



CST-250 Project Milestone Guide

Directions: Throughout this course, students will incrementally design and build a Minesweeper Game.

Contents

Introduction.....	2
What is Minesweeper?.....	2
Milestone 1: Console Application.....	3
Deliverables	11
Milestone 2: Interactive Playable Version	12
Game Loop.....	12
The Reward	14
Deliverables	15
Milestone 3: Using Recursion	16
Deliverables	17
Milestone 4: GUI Version	18
Milestone 5: GameStat Class	25
Milestone 6: Game Extensions	30

Introduction

What is Minesweeper?



Figure 1 Minesweeper online game.

Minesweeper is a classic puzzle game that has been a common feature on personal computers for decades. The game combines logic and strategy: clear a grid of squares without hitting hidden mines.

The gameplay involves a rectangular grid of tiles, each of which may conceal a mine. When a player clicks on a tile that does not contain a mine, the game reveals a number on that tile, indicating how many adjacent tiles contain mines. This clue helps the player deduce which neighboring tiles are safe to click next.

Tiles adjacent to a mine can be marked with a flag to help the player keep track of suspected mines and prevent accidental clicks. The game continues until either all non-mine tiles are revealed (resulting in a win) or a mine is triggered (resulting in a loss).

Minesweeper is not only a game of chance but a test of deductive reasoning and quick thinking. It offers various levels of difficulty, typically altering the size of the grid and the number of mines, allowing players from beginners to seasoned strategists to find a challenging yet manageable level.

Play the game at [here](#) until you can regularly win the game. It will be impossible to create a Minesweeper app unless you are familiar with its rules.

Milestone 1: Console Application

Overview

In this milestone, students will create three classes: Cell, Board, and Program.

1. UML Class Diagrams

Display the class properties and methods. A game cell should have the following properties:

- a. **Row** and **Column**. These should initially be set to -1.
- b. **IsVisited** Boolean value. This should initially be set to false. During the game, the player will "visit" or "clear" a square, changing the isVisited state from false to true.
- c. **IsBomb** Boolean value. This should initially be set to false. "isLive" set to true will indicate that the cell is a "live bomb" cell. That is, a bomb is planted on the cell.
- d. **IsFlagged** Boolean value. This should initially be set to false. "isFlagged" set to true will indicate that the player has placed a flag on the cell. It may or may not be accurate to say that a flagged cell contains a bomb.
- e. **NumberOfBombNeighbors**. The number of neighbors that contain bombs. This should initially be set to 0 but will change after running a special method that counts the surrounding bombs.
- f. **HasSpecialReward**. Depending on your preference, a "special reward" cell can give the player a bonus when it is revealed. Choose one of these examples to implement as a reward. Some of these rewards sound fun but could actually make the game too easy to play, so choose carefully on what you want to give away with the special reward.
 - i. **Hint** – Allows the player to use a "Show Bomb" option, revealing one bomb chosen at random.
 - ii. **Time Freeze** – Stops the play clock for a period of time.
 - iii. **Bomb Defuse Kit** – Allows the player a "free space" click. If a player uses this on a space, a bomb will not end the game if present.
 - iv. **Bomb Squad** – Gives the player a bomb-defuse suite for 10 seconds of immunity. Any bombs clicked in the next 10 seconds will not end the game.
 - v. **Undo** – Provides the player with a mulligan, or a do-over.

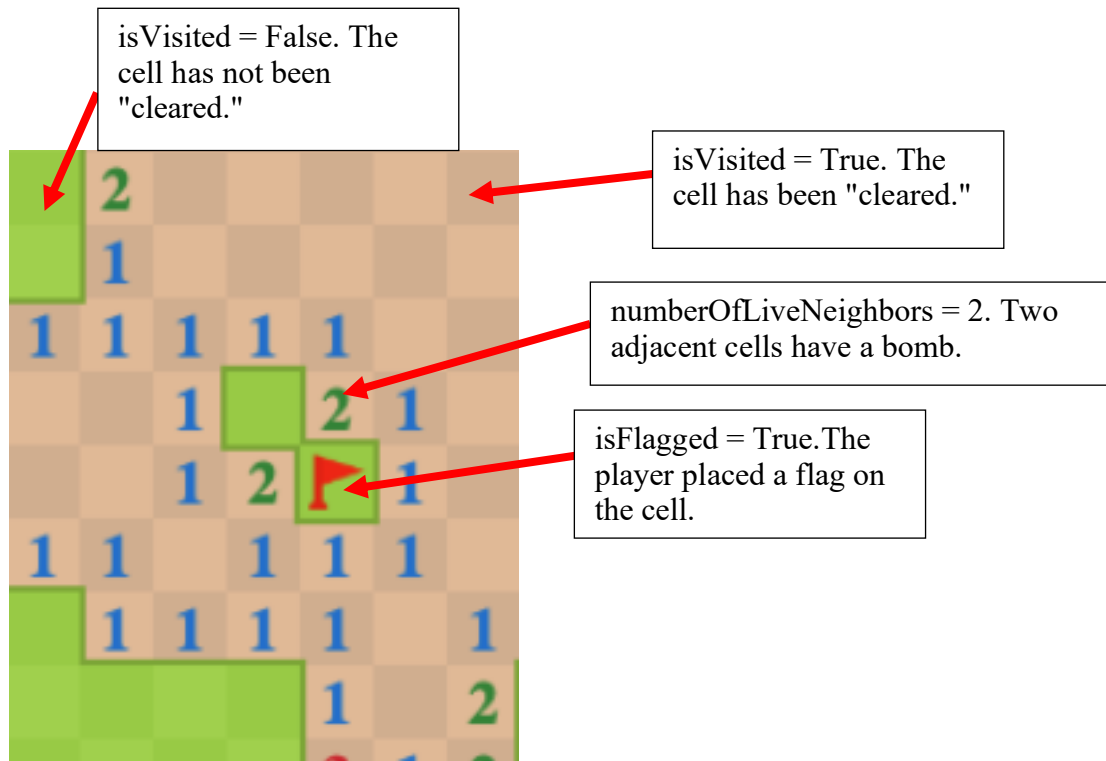


Figure 2 Cell properties will be displayed in the UI through various forms. The Cell class itself contains data in the form of integers and Boolean values.

