Lab Report 11: Debugging and Bug Analysis in C# Console Games

Kushal Rathod
22110128
CS202: Software Tools and Techniques for CSE

April 2025

1 Introduction

This lab focuses on analyzing C# console game applications from the open-source repository `https://github.com/dotnet/dotnet-console-games`. The objective was to understand and explore how the Visual Studio Debugger can be used to trace the program flow, detect bugs, and fix issues leading to crashes.

We were required to investigate five bugs across one or more C# games, understand when, why, and where they occur, and correct them. This helped reinforce debugging skills and gave insight into error handling and mutation testing.

1.1 Learning Objectives

- Use the Visual Studio Debugger to trace execution flow of C# console games.
- Understand how bugs emerge in game logic or runtime.
- Fix syntactic and logical bugs, and rebuild projects successfully.
- Perform mutation testing to introduce bugs intentionally and analyze behavior.
- Analyze top-level program structures even in the absence of the traditional `Main()` method.

1.2 Development Environment

Tools Used:

- OS: Windows 11
- IDE: Visual Studio 2022 (Community Edition)
- Framework: .NET 8.0
- Language: C# 10.0
- Games Tested: Console-based games from `dotnet-console-games` GitHub repo

2 Implementation Approach

2.1 Step 1: Game Selection

From the repository, I selected the following games for testing and debugging:

- Snake
- Tanks
- Hangman

The games were cloned locally and opened in Visual Studio as individual projects. The top-level statement structure was analyzed for entry points in the absence of an explicit `Main()` method.

2.2 Step 2: Debugging Setup and Breakpoints

Breakpoints were added at key points such as:

- Game loop start
- Input handling sections
- Score update logic
- Collision detection

Using Visual Studio Debugger, I stepped through using:

- `Step Into (F11)` to dive into function calls
- `Step Over (F10)` to move over function calls
- `Step Out (Shift+F11)` to return from methods

2.3 Step 3: Bug Detection and Fixes

Bug 1: Index Out of Range (Snake)

- **What:** Index Out of Range.
- **When:** Crash occurred randomly after the snake became of a particular size.
- **Where:** `int index = Random.Shared.Next(possibleCoordinates.Count + 3600);` This line is selecting a space, where to place the next food.
- **Why:** Selecting from a bigger grid than available, so there is a probability that index comes out of bound.
- **Fix:** Removed the additional number added, which is 3600 in this case, so it chooses from the available grid only.

Bug 2: Tank shooting through (Tanks)

- **What:** Tanks shooting through the partition wall.
- **When:** The tanks when moved just next to the partition wall can shoot through the wall!
- **Where: foreach (var tank in AllTanks)** line 418 to line 446 in the program.cs file.
- **Why:** The bound checking of the newly generated bullet is not proper. It is checking from after 1 step and not from the position where the bullet is generated.
- **Fix:** Fix the bound check code and check it from the place where the bullet is generated in the first place itself.

Bug 3: Crash (Tanks)

- **What:** The whole game crashes.
- **When:** The tanks when moved just next to any of the bordering walls and the shot is made on the wall, the whole game crashes.
- **Where: foreach (var tank in AllTanks)** line 418 to line 446 in the program.cs file. Same as Bug 2.

- **Why:** The bound checking of the newly generated bullet is not proper. It is checking from after 1 step and not from the position where the bullet is generated. Same as Bug 2.
- **Fix:** Fix the bound check code and check it from the place where the bullet is generated in the first place itself. Same as Bug 2.

Bug 4: Infinite Loop (Tanks)

- **What:** Infinite loop while killing in Tanks.
- **When:** When we try to kill another tank using a bullet, and before it is killed (disappears here) we hit it with another one and keep on doing this, then we will ger stuck in an infinite loop.
- **Where: Render(tank.ExplodingFrame greater than 9** and **if (Tanks[i].ExplodingFrame greater than 10)** on lines 186 and 452.
- **Why:** These lines of code are providing an animation whike the tank is killed, a blinking animation, which is taking some time. So if someone hits it again in that time the whole animation will start from beginning, and if this keeps happening this will become an infinite loop.
- **Fix:** We reduced the time of animation to an extent such that, now it will directly disappear if it is hit by the fourth bullet, and thery will be no time to hit it with a bullet again. Due to this tthere is no chance of an infinite loop again.

Bug 5: Division by zero (Snake)

- **What:** Division by zero happening in Snake Game.
- **When:** When you gust go into the game without selecting a particular speed level, the game crashes, showing a Divide by Zero Exception.
- **Where: TimeSpan sleep = TimeSpan.FromMilliseconds(10000/(100-velocity));** and **ispeedInput = 1;** on lines 13 and 25.
- **Why:** Line 13 shows that if we press enter without selecting a speed, it will select a default speed, which is [1]. The velocity of 1 is 100 and if we see in the code on line 25, we can see if we put velocity = 100, the there will be a division by Zero.

