Peyton Wolf

4/27/2025

Cst-250 Programming in C# ||

Mark Smithers

Grand Canyon University

Milestone 6

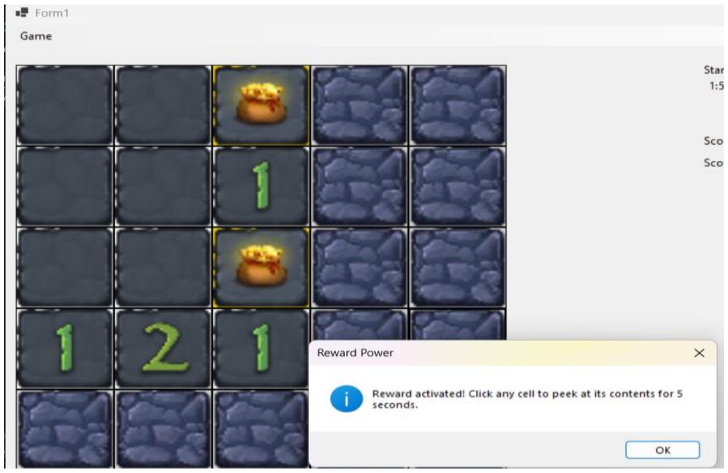
Loom Screencast Videos

https://www.loom.com/share/3cb01c0a088143958b0e675135ac20af?sid=5e52005a-7096-4ea5-8ee3-c0615e8d61a8

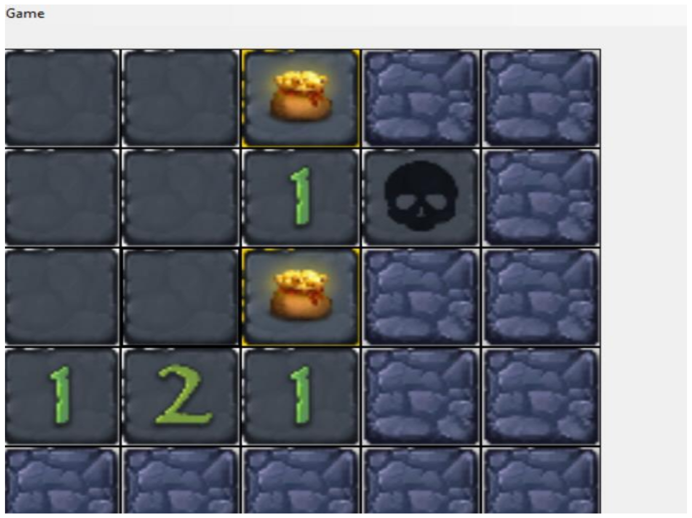
GitHub Link

https://github.com/KnoxHighStax/CST250/tree/main/Milestone6

Using Special Reward

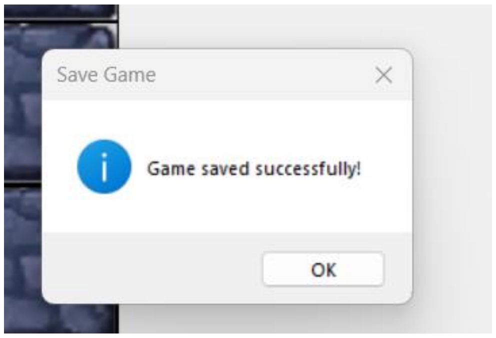


Reward activated! Click any cell to peek at its contents for 5 seconds.