- g. Create a class that models a game board. A game board should have the following **properties**:
- Size**. The board will be square, where the size includes the dimensions of both the length and width of the board.
 - StartTime** – DateTime type. Stores the exact second that a game begins.
 - EndTime** – DateTime type. Stores the exact second that a game ends.
 - Cells** – The grid will be a 2-dimensional array of the type cell.
 - Difficulty** – A number that represents how challenging it is to play the game. Decide from these options on how you want to determine the number of bombs in the game. Any one of these options will work but will be programmed differently.

Difficulty Levels – Integer (1, 2, 3) representing "Easy", "Medium" or "Hard". These levels will determine some number of bombs to be placed on the board.

Percentage – Float (0.0 to 0.25) representing the probability that a cell will be assigned a bomb. Difficulty – 0.1 would indicate that 10% of cells will get bombs placed on them.

Count of Bombs – Integer (10, 20, 30, etc.) representing the total number of bombs to be placed on the board.

- RewardsRemaining** – Integer value of the number of special rewards the player has collected. Initially start with value of zero.

- vii. **GameState** – Enum property to indicate the state of the game (e.g., **InProgress, Won, Lost**). This can be used to handle different outcomes and update the UI accordingly.
- h. The Game board will have some methods:
 - i. The **constructor** for the Board should have a single parameter to set the size of the Grid. In its constructor, the Grid should be initialized so that a Cell object is stored at each location.
 - ii. **SetupBombs** – A method to randomly initialize the grid with live bombs. The method should utilize the Difficulty property to determine which cells in the grid will be set to "live" status.
 - iii. **CountBombsNearby** – A method to calculate the live neighbors for every cell on the grid. A cell should have between 0 and 8 live neighbors. If a cell itself is "live," then you can set the neighbor count to 9.



Figure 3 Game play reveals the "LiveNeighbor" values as color coded integers. In this case, most of the bombs have been successfully flagged by the player. 4 is the maximum number of bombs adjacent to any square in this game. Many squares have zero bombs adjacent which is represented by a blank grey square, not the number 0.

- iv. **DetermineGameState** – Based on the state of the game, the method will return "GameLost", "GameWon" or "StillPlaying" status. We won't implement this method until later milestone assignments.
- v. **UseSpecialBonus** – Depending on the "Special Reward" definition you choose in the Cell class, this method will put the reward into practice. We won't implement this method until later milestone assignments.

- vi. **DetermineFinalScore** – This method will use a formula to calculate a score. The formula will be your invention taking into account the difficulty level, the time elapsed, the size of the board, and special bonuses used. We won't implement this method until later milestone assignments.

```

7  namespace MinesweeperClasses
8  {
9      public class Board
10     {
11         public int Size { get; set; }
12         public float Difficulty { get; set; }
13         public Cell[,] Cells { get; set; }
14         public int RewardsRemaining { get; set; }
15         public DateTime StartTime { get; set; }
16         public DateTime EndTime { get; set; }
17         public enum GameStatus { InProgress, Won, Lost }
18
19         Random random = new Random();
20
21         public Board(int size, float difficulty)
22         {
23             Size = size;
24             Difficulty = difficulty;
25             Cells = new Cell[size, size];
26             RewardsRemaining = 0;
27             InitializeBoard();
28         }
29
30         private void InitializeBoard()
31         {
32             SetupBombs();
33             SetupRewards();
34             CalculateNumberOfBombNeighbors();
35             StartTime = DateTime.Now;
36         }
37         // used when player selects a cell and chooses to play the reward
38         public void UseSpecialBous() { }
39
40         // use after game is over to calculate final score
41         public int DetermineFinalScore() { return 0; }
42
43         // helper function to determine if a cell is out of bounds
44         private bool IsCellOnBoard(int row, int col) ...
45
46         // use during setup to calculate the number of bomb neighbors for each cell
47         private void CalculateNumberOfBombNeighbors() ...
48
49         // helper function to determine the number of bomb neighbors for a cell
50         private int GetNumberOfBombNeighbors(int i, int j) ...
51
52         // use during setup to place bombs on the board
53         private void SetupBombs() ...
54
55         // use during setup to place rewards on the board
56         private void SetupRewards() ...
57
58         // use every turn to determine the current game state
59         public GameStatus DetermineGameState() ...
60     }
61 }

