Peyton Wolf

4/13/2025

Cst-250 Programming in C# ||

Mark Smithers

Grand Canyon University

Milestone 4

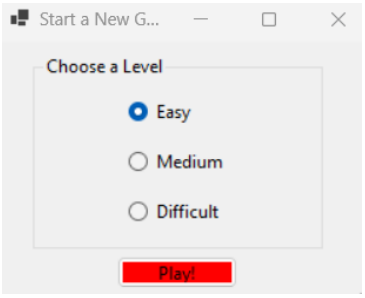
Loom Screencast Videos

https://www.loom.com/share/23ed9f987cc844718a9cd0481e0b1f51?sid=9089824b-6c26-4944-9809-f78e39cdb13f

GitHub Link

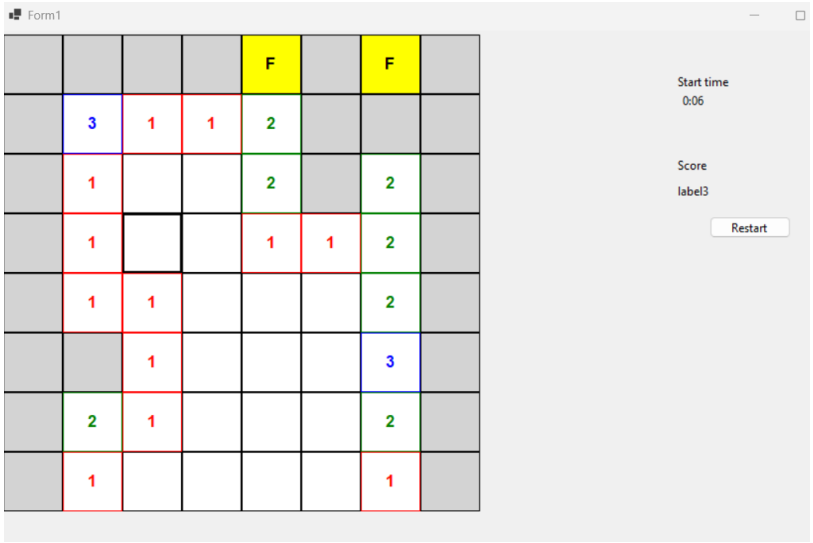
https://github.com/KnoxHighStax/CST250/tree/main/Milestone4

Starting a Game to Play



This is going to be the start of our game and will be the first page that will be poping up for the user to interact with. The user will be presented with 3 options to select the “Difficulty”, essentially the Size of the board and the percentage of bombs that will populate. When the user picks a radio button options, being Easy, Medium, or Difficult, and then they press play the game will randomly generate a game board and get the percentage of bombs for that board.

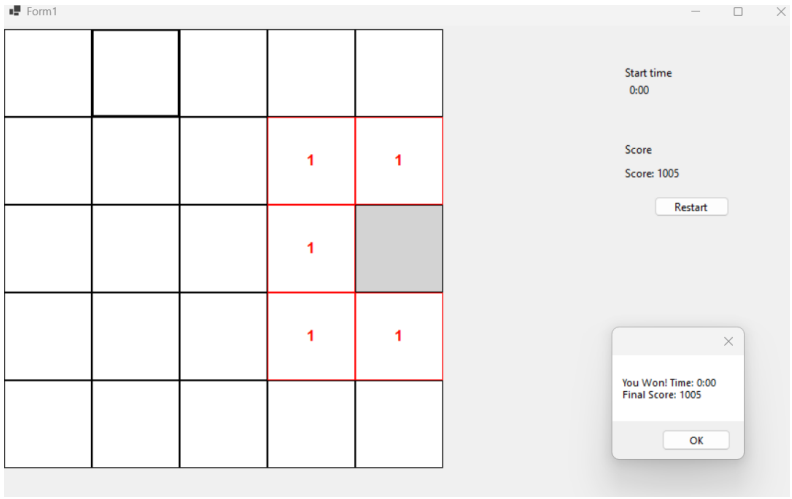
Game In Progress with Background Color and Showing Cell State



In this screenshot we are demonstrating how we have connected the back-end logic with the GUI that the user will be utilizing. Here we have added the ability for the user to be able

