeRequested)` to `while (!moved && !closeRequested)`. The Autos window now shows that `playerTurn` is `false`, indicating the loop is no longer infinite. The Solution Explorer and Call Stack windows are also visible.

Bug 3:

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

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes File, Edit, View, Git, Project, Debug, Test, Analyze, Tools, Extensions, Window, and Help. The title bar says "Lab_11". The status bar at the bottom indicates "No issues found".

The code editor displays the following C# code:

```
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.");
24               }
25           }
26           else
27           {
28               ComputerTurn();
29               if (CheckForThree('X'))
30               {
31                   EndGame(" You Lose.");
32               }
33           }
34       }
35   }
```

The output window on the right shows a 3x3 grid with the first row filled by 'O' and the second row filled by 'X'. It also contains the instruction: "Use the arrow and enter keys to select a move."

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.

The code editor shows the modified C# code. The winning condition for 'O' has been added to the 'else' block of the player's turn loop.

```
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.");
24                   break;
25               }
26           }
27           else
28           {
29               ComputerTurn();
30               if (CheckForThree('X'))
31               {
32                   EndGame(" You Lose.");
33               }
34           }
35       }
36   }
```

Bug 4:

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

The screenshot shows the Microsoft Visual Studio interface during a debugging session. The title bar says "Lab_11". The main window displays the code for "Program.cs" in the "Tic Tac Toe" project. A break point is set at line 109. The code is as follows:

```
95     void ComputerTurn()
96     {
97         var possibleMoves = new List<int X, int Y>();
98         for (int i = 0; i < 3; i++)
99         {
100            for (int j = 0; j < 3; j++)
101            {
102                if (board[i, j] != ' ')
103                {
104                    possibleMoves.Add((i, j));
105                }
106            }
107        }
108        int index = Random.Shared.Next(0, possibleMoves.Count);
109        var (X, Y) = possibleMoves[index];
110        board[X, Y] = '0';
111    }
```

The cursor is at the line "var (X, Y) = possibleMoves[index];". The status bar indicates "Ln: 109 Ch: 2 Col: 5 TABS CRLF". To the right, the Output window shows the application's startup logs and the error message:

```
Unhandled exception. System.ArgumentOutOfRangeException: 'Index was out of range. Must be non-negative and less than the size of the collection. (Parameter 'index')'
at System.Collections.Generic.List`1.get_Item(Int32 index)
at Program.<<Main>>g__ComputerTurn|0_1(<>c__Display
) in C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221_A4\Lab_11\Projects\Tic Tac Toe\Program.cs:line 109
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 29

C:\Users\bhanu\OneDrive\Desktop\courses\sem6\STT\22110221_A4\Lab_11\Projects\Tic Tac Toe\bin\Debug\Tic Tac.exe (pr
) exited with code 0 (0x0).
To automatically close the console when debugging stops, go to Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

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

The screenshot shows the Microsoft Visual Studio interface during a debugging session. The title bar says "Lab_11". The main window displays the code for "Program.cs" in the "Tic Tac Toe" project. A break point is set at line 109. The code is the same as in the previous screenshot. The status bar indicates "Ln: 104 Ch: 8 Col: 20 TABS CRLF". The Autos window shows the variable values:

Name	Type	Value
Rando...	{System.Random.ThreadLocal<Random>}	
index	int	0
possible...	Count = 0	

The Call Stack window shows the current call stack:

Name	Lang
[External Code]	
Tic Tac Toe.dll!Program.<Main>\$._ComputerTurn()	C#
Tic Tac Toe.dll!Program.<Main>\$String[] args	C#

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.

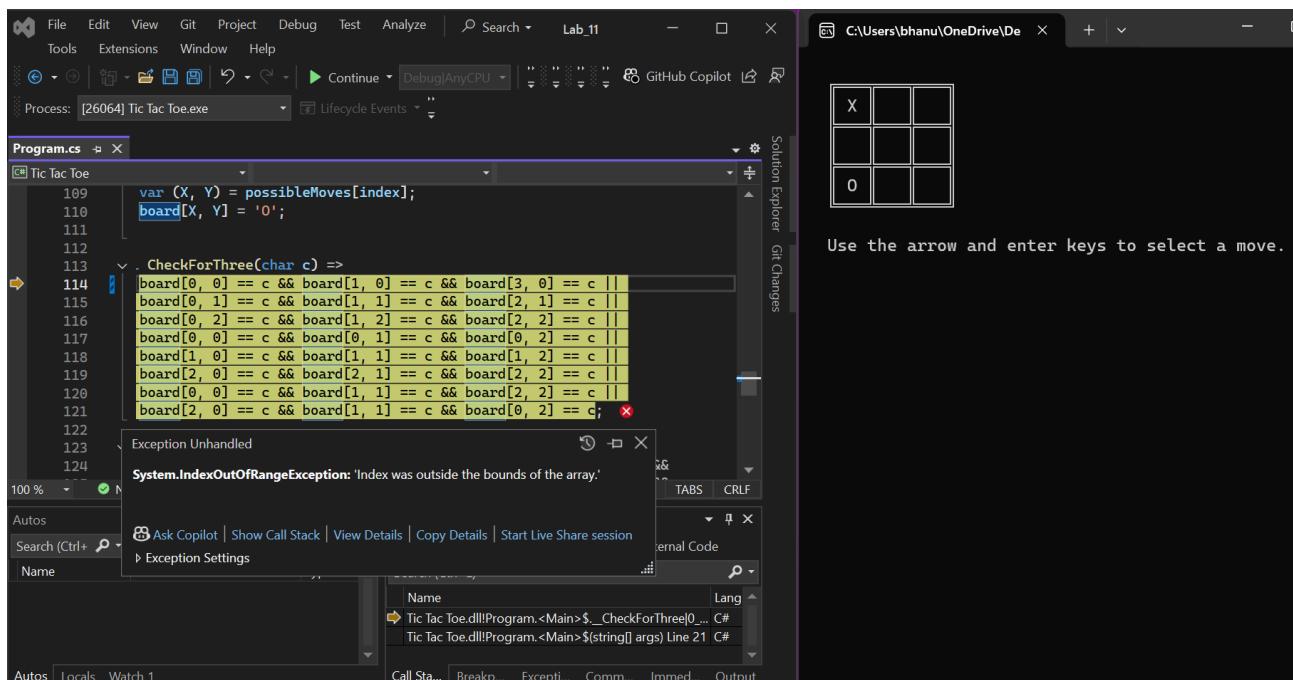
Computer was not able to find any turn because it is not adding the possible moves which it found into the array

```
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.