```

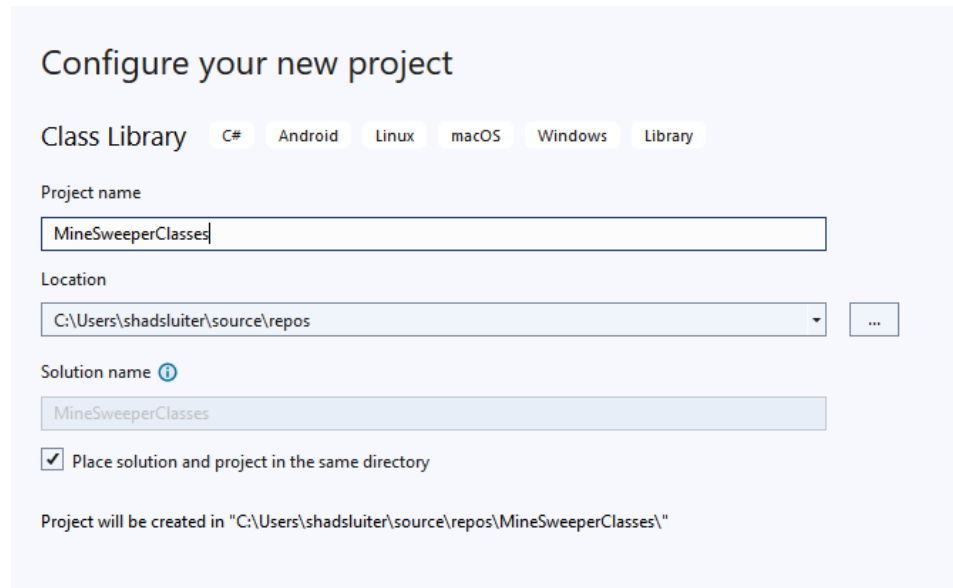
Figure 4 Board class with methods. Some methods are optional and are used to support other methods.

2. UML

- a. Show a "Composition" relationship between the classes.
- b. You can use any drawing software such as Visio or Draw.io to create the diagrams. Save the resulting picture in a Microsoft Word document.

Create a Project

1. Create a Class Library Project to hold the Board and Cell classes.



Configure your new project

Class Library C# Android Linux macOS Windows Library

Project name
MineSweeperClasses

Location
C:\Users\shadsluiter\source\repos

Solution name ⓘ
MineSweeperClasses

☒ Place solution and project in the same directory

Project will be created in "C:\Users\shadsluiter\source\repos\MineSweeperClasses\"

Figure 5 Creating a new Class Library project.

2. Based on the UML diagram create the Board.cs and Cell.cs files in the MineSweeperClasses project.

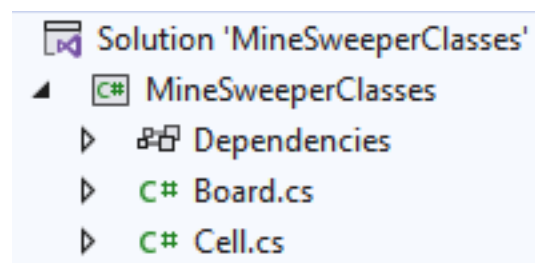


Figure 6 Two classes are in the project.

3. Create a console application project in the same solution.

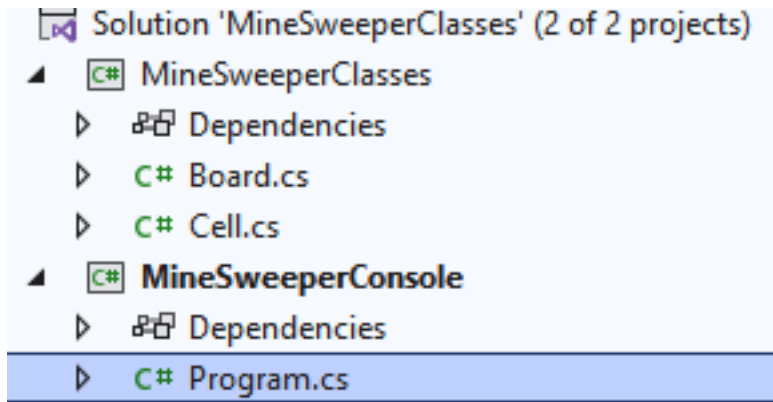


Figure 7 Second project contains a console application.

4. Code the Methods

The Program.cs class should be the console app that drives the application. This is the class that should contain a main() method.

- Create a project reference to the class library class.
- Create an instance of the Board class in Program.cs
- The main program should have a **PrintAnswers(Board board)** method that displays the contents of the Board.
- Ensure that the board can print a variety of sizes of game boards. Test the output with a board size 10 or less and a second board greater than size 10.

```
static void Main(string[] args)
{
    Console.WriteLine("Hello, welcome to Minesweeper");
    // size 10 and difficulty 0.1
    Board board = new Board(10, 0.1f);
    Console.WriteLine("Here is the answer key for the first board");
    PrintAnswers(board);

    // size 15 and difficulty 0.15
    board = new Board(15, 0.15f);
    Console.WriteLine("Here is the answer key for the second board");
    PrintAnswers(board);
}
```

Figure 8 Main method testing for two different board sizes.

