

Peyton Wolf

3/23/2025

Cst-250 Programming in C# ||

Mark Smithers

Milestone 1

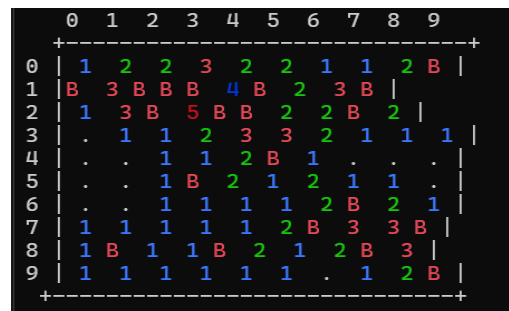
Loom Screencast Videos

1. <https://www.loom.com/share/c5427bb6b4c94f16b0b1c0a92c87a705?sid=f2789b97-3067-4817-a871-a4f25f521a64>
2. <https://www.loom.com/share/f27fc55e5bb147c4a980eab54c994296?sid=d7a64674-7f70-4da2-a6a6-9d27253d9c12>

GitHub Link

<https://github.com/KnoxHighStax/CST250/tree/main/Milestone%201>

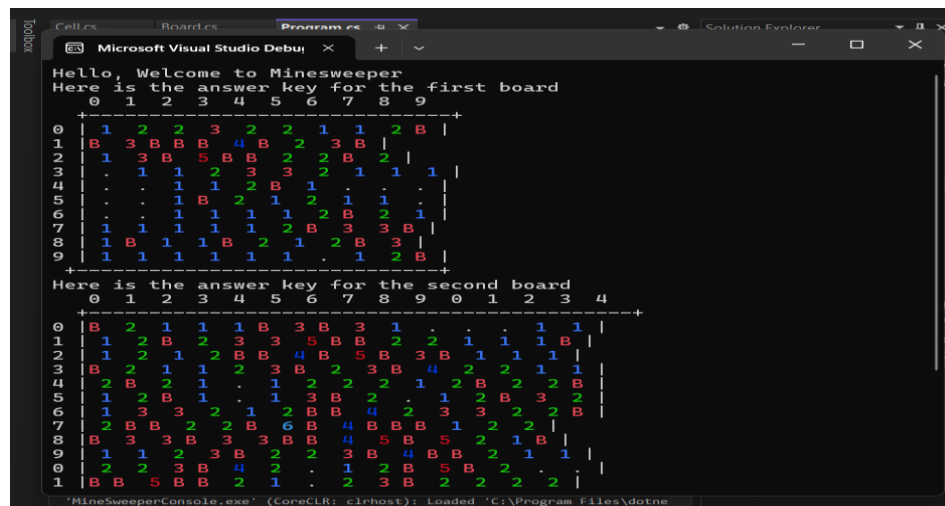
Creating Cell and Board Classes



	0	1	2	3	4	5	6	7	8	9
0	1	2	2	3	2	2	1	1	2	B
1	B	3	B	B	B	4	B	2	3	B
2	1	3	B	5	B	B	2	2	B	2
3	.	1	1	2	3	3	2	1	1	1
4	.	.	1	1	2	B	1	.	.	.
5	.	.	1	B	2	1	2	1	1	.
6	.	.	1	1	1	1	2	B	2	1
7	1	1	1	1	1	2	B	3	3	B
8	1	B	1	1	B	2	1	2	B	3
9	1	1	1	1	1	1	.	1	2	B

This screenshot is just demonstrating the class library that will contain the Cell and Board class and how it will be displayed.

Console Application



This screenshot is just what the overall console application will look like.