- **Fix:** We changed the code to **TimeSpan sleep = TimeSpan.FromMilliseconds(100000 velocity));** so that it never handles a case of division by zero, as the 3 levels of velocity are 100, 70 and 50.

Note: Screenshots showing breakpoints, variable watches, exceptions, and fixed output are attached below.

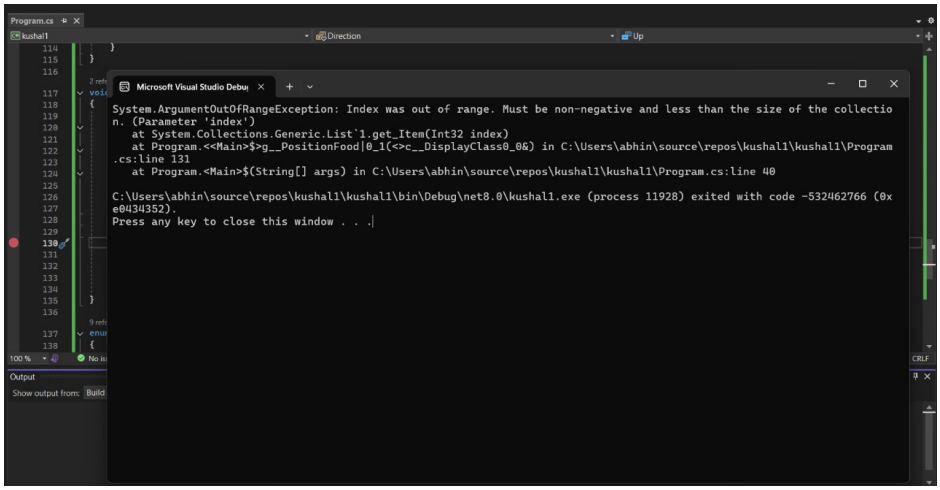

Figure 1: Bug1: Index Out of Range in Snake

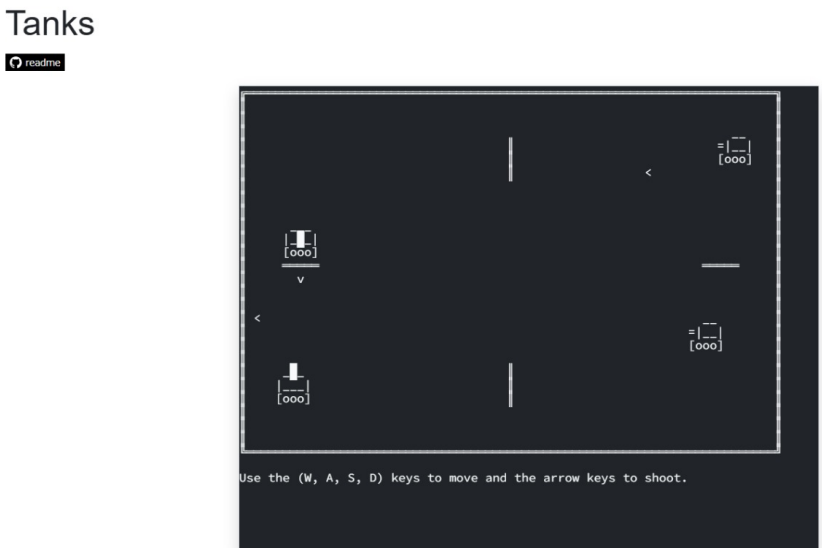

Figure 2: Bug2: Tank shooting through

Tanks

readme

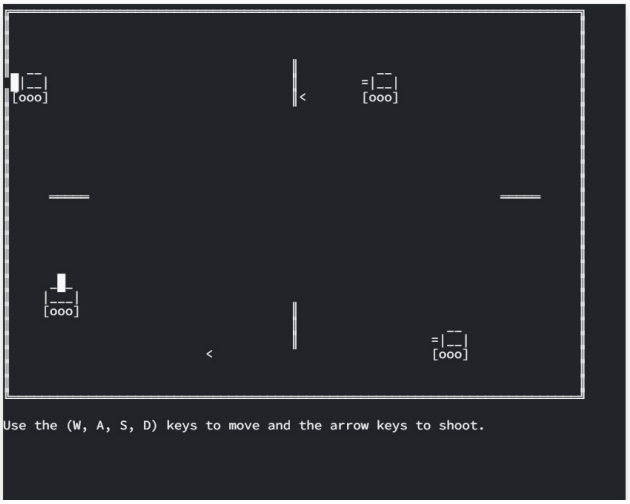

Figure 3: Bug3: Crash of game Tanks

Tanks

readme

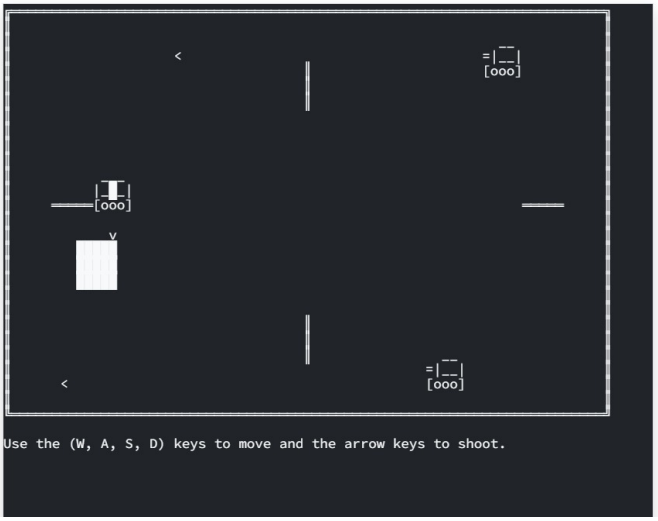

Figure 4: Bug4: Infinite loop in Tanks

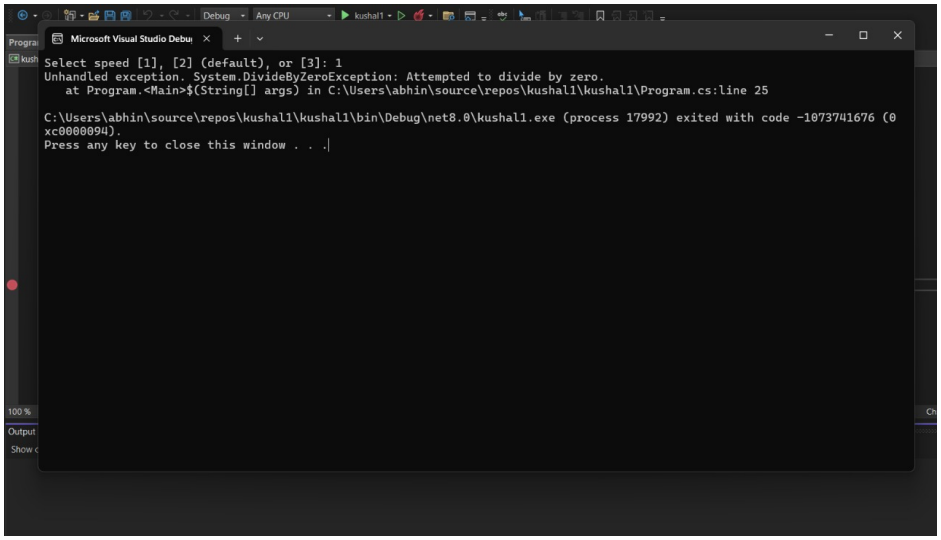

Figure 5: Bug5: Division by Zero in Snake

3 Footnote Answers

- This program uses C 9's top-level statements feature, where you don't need to explicitly define a Main() method. Instead, the code at the top level of Program.cs is automatically wrapped by the compiler into a hidden Main() entry point during compilation.
- This makes the file cleaner and easier to read for small apps or demos, and it's fully valid as long as your project targets .NET 5 or higher with C 9+.

4 Results and Observations

- Visual Studio Debugger helped pinpoint runtime errors quickly.
- Bugs introduced via mutation gave insight into common pitfalls like incorrect operators or constants.
- Top-level programs without explicit `Main()` still follow a logical entry flow which can be debugged.