- Print "B" for bombs. Print a number if there are any neighboring bombs. Print a "." when there are no neighbor bombs.
- Print column and row numbers. Print divider lines to display an outline around each cell.
- Use Console.ForegroundColor to change the print color for each symbol. For example, this sets the print color to Red. Console.ForegroundColor = ConsoleColor.Red;

```

Microsoft Visual Studio Debug Console
Hello, welcome to Minesweeper
Here is the answer key for the first board
  0  1  2  3  4  5  6  7  8  9
+---+---+---+---+---+---+---+---+---+---+
0 | . | . | . | . | . | . | 1 | B | B | 2 |
+---+---+---+---+---+---+---+---+---+---+
1 | . | . | . | . | . | . | 1 | 2 | 4 | B | 3 |
+---+---+---+---+---+---+---+---+---+---+
2 | . | . | . | . | . | . | 1 | B | 2 | 2 | B |
+---+---+---+---+---+---+---+---+---+---+
3 | . | . | . | . | . | . | 1 | 1 | 1 | 1 | 1 |
+---+---+---+---+---+---+---+---+---+---+
4 | . | . | . | . | . | 1 | 2 | 3 | 2 | 1 | . |
+---+---+---+---+---+---+---+---+---+---+
5 | 1 | 1 | . | . | 2 | B | B | B | 1 | . |
+---+---+---+---+---+---+---+---+---+---+
6 | B | 1 | . | . | 2 | B | B | 3 | 1 | . |
+---+---+---+---+---+---+---+---+---+---+
7 | 1 | 1 | . | . | 2 | 3 | 3 | 2 | 1 | 1 |
+---+---+---+---+---+---+---+---+---+---+
8 | . | . | . | . | 1 | B | 2 | 3 | B | 2 |
+---+---+---+---+---+---+---+---+---+---+
9 | . | . | . | . | 1 | 2 | B | 3 | B | 2 |
+---+---+---+---+---+---+---+---+---+---+
Here is the answer key for the second board
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
+---+---+---+---+---+---+---+---+---+---+---+---+---+
0 | . | . | . | . | . | . | . | . | . | 1 | 1 | 2 | 2 | 2 | 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
1 | 1 | 1 | . | . | . | . | 1 | 1 | 1 | 1 | B | 3 | B | B | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
2 | B | 1 | 1 | 1 | 1 | . | 1 | B | 1 | 1 | 2 | B | 4 | B | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
3 | 2 | 2 | 2 | B | 3 | 3 | 3 | 2 | 1 | . | 1 | 1 | 2 | 2 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
4 | 1 | B | 3 | 3 | B | B | B | 1 | . | . | . | . | . | 1 | B |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
5 | 2 | 2 | 3 | B | 5 | 5 | 3 | 1 | . | . | . | . | 1 | 2 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
6 | 1 | B | 2 | 2 | B | B | 1 | . | 1 | 1 | 1 | . | 1 | B | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
7 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 1 | 2 | B | 1 | . | 1 | 2 | B |
+---+---+---+---+---+---+---+---+---+---+---+---+---+
8 | . | . | . | . | . | . | 1 | B | 2 | 1 | 1 | . | . | 1 | 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 9 Expected output for two game boards. The first is size 10. The second is size 15.

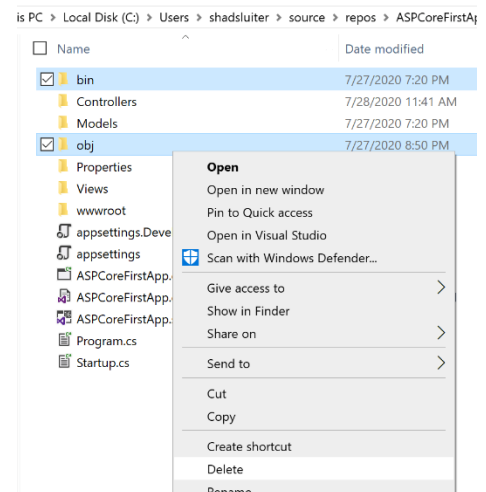
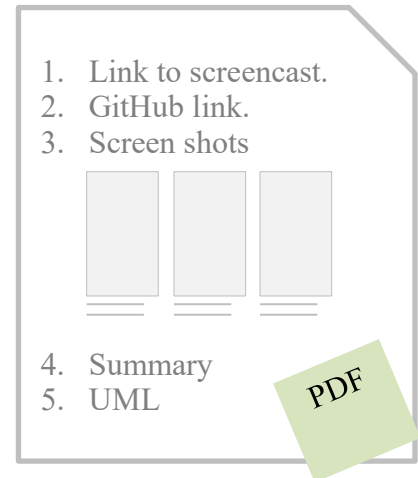
5. The main() method should:
 - a. Create an instance of the Board class.
 - b. Call the Board.SetupBombs and Board.CountBombsNearby commands to initialize the grid.
 - c. Call the PrintAnswers(board) method to display the contents of the grid.

GitHub

1. Review the "GitHub Tutorial," located in Class Resources, to learn how to create a git repository and use a version control system.
2. Submit the current milestone to a GitHub.com private repository.
3. Provide access to your instructor to view the repository.
4. Include the URL for the repository in the project Microsoft Word document.

Deliverables:

1. Create a screencast video of the application.
 - a. Demonstrate the application working per the requirements.
 - b. Show relevant sections of the code, *explaining* the key ideas of this milestone and how they were implemented.
 - c. Past a link to the video in a Microsoft Word document.
2. In the same Word document, add the link to the GitHub repository. Share the repository with your instructor's email address.
3. In the same Word document, add screenshots of the application being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
4. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson. Be sure to explain the key words introduced in this lesson.
5. In the same Word document, paste the UML diagrams for the classes.
6. Convert the Word document to a PDF and attach the document separately from the zip file. Multiple files can be uploaded with an assignment.
7. Submit a ZIP file of the project file. In order to save space, you can delete the bin and obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.



Milestone 2: Interactive Playable Version

Overview

In this milestone, students will create an interactive playable version of the Minesweeper game.

1. Update the UML class diagram based on the new features described below.
2. In **Board.cs**, implement the method **DetermineGameState**. This method will inspect the contents of the Board and return one of three values:
 - **Won** when all non-bomb cells have been either visited or correctly flagged.
 - **Lost** if a bomb cell has been visited.
 - **StillPlaying** if there is at least one cell left to visit.
3. In **Program.cs**, create a method called **PrintBoard** that displays the board during the game play. The only difference between **PrintBoard** and **ShowAnswers** is that **PrintBoard** takes into account the **IsVisited** property of each cell. If a cell is not visited, display a "?" symbol.

Game Loop

In the main method of Program.cs, create a playable version of the game in a while. Use this pseudocode as a guide.