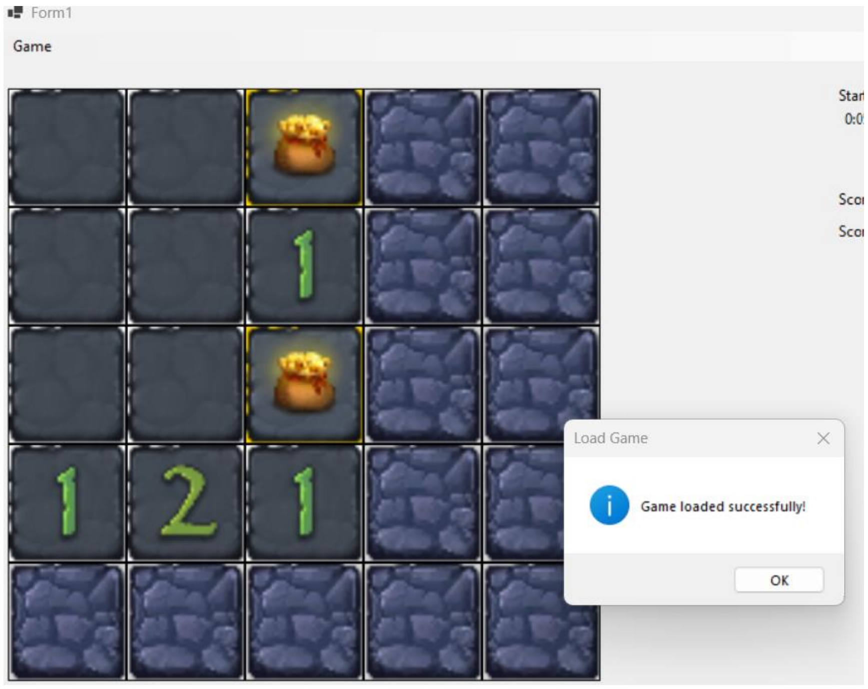


In this screenshot we are demonstrating how we have added for the ability to receive "Rewards" (Gold) randomly while clicking through cells. From receiving the reward, the user can now right click twice on a cell to use the reward and reveal the cell for 5 seconds to see the state of the cell. After 5 seconds the cell will revery back to its original state before the double right click event occurs.

<u>Saving / Loading saved Game</u>


Save Game

Game saved successfully!

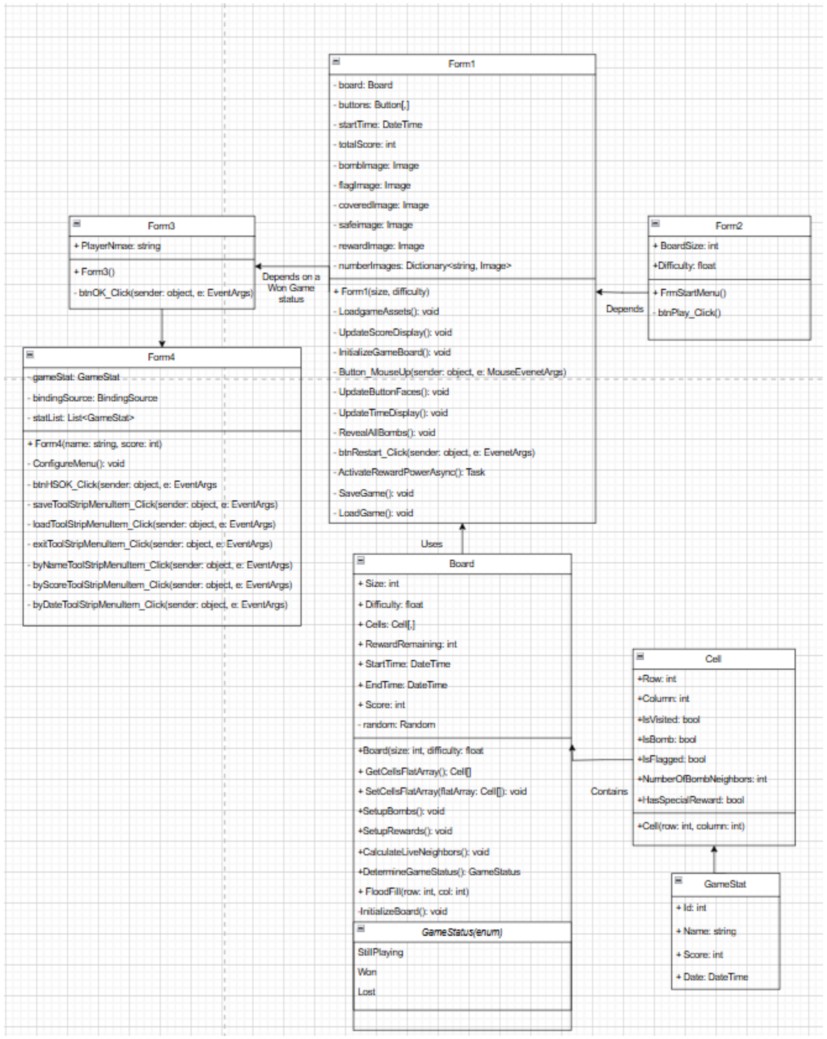


Load Game

Game loaded successfully!

