Peyton Wolf

4/6/2025

Cst-250 Programming in C# ||

Mark Smithers

Grand Canyon University

Milestone 3

Loom Screencast Videos

https://www.loom.com/share/f4195b99cfb24ccf97f0a5ff62260638?sid=ab6d34f8-caae-4010-b0ea-15ce39f50328

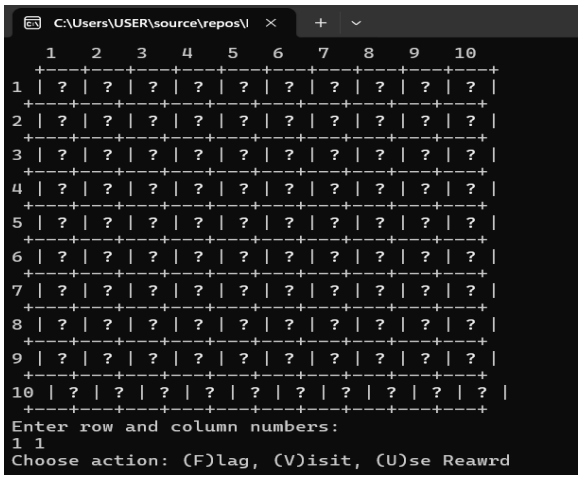
GitHub Link

https://github.com/KnoxHighStax/CST250/tree/main/Milestone3

Answers Populating for Testing Purposes

In this screenshot we are demonstrating us displaying an answer key, this is just for testing purposes to be able to test the functionality of all aspects of the code. The purpose for this is to be able to test specific things (like if there are LiveNeighbors or Rewards). In later iterations this will be removed and only will be shown at the end of a losing game.

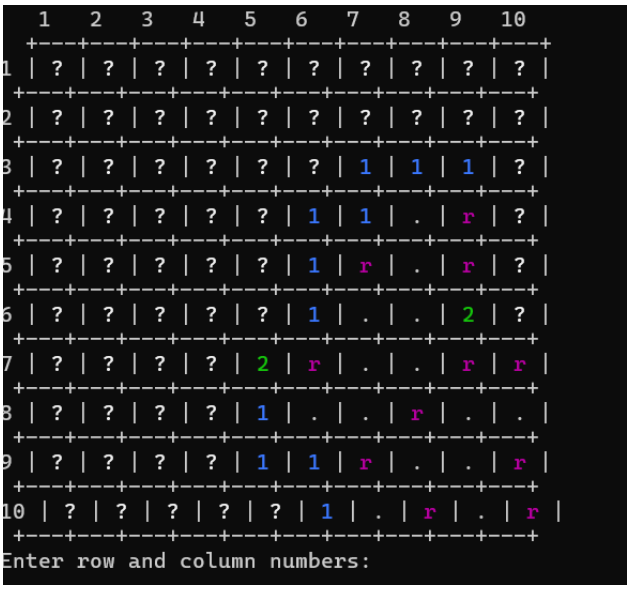
First Moves and Fill Selection



In this screenshot we are demonstrating the first moves options that will be displayed for the user to get their input on the chosen row and column of the cell they wish to visit. Once the user inputs 2 integers with a space in-between them the console will then prompt the

user to input how they would like to visit the cell, either Flag the cell, Visit the Cell, or use a Reward on a cell you may not be sure of, if you have any to use.