```
ShowAnswers() // technically this is cheating to do this first.
```

```
bool victory = false; // player wins
```

```
bool death = false; // player loses
```

```
// repeat until the game is over
```

```
while (!victory && !death) {
```

1. Ask the user for a row and column number.
2. Ask if this is for "Flag" or "Visit" or "Use Reward."
3. Update the status of the chosen cell according to the move.
4. Determine if victory == true or if death == true
5. Print the grid.

```
} // end game
```

```
Print a victory or failure message
```

```

C:\Users\shadsluiter\source\... x + v
Hello, welcome to Minesweeper
Here is the answer key for the first board
  0  1  2  3
+---+---+---+---+
0 | 1 | 2 | 1 | 1 |
+---+---+---+---+
1 | B | 2 | B | 1 |
+---+---+---+---+
2 | 1 | 2 | 1 | 1 |
+---+---+---+---+
3 | . | . | . | . |
+---+---+---+---+
Here is the current board
  0  1  2  3
+---+---+---+---+
0 | ? | ? | ? | ? |
+---+---+---+---+
1 | ? | ? | ? | ? |
+---+---+---+---+
2 | ? | ? | ? | ? |
+---+---+---+---+
3 | ? | ? | ? | ? |
+---+---+---+---+
Enter the row number
0
Enter the column number
1
Enter 1 to visit the cell, 2 to flag the cell
1
Game in progress
Here is the current board
  0  1  2  3
+---+---+---+---+
0 | 1 | 2 | ? | ? |
+---+---+---+---+
1 | ? | ? | ? | ? |
+---+---+---+---+
2 | ? | ? | ? | ? |
+---+---+---+---+
3 | ? | ? | ? | ? |
+---+---+---+---+
Enter the row number
0
Enter the column number
0
Enter 1 to visit the cell, 2 to flag the cell
1
Game in progress
Here is the current board
  0  1  2  3
+---+---+---+---+
0 | 1 | 2 | ? | ? |
+---+---+---+---+
1 | ? | ? | ? | ? |
+---+---+---+---+
2 | ? | ? | ? | ? |
+---+---+---+---+
3 | ? | ? | ? | ? |
+---+---+---+---+
Enter the row number
0
Enter the column number
1
Enter 1 to visit the cell, 2 to flag the cell
1
Game in progress

```

Figure 10 Game in progress. First the answer key is shown. Then 2 moves were attempted at 0,0 and 0,1.

```

Game in progress
Here is the current board
  0  1  2  3
+---+---+---+---+
0 | 1 | 2 | 1 | 1 |
+---+---+---+---+
1 | ? | 2 | ? | 1 |
+---+---+---+---+
2 | 1 | 2 | 1 | 1 |
+---+---+---+---+
3 | . | . | ? | ? |
+---+---+---+---+
Enter the row number
3
Enter the column number
2
Enter 1 to visit the cell, 2 to flag the cell
1
Game in progress
Here is the current board
  0  1  2  3
+---+---+---+---+
0 | 1 | 2 | 1 | 1 |
+---+---+---+---+
1 | ? | 2 | ? | 1 |
+---+---+---+---+
2 | 1 | 2 | 1 | 1 |
+---+---+---+---+
3 | . | . | . | ? |
+---+---+---+---+
Enter the row number
3
Enter the column number
3
Enter 1 to visit the cell, 2 to flag the cell
1
Congratulations! You won!

```

Figure 11 After placing 14 "visit" moves, the game is over and the player wins!

The Reward

Implement the code to make the reward bonus work properly.

In this example, the cell at 1,1 is a special reward cell. You can see the "r" character was chosen to display the reward.

```
Hello, welcome to Minesweeper
Here is the answer key for the first board
  0  1  2  3
+---+---+---+---+
0 | 1 | B | 2 | B |
+---+---+---+---+
1 | 2 | r | 3 | 1 |
+---+---+---+---+
2 | 1 | B | 2 | 1 |
+---+---+---+---+
3 | 1 | 1 | 2 | B |
+---+---+---+---+
Here is the current board
```

Figure 12 Cell 1,1 was randomly chosen to contain the reward.

When the player visits cell 1, 1 the reward message is shown.

```
Here is the current board
  0  1  2  3
+---+---+---+---+
0 | ? | ? | ? | ? |
+---+---+---+---+
1 | ? | ? | ? | ? |
+---+---+---+---+
2 | ? | ? | ? | ? |
+---+---+---+---+
3 | ? | ? | ? | ? |
+---+---+---+---+
Enter the row number
1
Enter the column number
1
Enter 1 to visit the cell, 2 to flag the cell, 3 use a reward (bomb detector)
1
You found a reward!
```

Figure 13 Visiting 1,1 gets the reward.

On the next turn, the player uses the reward on cell 0,1. The reward allows the player "peek" to see if the cell is a bomb or not.