Creating Minesweeper Application

```
Hello, Welcome to Minesweeper
Here is the answer key for the first board
  1  2  3  4  5  6  7  8  9 10
+---+
1 | . | 1 | B | 1 | . | . | . | . | . |
+---+
2 | . | 1 | 1 | 1 | . | . | . | . | . |
+---+
3 | . | . | . | . | 1 | 1 | 2 | 1 | 1 | . |
+---+
4 | 1 | 1 | 1 | . | 1 | B | 2 | B | 1 | . |
+---+
5 | 1 | B | 3 | 2 | 2 | 1 | 2 | 1 | 1 | . |
+---+
6 | 1 | 2 | B | B | 1 | . | . | . | . | . |
+---+
7 | . | 2 | 3 | 3 | 2 | 1 | 1 | . | . | . |
+---+
8 | 1 | 2 | B | 1 | 1 | B | 1 | . | . | . |
+---+
9 | 2 | B | 3 | 1 | 1 | 1 | 1 | . | . | . |
+---+
10 | 2 | B | 2 | . | . | . | . | . | . | . |
+---+
Here is the answer key for the second board
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
+---+
1 | 1 | B | 2 | B | 1 | . | . | . | 1 | 1 | 2 | B | 2 | B |
+---+
2 | 3 | 3 | 4 | 2 | 2 | . | . | . | 1 | B | 2 | 1 | 2 | 4 | B |
+---+
3 | B | B | 3 | B | 3 | 1 | 2 | 1 | 2 | 1 | 2 | 2 | 2 | 3 | B |
+---+
4 | 2 | 2 | 3 | B | 3 | B | 3 | B | 3 | 1 | 2 | B | B | 2 | 1 |
+---+
5 | . | . | 1 | 1 | 2 | 1 | 3 | B | 3 | B | 2 | 2 | 2 | 1 | . |
+---+
6 | . | . | . | . | 1 | 1 | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | . |
+---+
7 | 1 | 1 | 1 | . | 1 | B | 1 | . | . | . | . | 1 | B | 1 | . |
+---+
8 | 1 | B | 1 | . | 2 | 3 | 3 | 1 | . | . | . | 1 | 1 | 1 | . |
+---+
9 | 1 | 1 | 1 | . | 1 | B | B | 2 | 1 | 1 | 1 | 1 | . | . | . |
+---+
10 | 1 | 1 | 1 | . | 1 | 2 | 3 | B | 1 | 1 | B | 1 | . | . | . |
+---+
11 | 1 | B | 2 | 1 | . | . | 1 | 1 | 1 | 2 | 2 | 2 | . | . | . |
+---+
12 | 1 | 3 | B | 2 | . | 1 | 1 | 1 | . | 1 | B | 1 | . | . | . |
+---+
13 | 1 | 3 | B | 2 | . | 1 | B | 2 | 1 | 1 | 1 | 1 | . | . | . |
+---+
14 | B | 2 | 1 | 1 | . | 1 | 2 | B | 1 | 1 | 1 | 1 | 1 | 2 | 2 |
+---+
15 | 1 | 1 | . | . | . | . | 1 | 1 | 1 | 1 | B | 1 | 1 | B | B |
+---+
```

In this screenshot we are showing how we are referencing the class library to create an instance of the Board class in the “Program.cs” file. The overall main part of the program will be utilizing the “PrintAnswers(Board board)” method that will help to display the contents of the Board itself. We have print two board total showing its versatility, one a 10 by 10 board size and a 15 by 15 board size.

Setting up Bombs, Neighbors and a . For Nothing

B	2	1	B	B	2	1
B	2	1	2	3	B	1
1	1	.	1	2	2	1
1	.	.	1	B	1	1
3	2	2	2	2	2	2
B	B	2	B	1	1	B
2	3	4	3	2	1	1

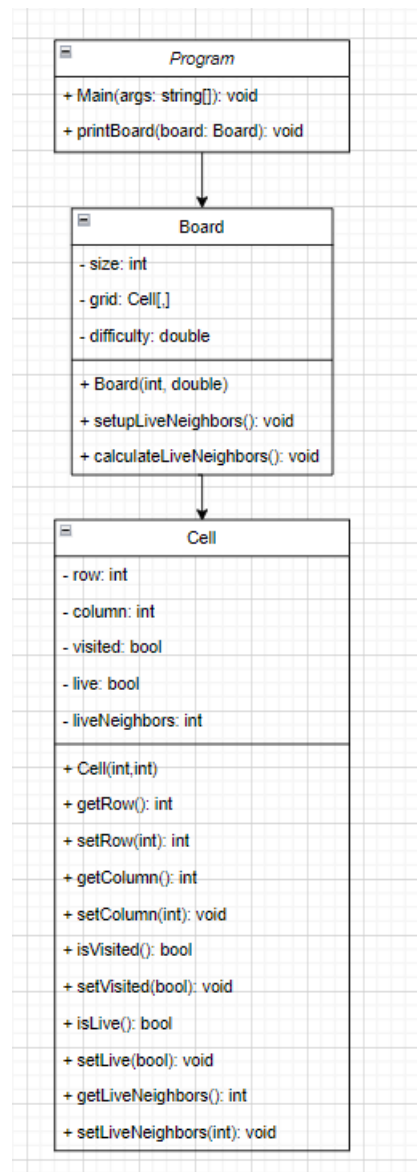
In this screenshot you can see how we have set up the Bombs to be display by the letter “B” and we also are printing the numbers in the cells by the bombs based on the number of neighbors that cell has. We have dividers between each of the cells to keep everything nice and neat and to keep things looking clean! Then we added some color to each of the symbols to make things look a bit better.