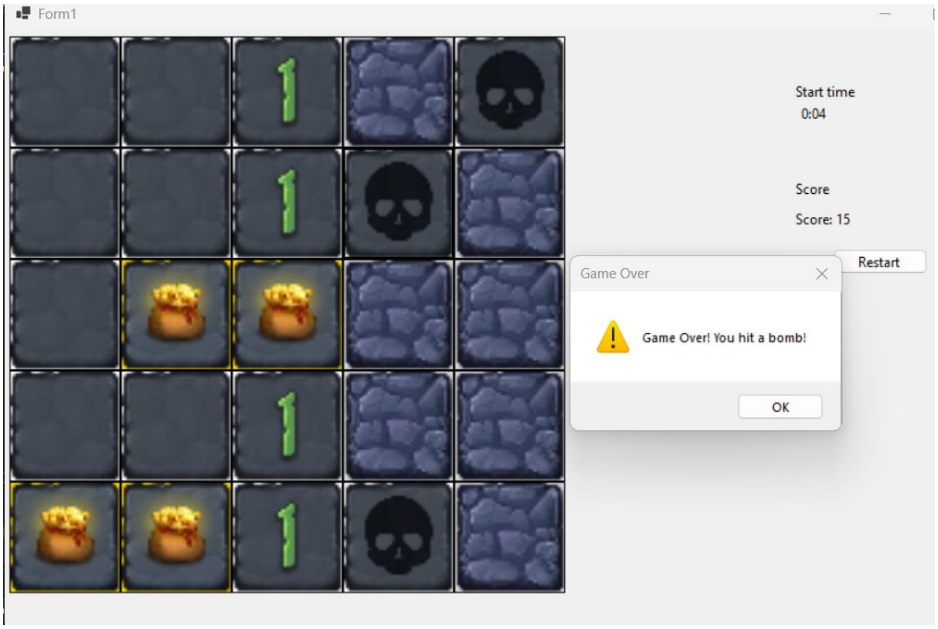
to play the game in a more interactive version! Here the user will be able to see the game perform proper Flood fills, live neighbor counts, as well as keep its ability to flag specific cells while keeping things looking great with color.

Game Is Over Status



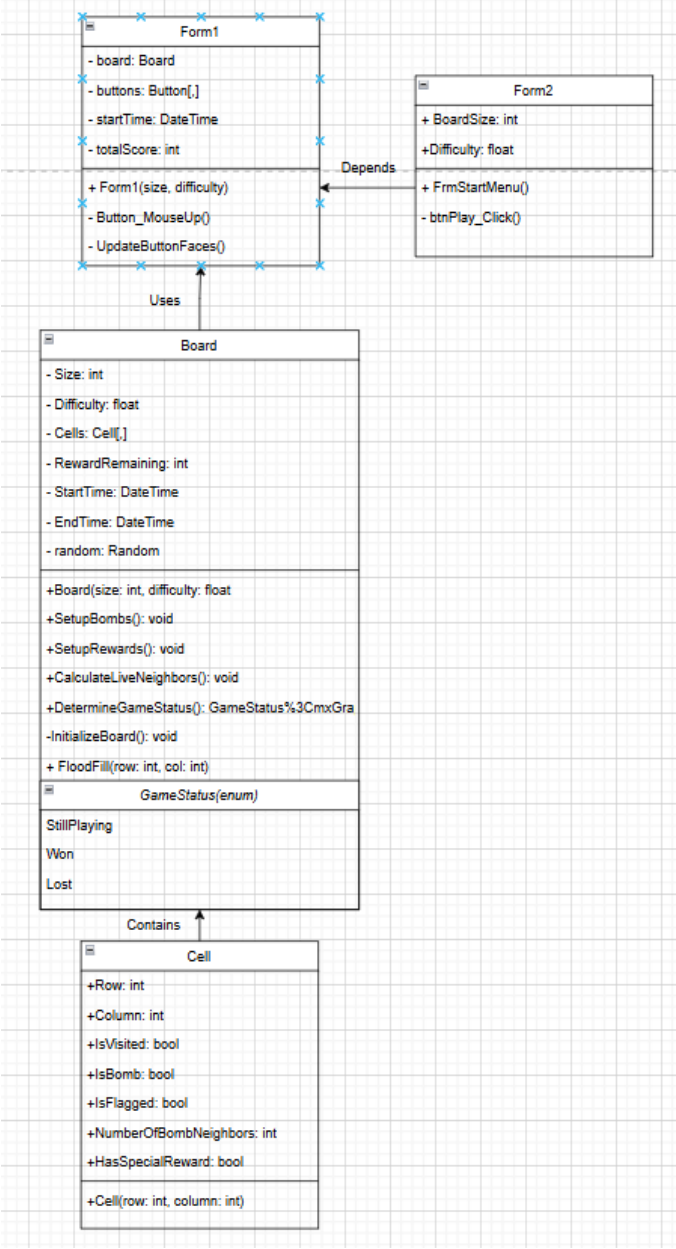
In this screenshot we are demonstrating how during the game if at any point the user reveals all safe cells or hits a bomb they will either be presented with a Game Over message status or a You Won status message with the total number of points that has been scored in this game. The points that can be received for certain cell clicks is hard coded in the backend logic for when a cell IsVisited, IsBomb, LiveNeighbor, etc.