```

Enter the row number
0
Enter the column number
1
Enter 1 to visit the cell, 2 to flag the cell, 3 use a reward (bomb detector)
3
Is it a bomb? True
Game in progress
Here is the current board
  0  1  2  3
0 | ? | ? | ? | ? |
+---+---+---+---+
1 | ? | 2 | ? | ? |
+---+---+---+---+
2 | ? | ? | ? | ? |
+---+---+---+---+
3 | ? | ? | ? | ? |
+---+---+---+---+
Enter the row number

```

Figure 14 Option 3 is chosen which uses the reward. The player gets information about cell 0,1.

GitHub

1. Review the "GitHub Tutorial," located in Class Resources, to learn how to create a git repository and use a version control system.
2. Create a commit of this Milestone and push it to the GitHub repository.
3. Your instructor will see the updates if you provided access.
4. Include the URL for the repository in the project Microsoft Word document.

Deliverables:

1. Create a screencast video of the application.
 - a. Demonstrate the application working per the requirements.
 - b. Show relevant sections of the code, *explaining* the key ideas of this milestone and how they were implemented.
 - c. Past a link to the video in a Microsoft Word document.
2. In the same Word document, add the link to the GitHub repository. Share the repository with your instructor's email address.

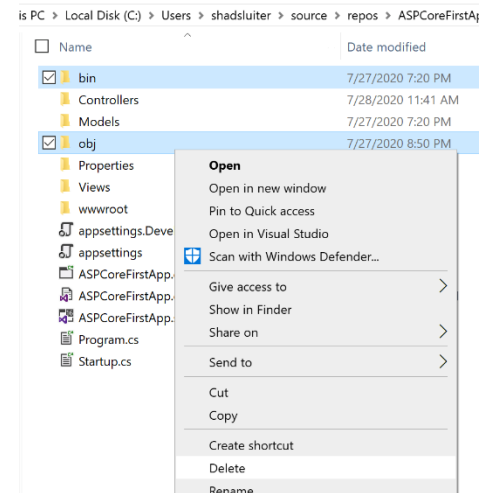
3. In the same Word document, add screenshots of the application being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.

4. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson. Be sure to explain the key words introduced in this lesson.

5. Include updated UML diagrams.

6. Convert the Word document to a PDF and attach the document separately from the zip file. Multiple files can be uploaded with an assignment.

7. Submit a ZIP file of the project file. In order to save space, you can delete the bin and obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.



Milestone 3: Using Recursion

Overview

In this milestone, students will use recursion to develop an algorithm that reveals "blocks" of cells with no live neighbors.

1. Write FloodFill(int row, int col) method in Board for an algorithm that reveals "blocks" of cells with no live neighbors in a Minesweeper game. The images below demonstrate the behavior of this algorithm.

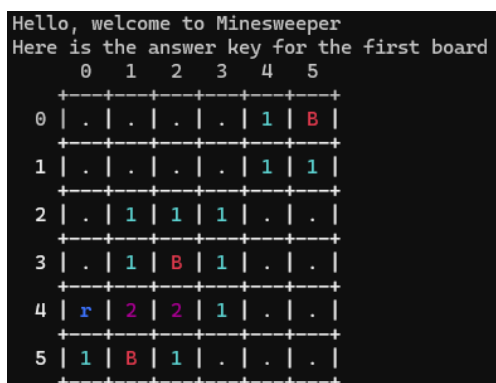


Figure 15 Start of the game. Answers revealed for testing purposes.


```

Here is the current board
  0  1  2  3  4  5
+---+---+---+---+---+
0 | ? | ? | ? | ? | ? |
+---+---+---+---+---+
1 | ? | ? | ? | ? | ? |
+---+---+---+---+---+
2 | ? | ? | ? | ? | ? |
+---+---+---+---+---+
3 | ? | ? | ? | ? | ? |
+---+---+---+---+---+
4 | ? | ? | ? | ? | ? |
+---+---+---+---+---+
5 | ? | ? | ? | ? | ? |
+---+---+---+---+---+
Enter the row number
0
Enter the column number
0
Enter 1 to visit the cell, 2 to flag the cell, 3 use a reward (bomb detector)
1

```

Figure 16 First move is 0,0