The screenshot shows two windows from Microsoft Visual Studio. The left window is titled 'Lab_11' and displays the 'Program.cs' code for a Tic Tac Toe game. The right window is titled 'Microsoft Visual Studio Debug' and shows the output of the program, which has crashed with an 'IndexOutOfRangeException'.

Program.cs

```
110     board[0, 0] = 'X';
111     board[0, 1] = 'O';
112     board[0, 2] = 'X';
113     bool CheckForThree(char c) =>
114         board[0, 0] == c && board[1, 0] == c && board[2, 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, 1] == c && board[1, 1] == c && board[2, 1] == c
121         board[0, 2] == c && board[1, 2] == c && board[2, 2] == c
122     bool CheckForFullBoard() =>
123         board[0, 0] != ' ' && board[1, 0] != ' ' && board[2, 0] != ' '
124         board[0, 1] != ' ' && board[1, 1] != ' ' && board[2, 1] != ' '
125         board[0, 2] != ' ' && board[1, 2] != ' ' && board[2, 2] != ' '
126
127         return true;
128     }
129 }
```

Diagnostic Tools

Tic Tac Toe

X

Unhandled exception. System.IndexOutOfRangeException: Index was outside the bounds of the array.
at Program.<Main>\$>g__CheckForThree|0_2(Char c, <>c__DisplayClass0_0&)
in 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 26980) 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 . . .

The screenshot shows the Microsoft Visual Studio IDE interface. The top menu bar includes 'File', 'View', 'Git', 'Project', 'Debug', 'Test', 'Analyze', 'Search', and 'Lab_11'. The left sidebar has 'Extensions' and 'Window' sections. The main code editor window displays C# code for a Tic Tac Toe game:

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

The status bar at the bottom shows 'No issues found', line 'Ln: 19', character 'Ch: 4', column 'Col: 10', and tabs for 'TABS' and 'CRLF'. Below the code editor is a 'Call Stack' window.

The right side of the interface features a 'Tic Tac Toe' board visualization with three rows and three columns. The first row contains 'X' in the first column and '0' in the second column. The status bar on the right says 'Use the arrow and enter keys to select a move.'

```

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, 1] == c && board[2, 2] == 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]

```

Output

```

'Tic Tac Toe.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft\Tic Tac Toe.exe' (CoreCLR: clrhost). The program '[6576] Tic Tac Toe.exe' has exited with code 0 (0x0).

```

Error List | Output

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] == c ||
    board[0, 2] == c && board[1, 2] == c && board[2, 2] == c

```

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

```

Then, added break points.

Breakpoints

```

6     int losses = 0;
7     while (true)
8     {

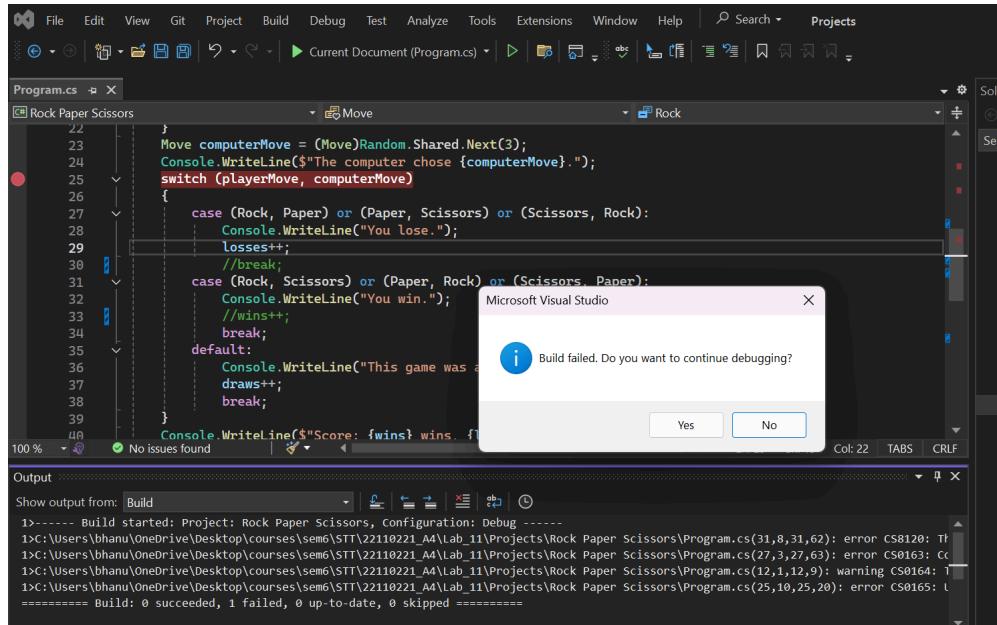
```

```

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");
goto GetInput;
Console.Write("Choose");
Move playerMove = -1;
switch ((Console.ReadLine()

```

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;
                    break;
                case (Scissors, Rock) or (Paper, Scissors) or (Rock, Paper):
                    Console.WriteLine("You win.");
                    wins++;
                    break;
                default:
                    Console.WriteLine("This game was a draw.");
                    draws++;
                    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.

```
File Edit View Git Project Debug | 🔎 Projects - X
Test Analyze Tools Extensions Window Help
Process: [17128] Rock Paper Scissors.exe Lifecycle Events
Program.cs X
Rock Paper Scissors Move Rock
35     losses++;
36     break;
37 default:
38     Console.WriteLine("This game was a draw.");
39     draws++;
40     break;
41 }
42 Console.WriteLine($"Score: {wins} wins, {losses} losses, {draws} draws");
43 Console.WriteLine("Press Enter To Continue..."); ⏱ 6ms elapsed
44 Console.ReadLine();
45
46 } Move
47
100 % No issues found
Autos Call Stack
← → Search Depth: 3 | View all Threads |
Search (Ctrl+E) ⏮ Name Value Type
Lang
```

Rock, Paper, Scissors
Choose [r]ock, [p]aper, [s]cissors, or [e]xit:
The computer chose Scissors.
You win.
Score: 0 wins, 5 losses, 1 draws

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.