Adding Game Assets



In this screenshot we are just showing how we have added game assets for our Minesweeper game. We have added various game assets to replace our previous code or just place a letter for what type of cell it was. Now we can display a graphic image to represent it. This will help to make the game more fun for the user and a lot easier to work with and see.

UML



Questions

1. What was challenging?
   I would have to say that the most challenging aspect of debugging this Minesweeper Game was identifying why the floodfill algorithm wasn't updating the GUI properly and figuring out why number images weren't displaying for live neighbor counts. The floodfill issue was particularly tricky because the logic seemed to be correct at first glance, but the order of operations in the click handler was causing the recursion to terminate prematurely.
2. What did you learn?
   Through this project I have learned the importance of proper sequencing in teh game logic, especially how setting a cell's "IsVisited" flag too early could break recursive algorithms like floodfill. I also gained a deeper understanding of how to manage GUI state in synchronization with the backend game logic. The image loading helped to reinforce the value of thorough testing for even seemingly simple loops to help ensure there are no errors. I also gained some practical techniques for debugging recursion functions and how to implement fallback.
3. How would you improve on the project?

   To help improve this project I will implement several enhancements. First, I'd add proper error handling and fallback rendering for missing image assets. Then I would optimize the floodfill algorithm to use an iterative approach instead of recursion to prevent stack overflow on large boards. Also, we could add visual feedback for the game state or maybe a more proper scoring system with time bonuses and difficulty multiplier.

4. How can you use what you learned on the job?
   I would say the problem-solving skills developed during this project, particularly in debugging complex state interactions between the GUI and the game logic. This experience has taught me how to methodically trace issues through multiple layers of an application, a skill valuable when working with any complex system. Also, the lessons from proper sequencing of operations and state management apply to any iterative approach.

Day-to-Day

Monday

Start:5pm End: 6:30pm Activity: Read Chapter 9.1

Start: End: Activity:

Start: End: Activity:

Tuesday

Nothing

Wednesday

Start:6pm End:9pm Activity: Complete Reading Chapter 9.2

Start:9pm End:10pm Activity: Activity DQ1

Thursday:

Start:6pm End:9pm Activity: Started on Assignment 4 / Pizza Maker

Start: 5pmEnd:6pm Activity: Tutoring session

Friday

Start: 5pmEnd:6pm Activity: Tutoring session

Start:8pm End:10pm Activity: Started on Milestone 4 Project

Saturday

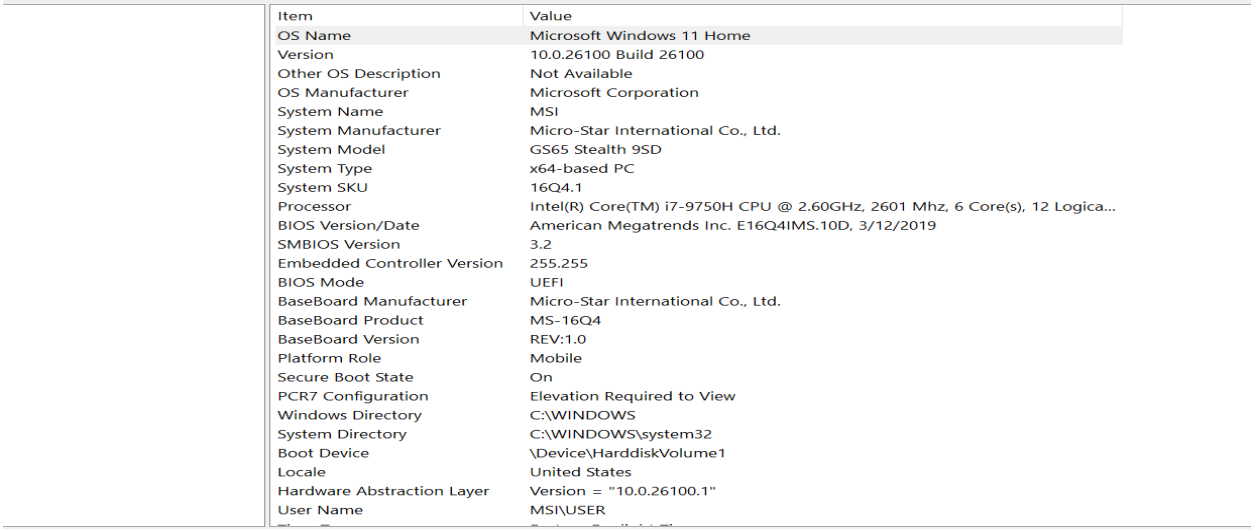
Start: 12pm End:4pm Activity: Continued Assignment 4 / GUI Version

Sunday

Start:11am End:3pm Activity: Continued Milestone 4 Project

Start:6pm End:8pm Activity: Continued Milestone 4 Project

Specs



Item	Value
OS Name	Microsoft Windows 11 Home
Version	10.0.26100 Build 26100
Other OS Description	Not Available
OS Manufacturer	Microsoft Corporation
System Name	MSI
System Manufacturer	Micro-Star International Co., Ltd.
System Model	GS65 Stealth 9SD
System Type	x64-based PC
System SKU	16Q4.1
Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2601 Mhz, 6 Core(s), 12 Logica...
BIOS Version/Date	American Megatrends Inc. E16Q4IMS.10D, 3/12/2019
SMBIOS Version	3.2
Embedded Controller Version	255.255
BIOS Mode	UEFI
BaseBoard Manufacturer	Micro-Star International Co., Ltd.
BaseBoard Product	MS-16Q4
BaseBoard Version	REV:1.0
Platform Role	Mobile
Secure Boot State	On
PCR7 Configuration	Elevation Required to View
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Locale	United States
Hardware Abstraction Layer	Version = "10.0.26100.1"
User Name	MSI\USER

Test Cases

Test Case Id	Test Case Description	Test Steps	Test Data	Expected Results	Actual Results	Pass/ Fail
Test1	Valid game start with medium difficulty.	1.Launch Form2 (Start Menu) 2. Select Medium difficulty 3. Click Play	BoardSize = 8 Difficulty = 0.15	Game board(8x8) Loads in Form1 with 15% bombs	8x8 board loaded correctly	Pass
Test2	Left-Click on safe cell	1. Start game (Medium) 2. Left-click on a cell with 0 neighbors	Row = 3 Col = 3 (Safe Cells)	Floodfill executes, revealing all connected safe cells	Only 1 cell revealed	Pass
Test3	Right-Click to flag/unflag	1. Start game (Medium) 2. Right-click on a cell	Row = 2 Col = 0	Cell toggles between flagged/unflagged state	Cell will become flagged/unflagged displaying the opposite of what	Pass

					was in cell	

Programming Conventions

1. We will be using the naming convention of camelCase for variables and parameters, like for “isVisited” and “numberOfBombNeighbors” and PascelCase for methods and classes, like for GameBoard making sure that constants in UPPER_CASE (MAX_BOARD_SIZE).
2. Making sure that we have the correct error handling in place for input validation to check the bounds before accessing cell (if (row < 0 || row>= Size) return;). Guard clauses can also asset in early returns for invalid states (if (cellIsVisited) return;) and exception handling to user try-catch for critical operations.
3. Ensuring comprehensive documentation practices with comments for all public members to be able to review and making sure we have concise inline comments to explain complex algorithms or non-obvious decisions, while trying to avoid any redundant explanations of simpler parts of the code.

Use Case Diagram With Flowchart

Description: A player interacts with a Minesweeper game to uncover safe cells while avoiding bombs.

Primary Actor: Player

Goals: - Successfully uncover all safe cells without triggering a bomb.

Stakeholders: - Player, wants to win.

- Developer, ensures logic works.