FloodFill for Non-Live Neighbor Cells

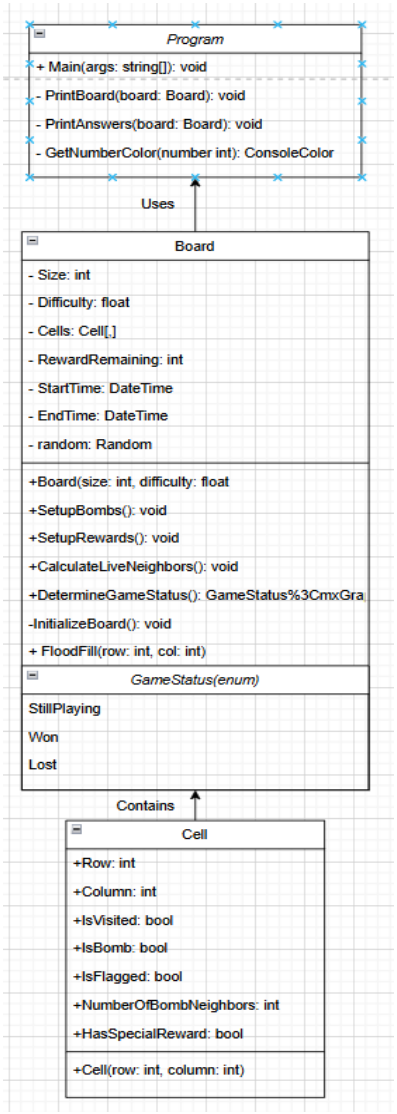


In this screenshot we are demonstrating the FloodFill operation to reveal all of the cells. When a player clicks on cell with 0 bomb neighbors, it will trigger the floodfill algorithm, this method will recursively reveal all connected empty cells until it hits a cell with bomb neighbors. The method will start by checking all 8 surrounding cells (top, bottom, left, right and diagonal); however, the recursion will stop at any point when the cell is 'out-of-bounds, already visited, is a bomb, is flagged, or has at least 1 bomb neighbor. After which each cell that has been processed will then be marked as "IsVisited = true" to prevent an infinite loop from occurring.

4. The bold are the key words introduced in this lesson.

In this lesson we have learned **recursive flood-fill algorithm** for the Minesweeper application, which automatically reveals connected safe cells when a player clicks on an empty cell (one with **0 bomb neighbors**). The overall concepts included **recursion** where the method calls itself on all 8 neighboring cells, **visited tracking** which is for marking cells as "IsVisited = true" to prevent redundant checks and **boundary conditions** helping ensure the algorithm stays within the game board's limits. The flood fill stops when it encounters numbered cells (having NumberOfBombNeighbors > 0), bombs, or flagged cells, mimicking the classic Minesweeper game behavior.

UML



Questions

1. What was challenging?

   I would have to sya the most challenging thing for this part of the project would been creating the flood fill algorithm to correctly handle Minesweeper's unique rules while avoiding infinte recursion. Ensuring that the cases properly terminated expansion at numbered cells (Where NumberOfBombNeighbors > 0) requiring careful debugging. Especially when testing edge cases near bomb clusters or for any board boundaries. Managing the recursive calls while tracking the visited cells

at the same time was initially very confusing but eventually was able to take what we learned in the Activity to work for our game.

2. What did you learn?
   I would have to say that through this project I have gained alot of practical experience implementing recursive algorithms for grid-based problems. With this exercise I have deepened my understanding of how to control recursion through proper base cases and state tracking, via the “IsVisited” flags. I also have been learning a lot about how game mechanics translate to algorithmic logic, especially with how minesweeper uses neighbor counts to govern cell revelation behavior.

3. How would you improve on the project?

   To enhance this project i would like to implement an iterative flood fill using a queue, I feel this could potentially help prevent any stack overflows for larger boards. I feel that it could also be fun to add additional features like to give the user different options of how they would like to use their reward, to either peek, give a hint, or maybe something else.

4. How can you use what you learned on the job?
   With this recursive problem-solving approach, it could directly apply to real world scenarios like pathfinding algorithms, image region selection tools and grid-based data processing. The debugging techniques are transferable to any complex system for troubleshooting. Understanding how to balance automation (floodfill) with user control (flags) provides UX design insight for workflow applications.

Day-to-Day

Monday

Start:6pm End:9pm Activity: Read Chapter 12

Start: End: Activity:

Start: End: Activity:

Tuesday

Nothing

Wednesday

Start:6pm End:8pm Activity: Complete Reading Chapter 12

Start:8pm End:9pm Activity: Activity DQ1

Thursday:

Start:6pm End:9pm Activity: Started on Assignment 3 / Recursion Solutions

Start: 5pmEnd:6pm Activity: Tutoring session

Friday

Start: 5pmEnd:6pm Activity: Tutoring session

Start:6pm End:7pm Activity: Activity DQ2

Start:8pm End:10pm Activity: Started on Milestone 2

Saturday

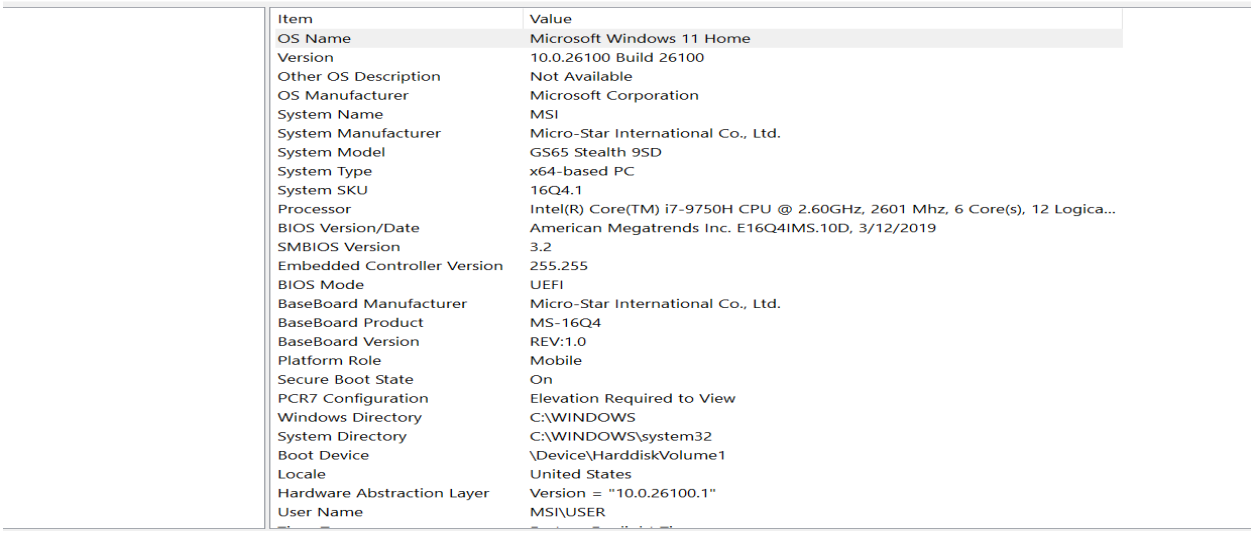
Start: 12pm End:4pm Activity: Continued Assignment 3 / Recursion Solutions

Sunday

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

Start:6pm End:8pm Activity: Continued Milestone 3 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	Verify flood-fill reveals all connected safe cells	1. Initialize a 5x5 board 2. Set cell(2,2).NumberOfBombNeighbors = 0 3. Call FloodFill(2,2)	Board Size:5x5 with empty center to 0 neighbors Border cells have 1-3 neighbors	9 center cells (2x2 area) marked visited Border cells remain hidden	Cell(1,1) to (3,3) marked visited Border cells unchanged	Pass
Test2	Verify flood fill stops at numbered cells	1. Initialize Board 3x3 2. Set: Cell(1,1)=0, Cell(1,2)=2 Call FloodFill(1,1)	Cell(1,1): 0 neighbors Cell(1,2): 2 neighbors	Only Cell(1,1) visited Cell(1,2) remains hidden	Cell(1,1)IsVisited = true Cell(1,2)IsVisited = false	Pass
Test3	Test corner cell expansion	1. Initialize 4X4 Board 2. Set Cell(0,0) =0 3. Cell FloodFill(0,0)	Corner cell(0,0): 0 neighbors	Only Cell(0,0) visited	Cell(0,0)IsVisited = ture	Pass

			Adjacent cells: 1 neighbor			

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: When a player reveals a cell with 0 adjacent bombs, the system automatically expands the revealed area using recursive flood-fill until bordered by numbered cells (1-8), optimizing gameplay by reducing manual clicks.

Primary Actor: Player

Goals: - Successfully reveal a non-bomb cell to progress.

- Avoid revealing a bomb or get game over.

- Discover rewards, if any.

Stakeholders: - Player, wants to win.

- Developer, ensures logic works.