Rock, Paper, Scissors

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

Program.cs

```
50    Rock
51    {
52        case (Rock, Scissors) or (Scissors, Paper) or (Paper, Rock)
53        Console.WriteLine("You win.");
54        wins++;
55        break;
56    default:
57        Console.WriteLine("This game was a draw.");
58        draws++;
59        break;
60    }
61    Console.WriteLine($"Score: {wins} wins, {losses} losses, {draws}
62    Console.WriteLine("Press Enter To Continue...");
63    Console.ReadLine();
```

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

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:
|

Program.cs

```
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; < 4ms elapsed
21       default:Console.WriteLine("Invalid Input. Try Again..."); goto GetInput;
22   }
```

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, Autos, and Search. The Call Stack window shows the current call stack entry: "Rock Paper Scissors.dll!Program.<Main>\$(str... C#)". The Autos window shows variable values, and the Search window shows the search term "Name". The output window on the right displays the message "The computer chose Rock.".

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

This screenshot shows the same Visual Studio environment after the bug fix. The code editor now contains the following corrected 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 on the right 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.

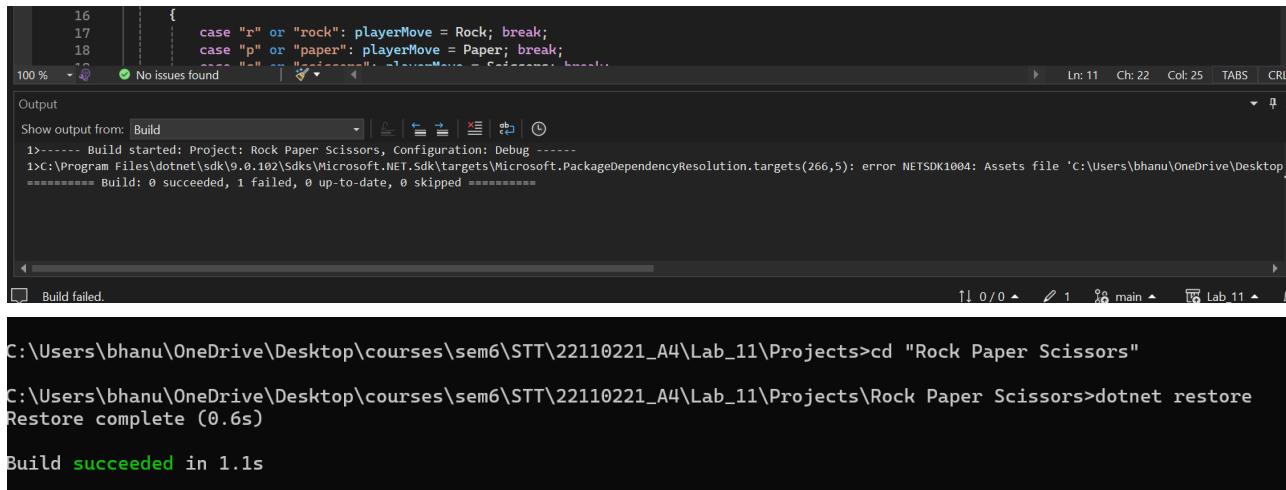
All bugs are found!

3. Results and Analysis

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 a debugger interface with two main windows. The top window is a code editor displaying C# code for a Rock-Paper-Scissors game. The bottom window is an 'Output' window showing the command-line interface used to run the project.

```
16
17
18
19     case "r" or "rock": playerMove = Rock; break;
20     case "p" or "paper": playerMove = Paper; break;
21     case "s" or "scissors": playerMove = Scissors; break;
22
23     return playerMove;
24 }
```

```
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.
1>----- Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped -----
```

```
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

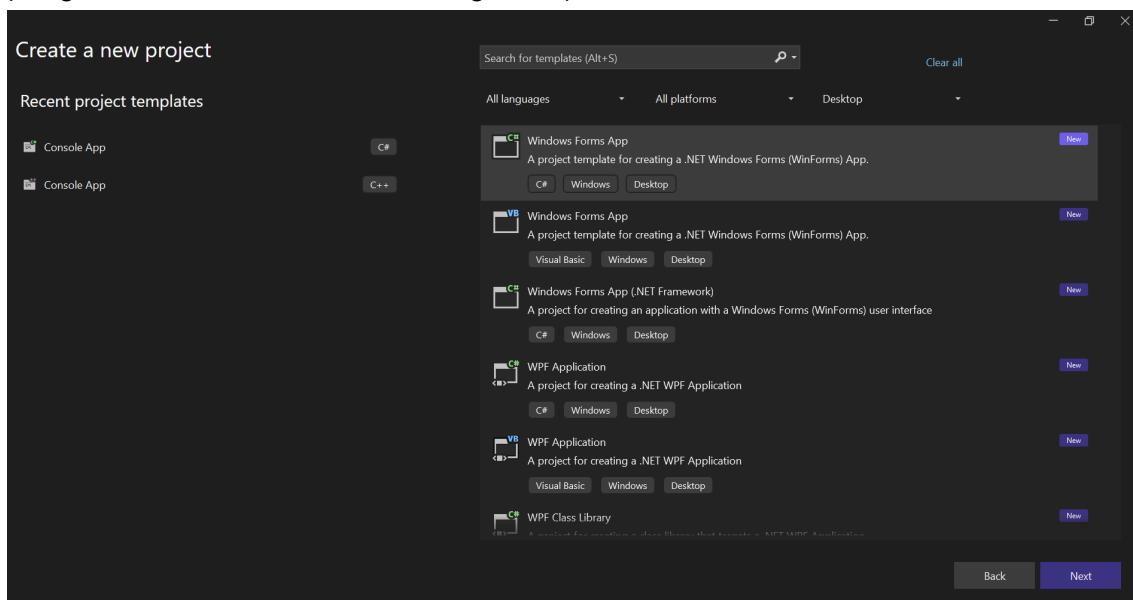
```

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).