In this screenshot we are demonstrating how we have added the ability for the user to Save their current game state and to be able to load any games that have been saved. We start by serializing the game state to a JSON file then create an anonymous object containing all critical game data and then save with “.mssave” extension. Then we can restore any GameStates from a saved file. First by starting off with the OpenFileDialog to select save file. Then we parse the JSON using System.Text.Json. We will reconstruct the board with the original dimensions and difficulty as well as restore all the cell states. Will then adjust the timer for the accurate game duration then rebuilds the GUI to match saved state.

UML


Form1
- board: Board
- buttons: Button[,]
- startTime: DateTime
- totalScore: int
- bombImage: Image
- flagImage: Image
- coveredImage: Image
- safeimage: Image
- rewardImage: Image
- numberImages: Dictionary<string, Image>
+ Form1(size, difficulty)
- LoadgameAssets(): void
- UpdateScoreDisplay(): void
- InitializeGameBoard(): void
- Button_MouseUp(sender: object, e: MouseEvenetArgs)
- UpdateButtonFaces(): void
- UpdateTimeDisplay(): void
- RevealAllBombs(): void
- btnRestart_Click(sender: object, e: EvenetArgs)
- ActivateRewardPowerAsync(): Task
- SaveGame(): void
- LoadGame(): void
Form3
+ PlayerNmae: string
+ Form3()
- btnOK_Click(sender: object, e: EventArgs)
Depends on a Won Game status
Form2
+ BoardSize: int
+Difficulty: float
+ FrmStartMenu()
- btnPlay_Click()
Depends
Form4
- gameStat: GameStat
- bindingSource: BindingSource
- statList: List<GameStat>
+ Form4(name: string, score: int)
- ConfigureMenu(): void
- btnHSOK_Click(sender: object, e: EventArgs
- saveToolStripMenuItem_Click(sender: object, e: EventArgs)
- loadToolStripMenuItem_Click(sender: object, e: EventArgs)
- exitToolStripMenuItem_Click(sender: object, e: EventArgs)
- byNameToolStripMenuItem_Click(sender: object, e: EventArgs)
- byScoreToolStripMenuItem_Click(sender: object, e: EventArgs)
- byDateToolStripMenuItem_Click(sender: object, e: EventArgs)
Uses
Board
+ Size: int
+ Difficulty: float
+ Cells: Cell[,]
+ RewardRemaining: int
+ StartTime: DateTime
+ EndTime: DateTime
+ Score: int
- random: Random
+Board(size: int, difficulty: float
+ GetCellsFlatArray(): Cell[]
+ SetCellsFlatArray(flatArray: Cell[]): void
+SetupBombs(): void
+SetupRewards(): void
+CalculateLiveNeighbors(): void
+DetermineGameStatus(): GameStatus
+ FloodFill(row: int, col: int)
-InitializeBoard(): void
GameStatus(enum)
StillPlaying
Won
Lost
Contains
Cell
+Row: int
+Column: int
+IsVisited: bool
+IsBomb: bool
+IsFlagged: bool
+NumberOfBombNeighbors: int
+HasSpecialReward: bool
+Cell(row: int, column: int)
GameStat
+ Id: int
+ Name: string
+ Score: int
+ Date: DateTime

Questions

1. What was challenging?

   I would have to say that one of the most challenging aspects was implementing the special reward cell functionality while maintaining clean, asynchronous code in a Windows Forms environment. Trying to ensure that the peek feature worked correctly, by temporarily revealing a cell for 5 seconds before reverting back to the cell's previous state, required careful handling of event subscriptions and task delays without blocking the UI thread.

2. What did you learn?

   I would have to say that through this project I learned how to bridge asynchronous programming with Windows Forms event driven architecture, particularly for time-based interactions. With this project we have reinforced the importance of proper event handler management, adding/removing subscriptions, and thread safety in UI updates. It also deepened my understanding of game state management, from reward mechanics to flood-fill algorithms for revealing empty cells.

3. How would you improve on the project?

   I would have to say to improve on this project I would refactor the reward system into a separate class to decouple it from the UI logic, making it easier to test and extend. Also, I would probably try to add visual feedback, like animations or sound effects, which would enhance the player's experience. The code could also benefit from dependency injection for assets (images) and a more robust error-handling system for file operations (save/load).