Pre-conditions: - Player must have selected a difficulty level

- Game board must be properly initialized

Post-Conditions:

- Players either wins by uncovering all safe cells or loses by triggering a bomb

- Final score and game time are recorded

Basic Flow:

1. Player launches the game and selects a difficulty level (Either Easy, Medium, Difficult).

2. System generate a game board with bombs and rewards base on the selected difficulty

3. Player clicks on a cell to reveal it

4. If the cell contains a bomb, the game ends and all bombs are revealed

5. if the cell is safe, the system reveals the number of adjacent bombs and performs the floodfill if no adjacent bombs exist.

6. Player can right-click to place/remove flags on suspected bomb cells

7. System updates the score based on the player actions (cell being flagging bombs or revealing cells

8. System checks game state after each move

9. When all safe cells are revealed, the game ends in victory and displays the final score and time

Alternative Paths:

Path 1: - Player clicks the restart button during gameplay

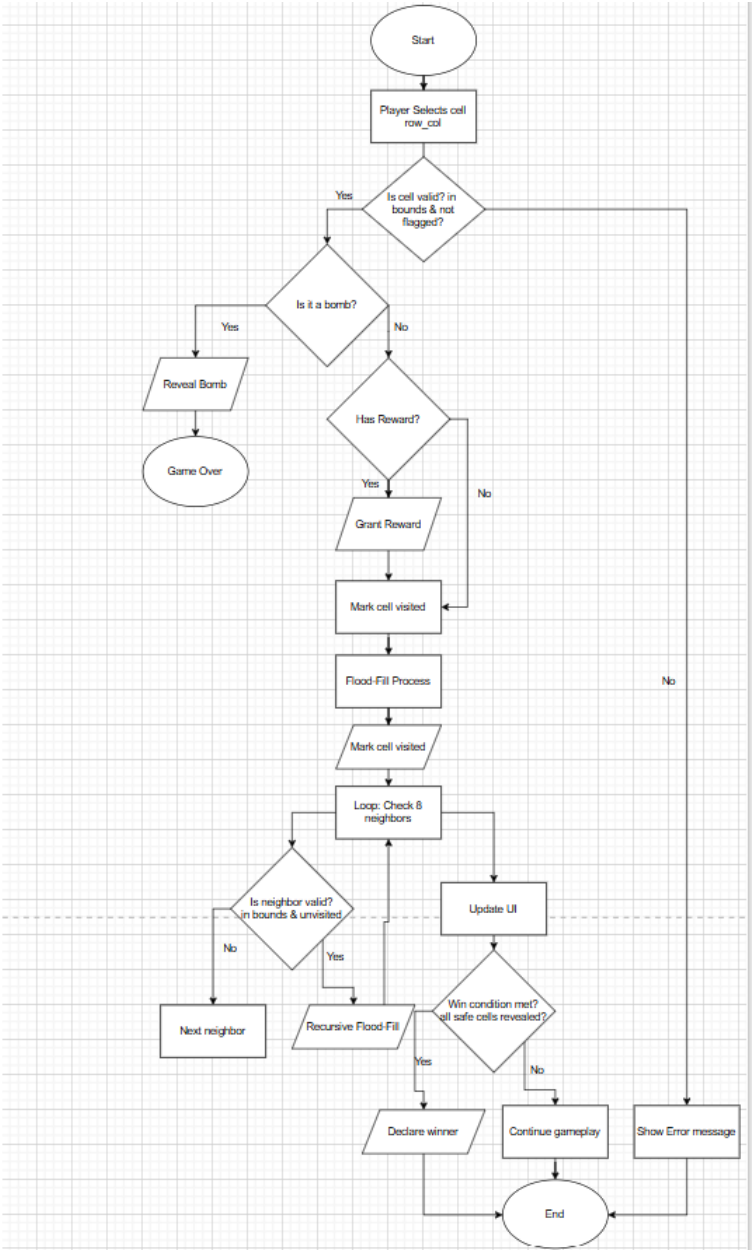
- System returns to the start menu
- Player selects a new difficulty level
- System generates a new game board with the selected difficult settings

Path2: - Player flags a cell that doesn't contain a bomb

- System deducts points form the score
- Game Continues

Path3:- Player reveals a cell containing a special reward

- System awards bonus points
- Game Continues



Bug Report

Nothing at this time.