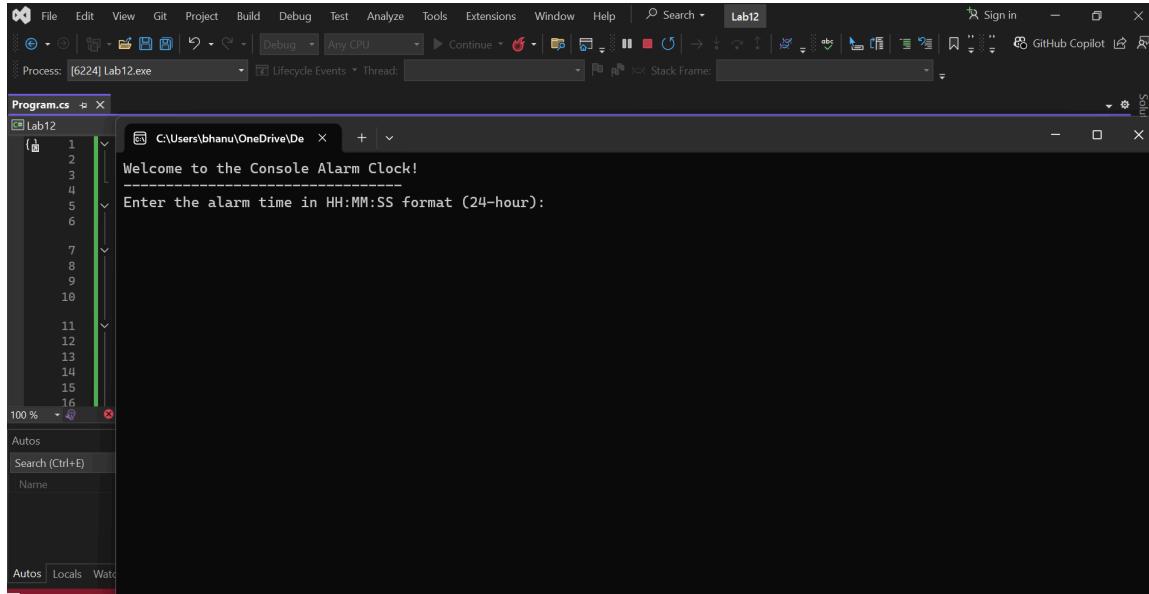
Step 2: Implementing Input, Validation, and Core Logic (Program.cs)

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;

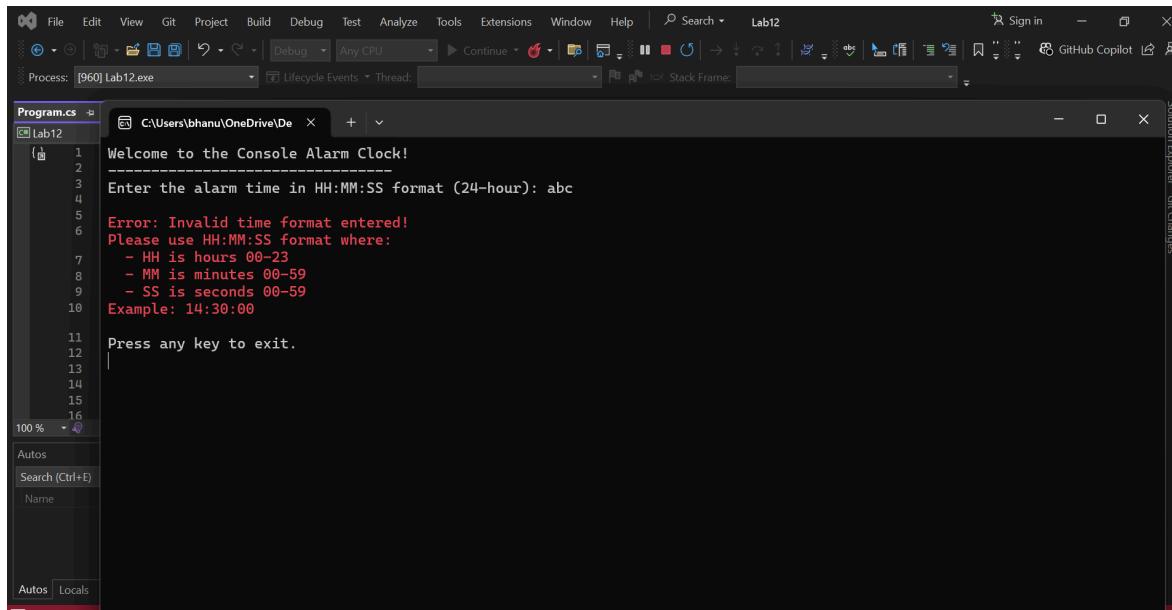
Next, I implemented the user interaction and time handling:

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



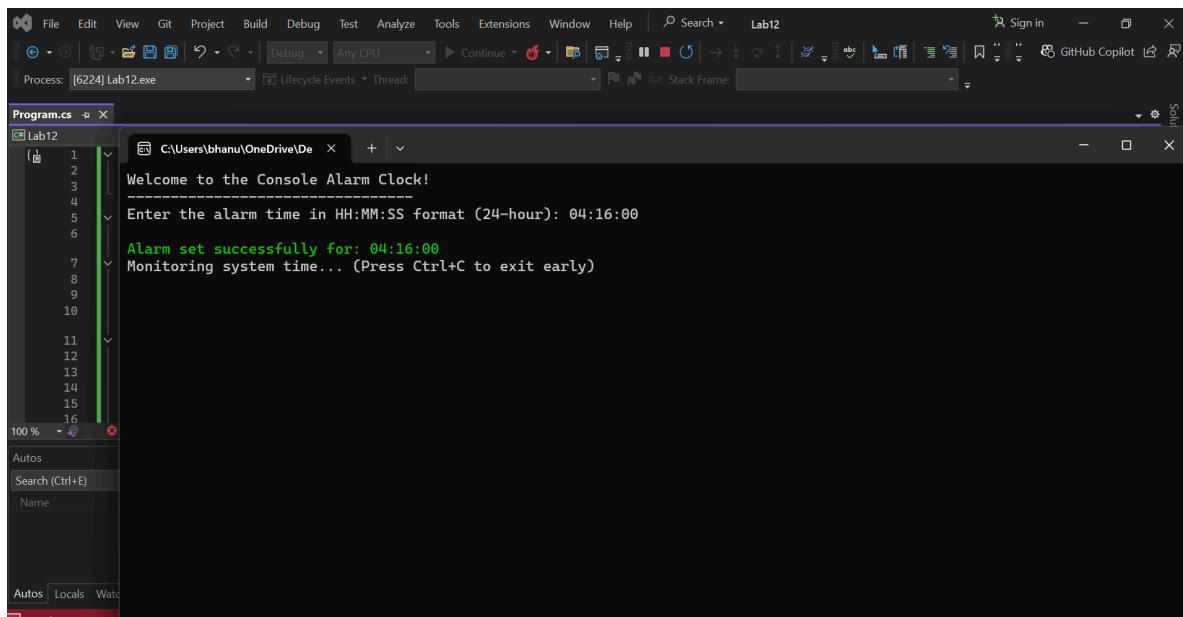
```
Lab12
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help | Search | Lab12 | Sign in | GitHub Copilot | 
Process: [6224] Lab12.exe | Lifecycle Events | Thread: | Stack Frame: | 
Program.cs > x C:\Users\bhanu\OneDrive\De + | 
Lab12
1 Welcome to the Console Alarm Clock!
-----
2 Enter the alarm time in HH:MM:SS format (24-hour):
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

- Read the user's input using `Console.ReadLine`.
- 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.