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

In this project we have demonstrated **object-oriented programming** concepts in a Minesweeper game implementation. With this fun new project, we are also introducing new things like the **class library**; which in our case are reusable components that contain the **Board** and **Cell** class. We created a private method called “CalculateNumberOfBombsNeighbors” which uses **encapsulation** for hiding internal details and learned a bit about **composition** being the **Board** class that contains a 2D array of **Cell** objects. We started this development process with a **UML**, so we translated the UML diagram into actual code. Within the application we created a **console application** structure using “Program.cs” as our entry point. We also utilize **grid algorithms** for handling our bomb placement and neighbor counting, while emphasizing **debugging techniques** like null checks and boundary validation to ensure that our application is running correctly and not coming in counter with any unexpected errors. We also have our **user interface** principles like color coded output with “Console.ForegroundColor” and grid formatting with borders and accurate position. With this project we showed proper

separation of concerns between the game logic itself (being our class library) and the display for the user (being the console application).

UML



Questions

1. What was challenging?

I would have to say that the most difficult part of this application was getting the visual placements for the board set up. There was a lot of running the application to see what it looked like then had no small adjustments to the placements of the “.”, “|”, “+”, etc to get them in the right location to make sure that the board is looking visually good.

2. What did you learn?

Through this project I would have to say I have learned a lot when it comes to my understanding of class design, encapsulation and 2D array manipulation. The neighbor counting algorithm taught me an efficient way to transverse and analyze grid-based data structures, which is common for in game development. I also gained knowledge on how randomized systems behave, like bomb placement and console formatting techniques, like adding text color.

3. How would you improve on the project?

I could enhance this project by adding input validation for board size/difficulty settings and implementing a more dynamic difficulty system. The console could also be improved with interactive cell selection using cursor navigation besides coordinate input. The bomb-counting algorithm could also be optimized by pre calculating neighbor offsets.

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

With the problem-solving approaches learned in this project, they can translate directly to any type of professional work, especially in grid-based algorithms, which are useful in data visualization or game development. The class design principles apply to an object-oriented system and the separation of UI from business logic, which mirrors modern architecture patterns like MVC.

Day-to-Day

Monday

Start:6pm End:9pm Activity: Read Chapter 5

Start: End: Activity:

Start: End: Activity:

Tuesday

Nothing

Wednesday

Start:6pm End:8pm Activity: Complete Reading Chapter 5

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

Thursday:

Start:6pm End:9pm Activity: Started on Assignment 1/CarStore

Start: 9pmEnd:10pm Activity: Responded to classmates

Friday

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

Start:8pm End:10pm Activity: Started on Milestone 1

Saturday

Start: 12pm End:4pm Activity: Completed Assignment 1/CarStore

Sunday

Start:11am End:3pm Activity: Complete Milestone 1 Project

Start:6pm End:8pm Activity: Complete Milestone 1 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 bomb placement for easy difficulty (10% bombs)	1. Create Board(10). 2.Set Difficulty=1. 3.Call SetupBombs(). 4.Count bombs.	Size=10, Difficulty=1	About 10 bombs	9-12 bombs (randomized)	Pass
Test2	Validate neighbor bomb counting (manual bomb placement)	1.Create Board(3). 2.Set bombs at (0,0) and (2,2). 3.Call CountBombs() Nearby(). 4.Check neighbors	Bombs at (0,0) and (2,2)	(0,1)=1 (1,1)=2 (2,1)=1	(0,1)=1 (1,1)=2 (2,1)=1	Pass
Test3	Edge case: no bombs on the board (Difficulty = 0)	1.Create Board (5). 2.Set Difficulty=0. 3.Call SetupBombs	Size=5, Difficulty=0	All cells show 0 neighbors	All cells show 0 neighbors	Pass

		()4. Verify neighbors.				
--	--	------------------------	--	--	--	--

Programming Conventions

1. We will be using the naming convention of camelCase for variables and parameters, like for “isVisited” and “numberOfBombs” and PascalCase for methods and classes, like for “CountBombsNearby(), GameBoard making sure that constants in UPPER_CASE (MAX_BOARD_SIZE).
2. When it comes to our code's readability through utilizing a 4-space indentation and ensuring that we have proper brace placement and strategic spacing around operators and commas.
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

Description: A player/user will open the game application to be able to view the current game board and view examples on randomly generated board in a Minesweeper game. Showing the user where bombs are generated, and the placement of any live cell neighbors will be resembled with color making it easier for the user to review.