4. How can you use what you learned on the job?

   From what I have learned in the project, the async/await patterns and event management techniques can be directly applicable to enterprise applications, such as handling API calls or background tasks in desktop apps. The state management lessons translate to workflow systems, essentially, tracking multi-step processes. Additionally, the focus on clean separation of UI and game logic aligns with modern architectures like MVVM, which I could apply to front-end projects.

<u>Day-to-Day</u>

Monday

Start:5pm End: 6:30pm Started on Milestone 6

Tuesday

Nothing

Wednesday

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

Thursday:

Start:6pm End:9pm Continued on Milestone 6

Friday

Start:8pm End:10pm Activity: Milestone 6

Start: 10pm End:11pm Activity: Activity DQ2

Saturday

Start: 5pmEnd:6pm Activity: Tutoring session

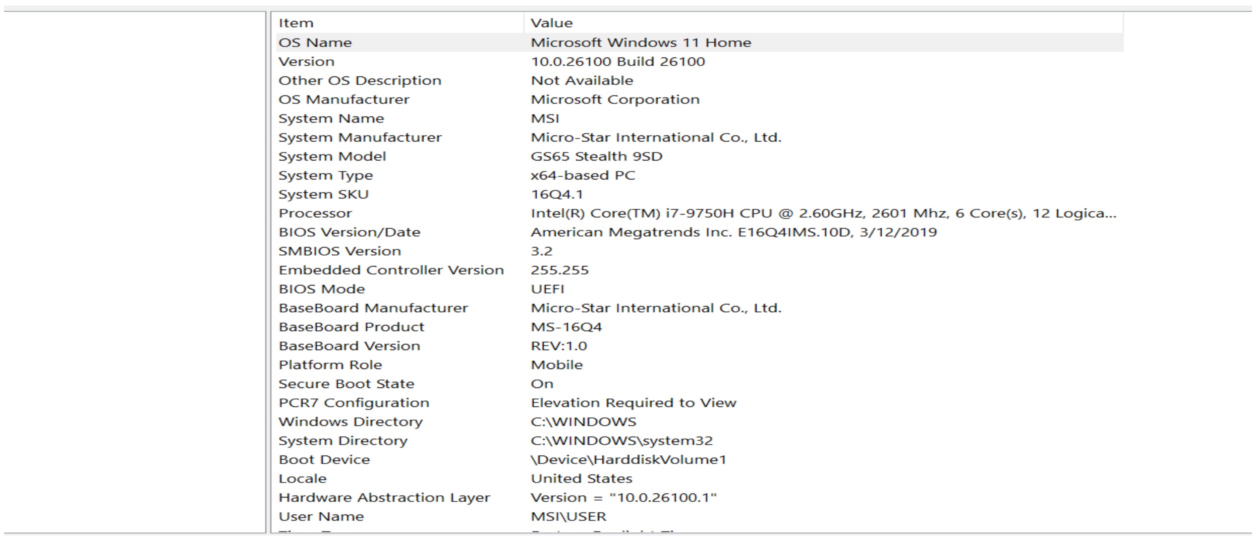
Start:8pm End:10pm Activity: Milestone 6

Sunday

Start: 5pmEnd:6pm Activity: Tutoring session

Start:6pm End:8pm Activity: Continued Milestone 6 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 clicking a reward cell activates peek mode	1. Launch Minesweeper Game 2. Identify and click a cell containing a reward 3. Observe message prompt 4.Click any unrevealed cell	- Board with at least 1 reward cell - Unrevealed non-reward cell to peek	- Reward cell is consumed after click - “Reward activated!” massage appears - Peeked cell shows contents for 5 seconds, then reverts back	Peeks at cell for 5 seconds then reverts to “covered Image”	Pass

Test2	Verify peek last 5 seconds	1. Activate reward 2. Click an unrevealed cell 3. Time visibility duration	- Cell with bomb to peek - Stopwatch	- Cell shows bomb image for exactly 5 seconds - Automatically covers again after duration	Cell is revealed for 5 seconds then reverts to its original Image	Pass
Test3	Verify score updates when reward is collected	1. Note initial score 2. Click reward cell 3. Check score change	Board with known reward cell position	- Score increases by expected value (50) - RewardRemaining counter decrements	Score increases by 50 points and there is one less RewardRemaining	Pass

<u>Programming Conventions</u>

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 activates a special reward cell in Minesweeper to temporarily reveal the contents of any hidden cell.