```
Lab12
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help | Search | Lab12 | Sign in | GitHub Copilot | 
Process: [960] Lab12.exe | Lifecycle Events | Thread: | Stack Frame: | 
Program.cs > x C:\Users\bhanu\OneDrive\De + | 
Lab12
1 Welcome to the Console Alarm Clock!
-----
2 Enter the alarm time in HH:MM:SS format (24-hour): abc
3 Error: Invalid time format entered!
4 Please use HH:MM:SS format where:
5   - HH is hours 00-23
6   - MM is minutes 00-59
7   - SS is seconds 00-59
8 Example: 14:30:00
9
10
11
12
13
14
15
16
```

- 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.



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help | Search | Lab12 | Sign in | GitHub Copilot | Process: [6224] Lab12.exe | Lifecycle Events | Thread: | Stack Frame: | 

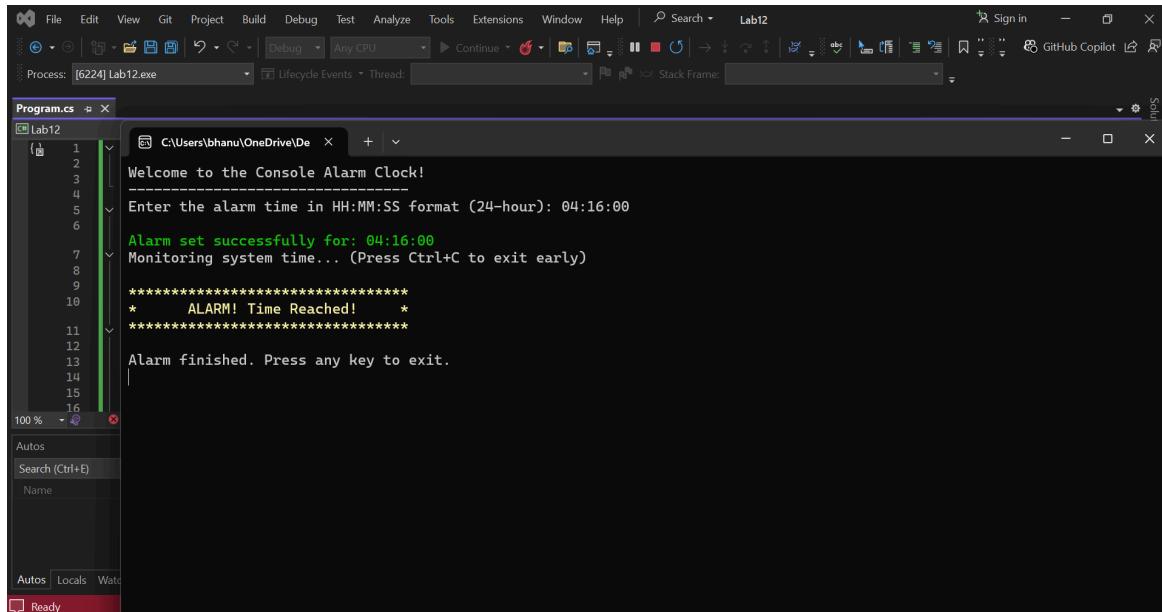
Program.cs | x | C:\Users\bhanu\OneDrive\De | + | v | 
Lab12
1 Welcome to the Console Alarm Clock!
-----
2 Enter the alarm time in HH:MM:SS format (24-hour): 04:16:00
3 
4 Alarm set successfully for: 04:16:00
5 Monitoring system time... (Press Ctrl+C to exit early)
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16

100 % | Autos | Locals | Watch | 

Autos | Search (Ctrl+E) | Name | 

Autos | Locals | Watch | Ready |
```

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.



```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help | Search | Lab12 | Sign in | GitHub Copilot | Process: [6224] Lab12.exe | Lifecycle Events | Thread: | Stack Frame: | 

Program.cs | x | C:\Users\bhanu\OneDrive\De | + | v | 
Lab12
1 Welcome to the Console Alarm Clock!
-----
2 Enter the alarm time in HH:MM:SS format (24-hour): 04:16:00
3 
4 Alarm set successfully for: 04:16:00
5 Monitoring system time... (Press Ctrl+C to exit early)
6 
7 
8 
9 
10 
11 
12 
13 
14 
15 
16

100 % | Autos | Locals | Watch | 

Autos | Search (Ctrl+E) | Name | 

Autos | Locals | Watch | Ready | 

***** *      ALARM! Time Reached!      * *****
***** * ***** * ***** * ***** * ***** * *****

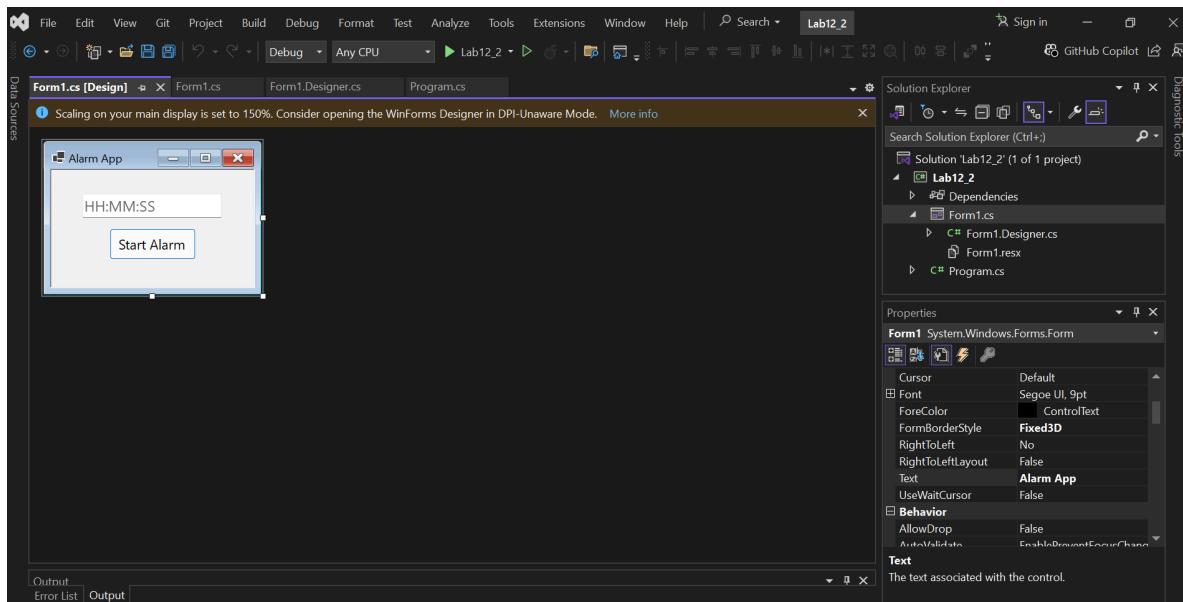
Alarm finished. Press any key to exit.
|
```