Primary Actor: Player/Myself/Instructor

Goals: - Show bomb placement

- Show bomb neighbors

- Give bomb neighbors color

Stakeholders: - Myself/Developer

Pre-conditions: - Game board initialized

- Difficulty Set

- Bombs randomly placed

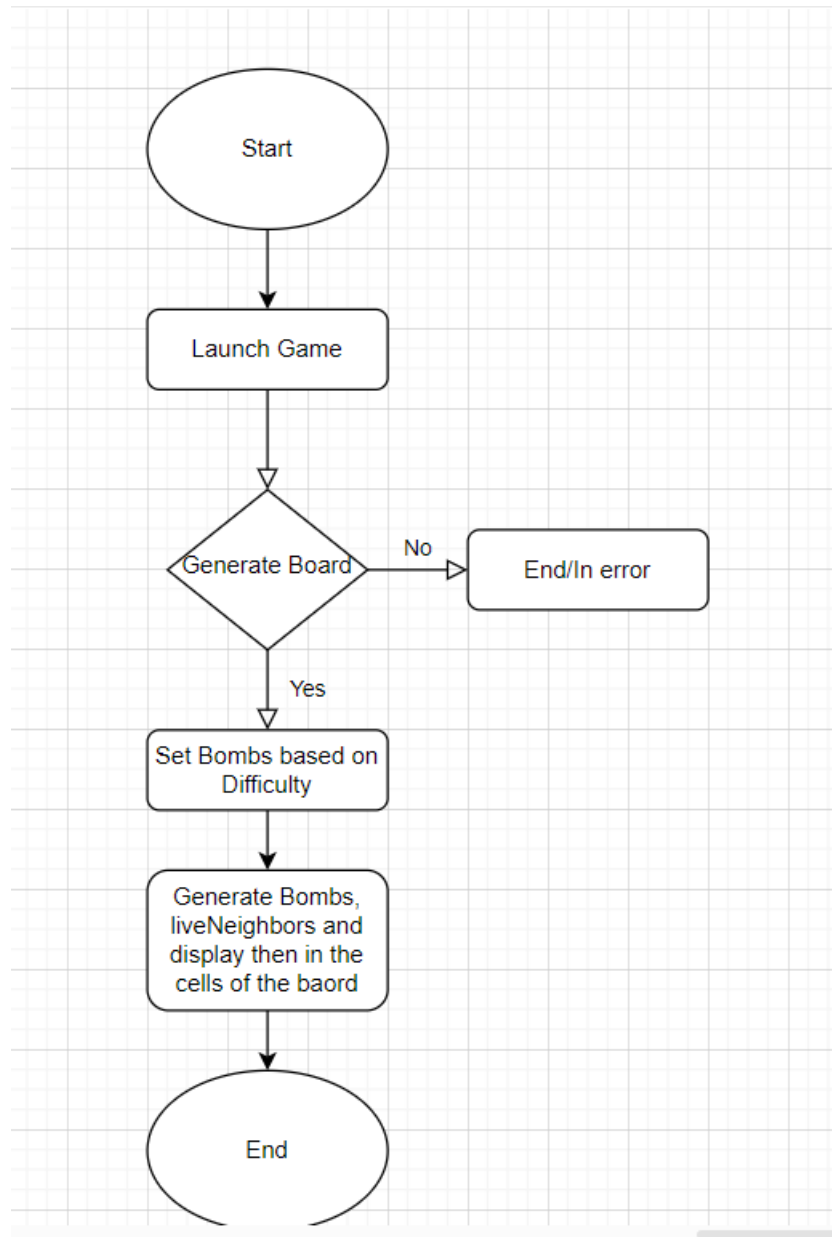
Post-Conditions: None

Basic Flow:

1. Player launches Minesweeper application.
2. The system will then generate 2 game boards to be displayed.
3. One size 10 the other 15.
4. Will next randomly generate bombs and place them.
5. Then generate the liveNeighbors around each of the bombs and display them.
6. Any empty cells will then be filled with a ".".

Alternative Paths:

1. Currently the Flow of the program has no alternative passed since all the back-end logic has no ability (currently) to accept user input



Bug Report

Nothing at this time.