```

Game in progress
Here is the current board
  0  1  2  3  4  5
+---+---+---+---+---+
0 | . | . | . | . | 1 | ? |
+---+---+---+---+---+
1 | . | . | . | . | 1 | 1 |
+---+---+---+---+---+
2 | . | 1 | 1 | 1 | . | . |
+---+---+---+---+---+
3 | . | 1 | ? | 1 | . | . |
+---+---+---+---+---+
4 | 1 | 2 | 2 | 1 | . | . |
+---+---+---+---+---+
5 | ? | ? | 1 | . | . | . |
+---+---+---+---+---+
Enter the row number

```

Figure 17 Flood fill reveals most of the board. A few ? squares remain that were not touched by flood fill.

2. This flood fill method should be recursive. Inside the flood fill method, mark cells as "visited" = true when they are included in the block of affected cells. Recursively call floodfill on surrounding cells. You may wish to review the FloodFill Animations Presentation, located in the Topic 3 activity.
3. Update the UML class diagram to include the new method.

GitHub

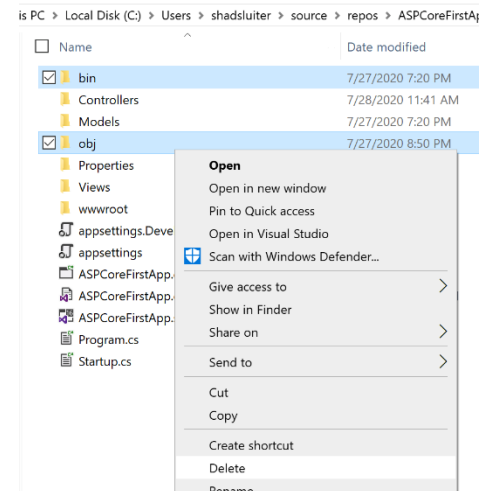
1. Review the "GitHub Tutorial," located in Class Resources, to learn how to create a git repository and use a version control system.
2. Create a commit of this Milestone and push it to the GitHub repository.
3. Your instructor will see the updates if you provided access.
4. Include the URL for the repository in the project Microsoft Word document.

Deliverables

Deliverables:

1. Create a screencast video of the application.

- a. Demonstrate the application working per the requirements.
 - b. Show relevant sections of the code, *explaining* the key ideas of this milestone and how they were implemented.
 - c. Past a link to the video in a Microsoft Word document.
2. In the same Word document, add the link to the GitHub repository. Share the repository with your instructor's email address.
3. In the same Word document, add screenshots of the application being run. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
4. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson. Be sure to explain the key words introduced in this lesson.
5. In the same Word document, include the updated UML diagram, flowchart for the flood fill process, and screenshots of the application being run successfully. Caption each image to explain what is being shown. Be sure to demonstrate the flood fill method working successfully, then convert it to a PDF document.
6. Submit a ZIP file of the project file. In order to save space, you can delete the bin and obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.



Milestone 4: GUI Version

Overview

In this milestone, students will create a GUI version of the Minesweeper game.

1. Add a new project to the solution. This project is a WinForms app.

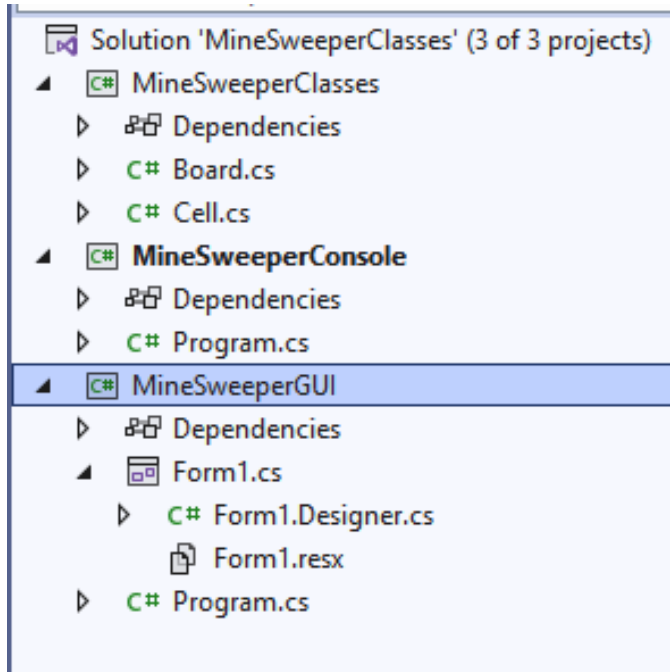


Figure 18 Three projects form the solution. MinesweeperGUI is the startup project for the solution.

2. Add a second form to get the game size and difficulty values from the user. Here are two suggested layouts for getting size and difficulty values from the user. Your form may need to be different depending on the properties of the Board class.

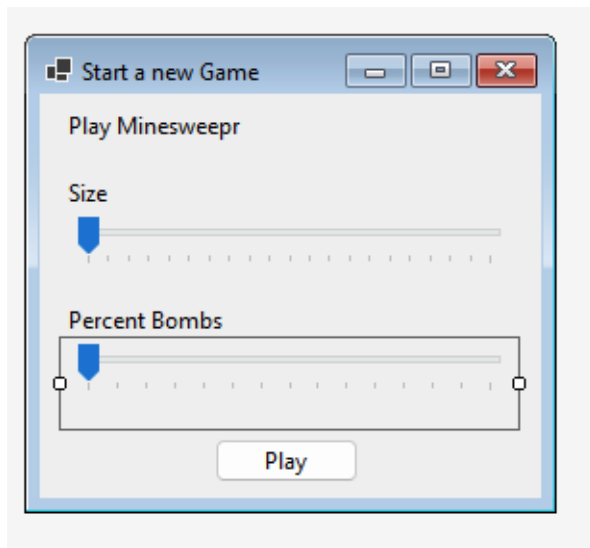


Figure 19 Two trackbars to get size and difficulty level.

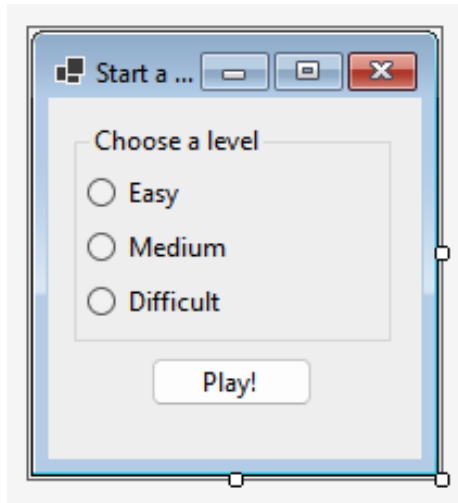


Figure 20 Three radio buttons used to get starting difficulty from user.

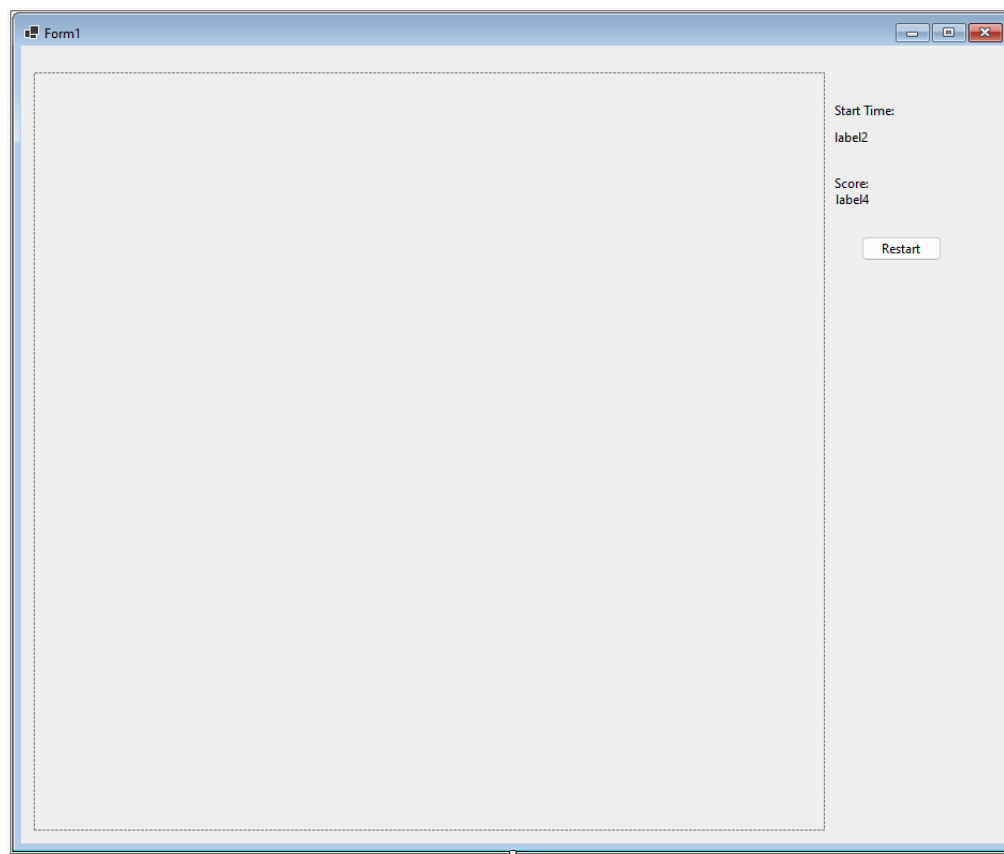


Figure 21 Suggested layout for the game form. A panel is used to contain the grid of buttons. Labels are used to display messages. A button is used to close the form and return to the start form.

3. Pass the data from Form2 to Form1 to begin the game.