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)

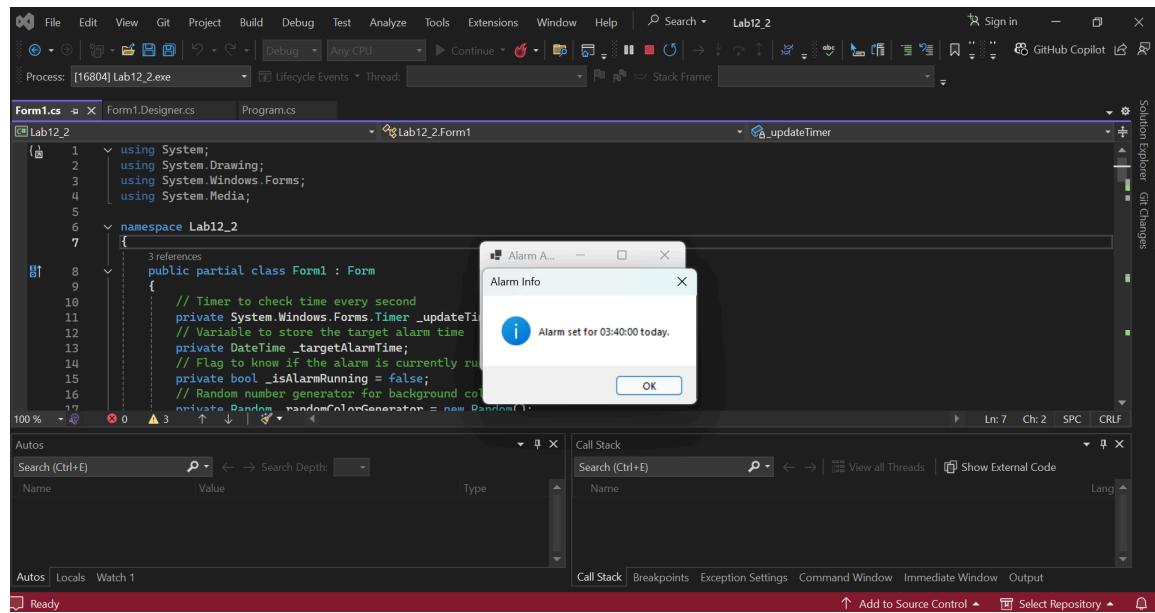
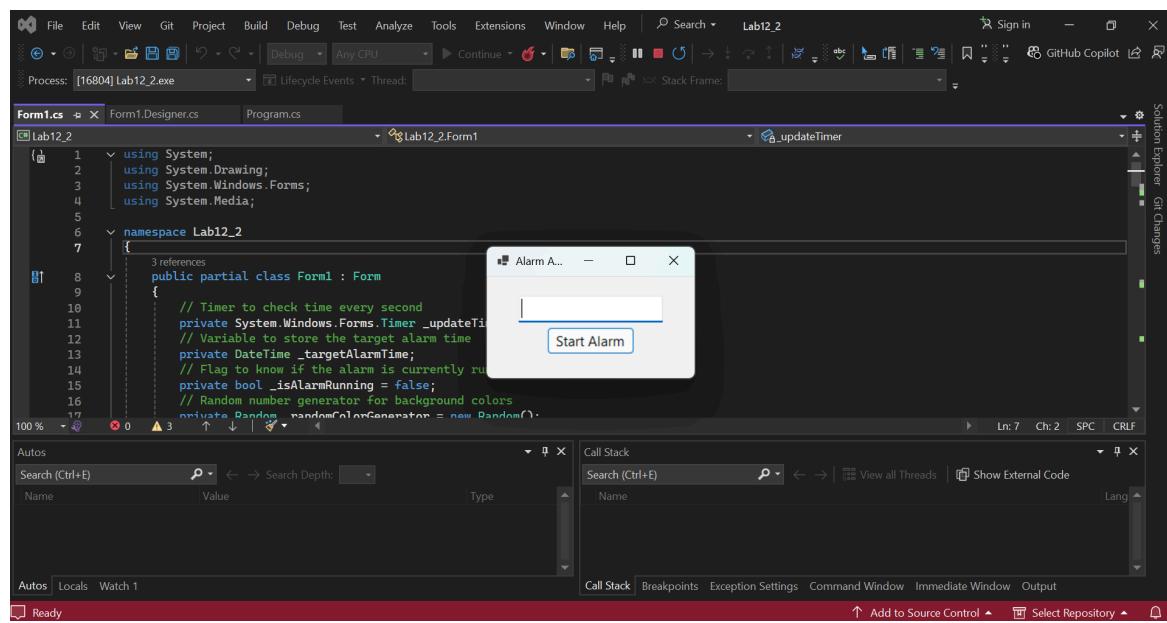
I opened Form1.cs in the Designer and added a TextBox (named txtTime) for time input and a Button (named btnStart) to initiate the alarm from the Toolbox. Then, configured control properties (PlaceholderText, Button Text, Font, Size, Location) via the Properties window for layout and usability. Set Form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