- Fixes led to stable execution in all tested games after rebuild and re-execution.

Discussion and Conclusion

Challenges Faced

- Had to give a lot of time playing different games to find these Bugs.
- Some bugs were non-obvious (e.g., logical flaws causing infinite loops).
- Showing the debugging of the bugs was a difficult task to do.

Reflections

This lab was a powerful demonstration of debugging as a diagnostic and learning tool. By intentionally introducing and then fixing bugs, I became more confident in using debugging workflows and understanding how small code issues can escalate to crashes or incorrect behavior.

Lessons Learned

1. **Debugger is Your Friend:** It helps trace logic and avoid guesswork.
2. **Bugs Have Patterns:** Many errors follow recurring themes — boundary issues, null checks, loops.
3. **Input Handling Matters:** Especially in games with dynamic user input.
4. **Mutation Teaches Robustness:** Thinking like a bug helps prevent bugs.

Summary

Lab 11 emphasized real-world debugging scenarios. From identifying and fixing runtime crashes to understanding execution flow via breakpoints, this assignment strengthened my ability to write and maintain error-resilient code in C#. The mutation-based approach was especially helpful in simulating realistic errors and practicing recovery strategies.

Note: **All bug screenshots, debug snapshots, and fixed versions of game code are submitted in the GitHub repository under the branch `STT_lab11`. Please refer to the `debug_snaps/` and `bugfixes/` folders.**