Primary Actor: Player

Goals:

1. Discover and activate a reward cell
2. Use the peek power to reveal a hidden cell temporarily
3. Strategically use this information to progress in the game

Stakeholders:

- Game Developer
- Players
- Testers
- Ui/UX Designers

Pre-conditions:

1. Game is initialized and in progress
2. At least one unactivated reward cell exists on the board
3. Player has not yet used the current reward cell

Post-Conditions:

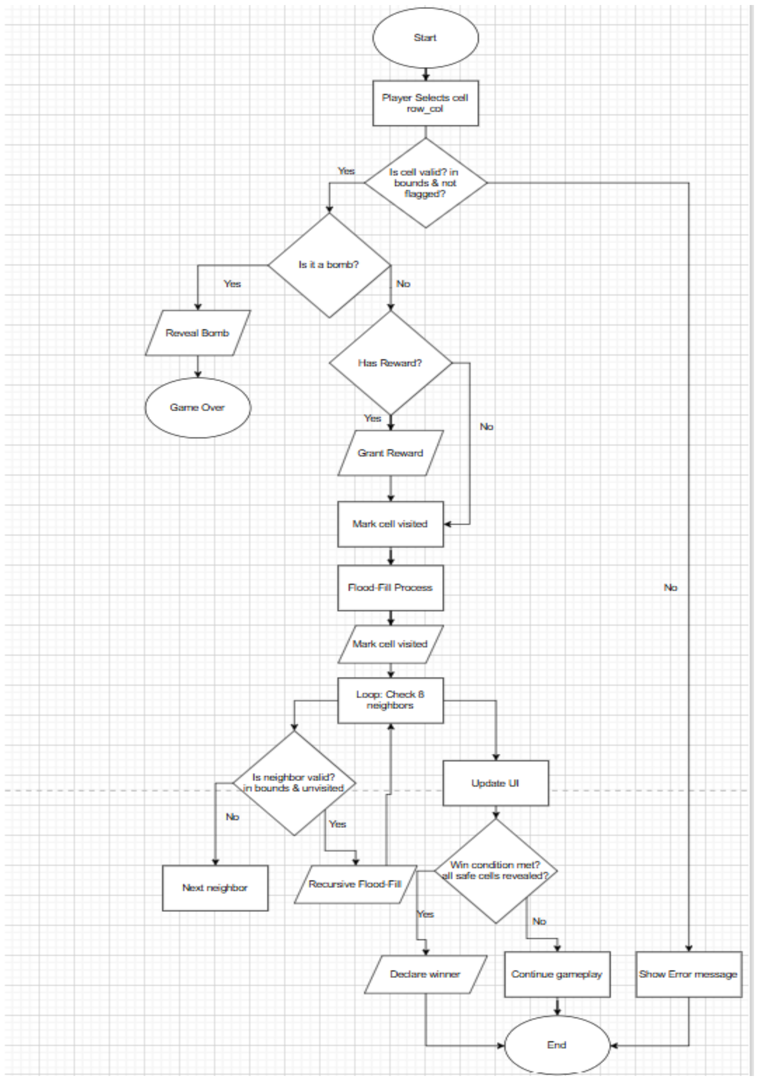
- Reward cell is consumed after activation
- Peeked cell returns to hidden state after 5 seconds
- Game state is preserved (no permanent reveals)

Basic Flow:

1. Player clicks on a covered cell containing a special reward
2. System displays "Reward Activated!" notification
3. System enters 'Peek Mode"
4. Player clicks any unrevealed, unflagged cell
5. System immediately reveals the cell's content (bomb/number/empty)
6. After 5 second, system automatically re-hides the cell
7. Game continues normally

Alternative Paths:

1. Player tries to peek a flagged cell
    - Player clicks a flagged cell
    - System ignores the click (cell remains unchanged)
    - Peek power remains available for another cell
2. Player clicks revealed cell during peek mode
    - Player clicks an already revealed cell
    - System ignores the click
    - Peek power remains available
3. Multiple reward cells exist
    - Player activates a second reward cell while first peek is active
    - System queues the reward (only one peek active at a time)



Bug Report

Nothing at this time.