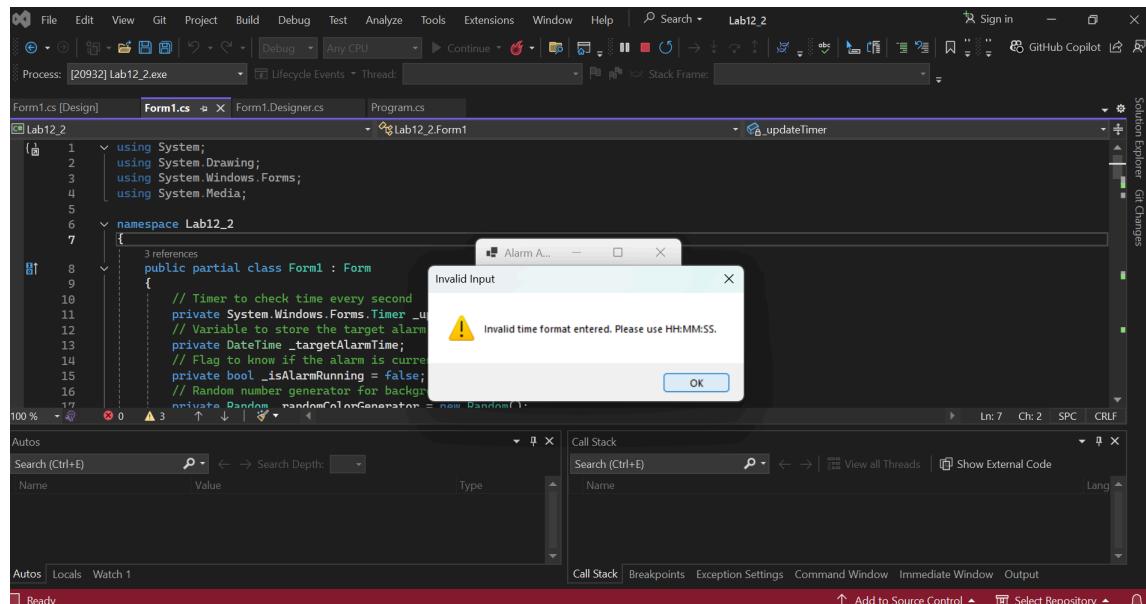
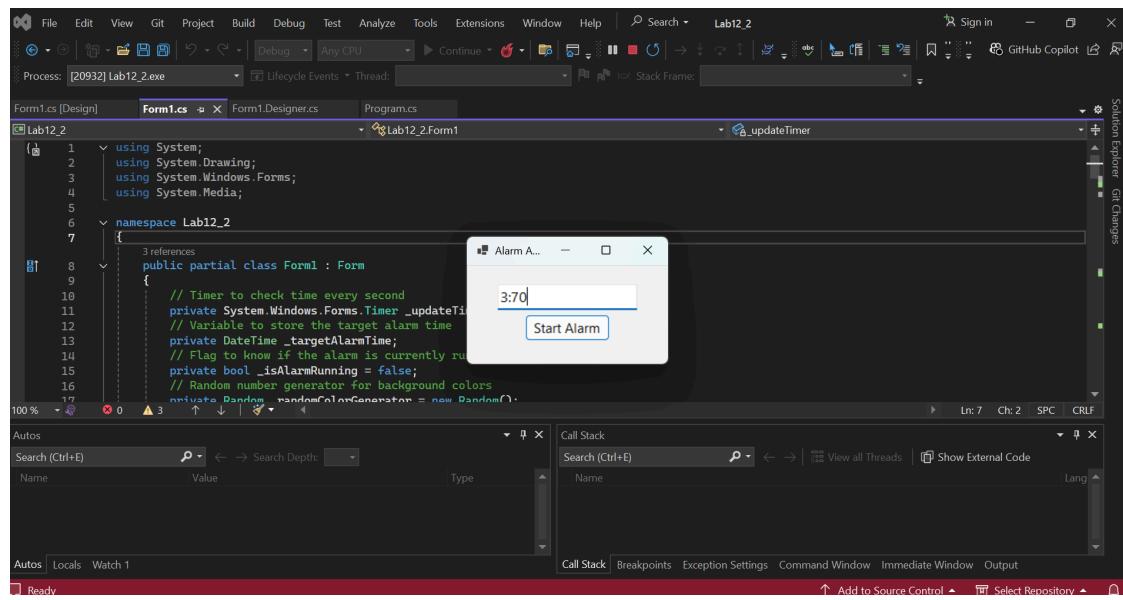


Step 3: Implementing Core Logic (Form1.cs - Code-Behind)