Pre-conditions: - Game board initialized

- Bombs and rewards randomly placed

- Player has not yet lost/won

Post-Conditions: - Cell marked as visited

- Reward claimed, if available

- Game state updated, continue/win/lose

Basic Flow:

1. Player action: Selects coordinates and action will be “Visited”

2. System validation: Will confirm cell exists and is not flagged. If invalid, prompts re-entry.

3. Cell Evalution: If bomb = Game Over. If 0 neighbors will invoke floodfill(4,4). Then it will recursively reveal all adjacent 0 neighbor cells. Then stop at numbered cells.

4. Board Update: UI refreshes to show revealed safe zone. Win condition checked.

5. Game Continues: Player will be able to strategize what their next move may be.

Alternative Paths:

- Invalid/Flagged cell- 1. Triggered when player selects a flagged or invalid cell

2. The response would be an error, and the board would remain unchanged.

- Bomb Hit – 1. Will be triggered when selected cell is a bomb.

2. A response will cell turning red. Immediate loss.

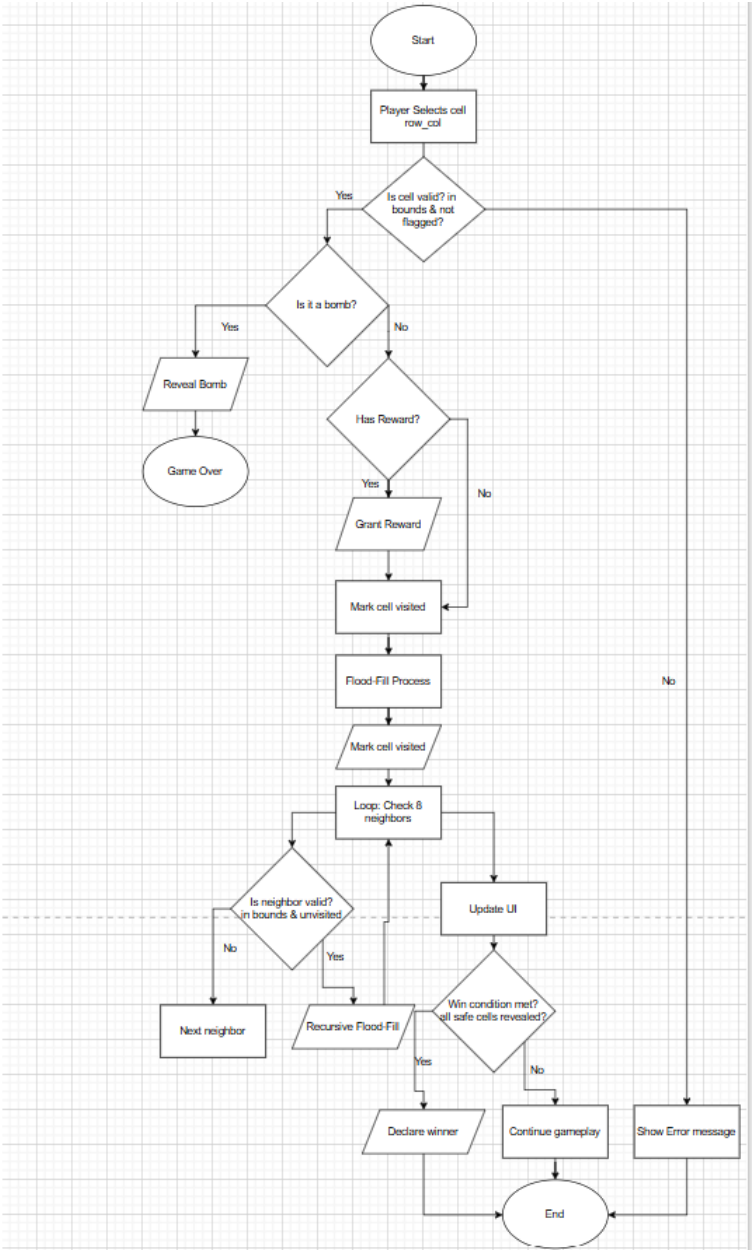
- Reward Discovery – 1. Triggered when a revealed cell has a reward.

2. Responded with a message, “Reward Earned, peek at cell.”. Reward remaining incremented.

- Win via Flood-Fill – 1. Triggered when flood-dill reveals last safe cells.

2. it will respond will a message, “You Won!”.

GameStatus=Won.



Bug Report

Nothing at this time.