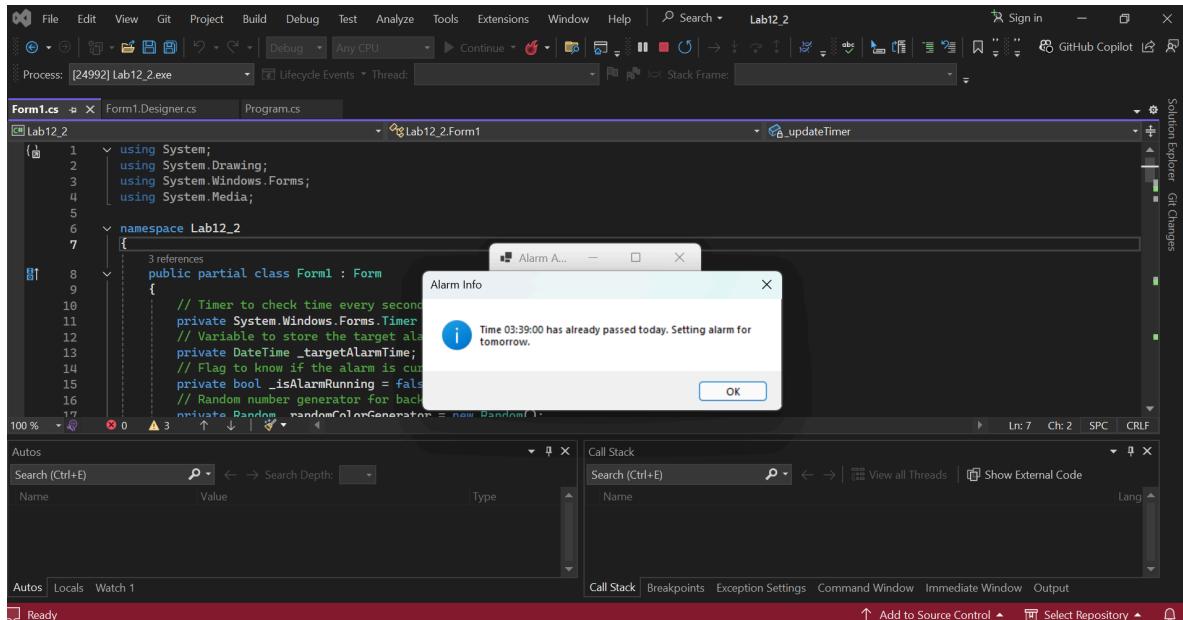
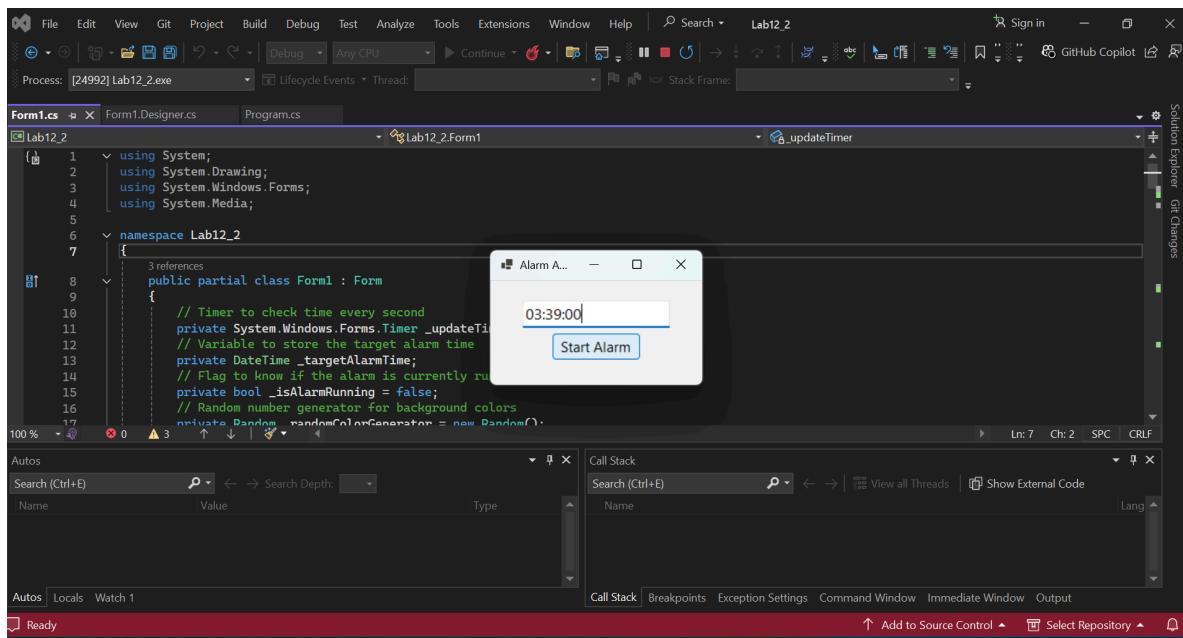
After setting up the visual elements, I switched to the code view of Form1.cs. First, I declared the necessary class-level variables to manage the application's state, such as `_updateTimer`, `_targetAlarmTime`, and `_isAlarmRunning`.

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`.



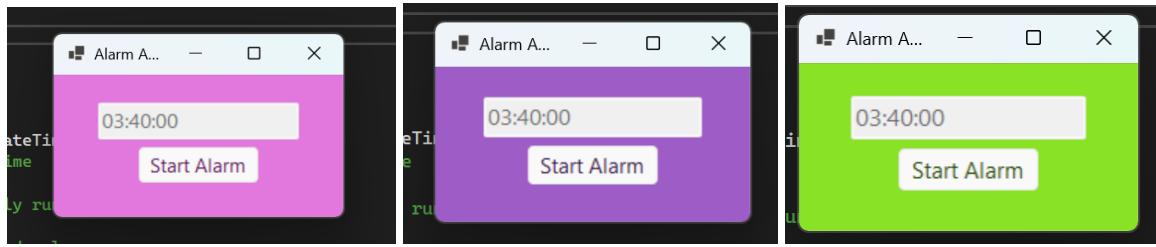


If the input was valid, I calculated the target alarm time (`_targetAlarmTime`), carefully checking if the time had already passed for today and setting it for tomorrow if needed, before showing a confirmation MessageBox.

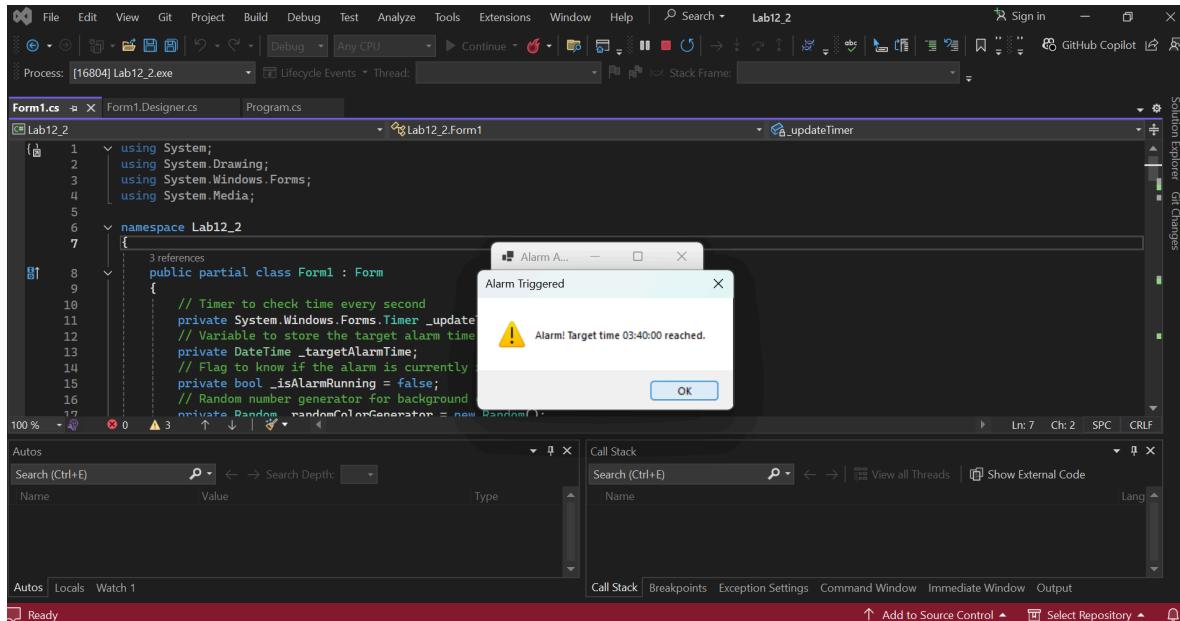


Next, I initialized and started the `_updateTimer`, set the `_isAlarmRunning` flag to true, and disabled the input controls. Following that, I created the `UpdateTimer_Tick` event handler method and assigned it to the timer's `Tick` event.

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](#)