```

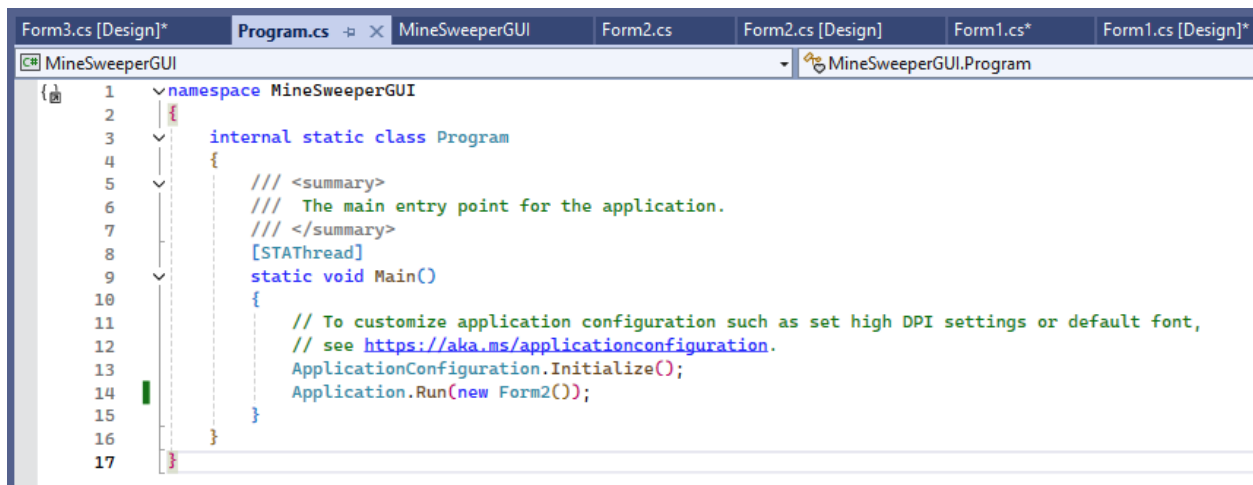
namespace MinesweeperGUI
{
    public partial class Form1 : Form
    {
        // instance of the Board class used to manage the game data
        Board board;
        // 2D array of buttons used to represent the game board in the GUI
        Button[,] buttons;

        // getting the size and difficulty from Form2 and setting up the game board
        public Form1(int size, float difficulty)
        {
            board = new Board(size, difficulty);
            InitializeComponent();
        }
    }
}

```

Figure 22 Initial code for Form1. Notice the two values from Form2 are passed in the constructor.

4. Using the Chessboard app from Activity 2, create a method to update the button faces based on the contents of the board.Cells properties. Use this suggested scheme or make one up.
 - a. Print an "F" for a flagged cell.
 - b. Print a "B" for a bomb.
 - c. Print a blank or a number to indicate the number of neighboring bombs.
 - d. Set the button background color to gray if it has not been visited yet.
5. Update the Program.cs file in the WinForms project. The startup form is shown on line 14. By default the startup form is Form1.



```

1  namespace MinesweeperGUI
2
3  internal static class Program
4  {
5      /// <summary>
6      /// The main entry point for the application.
7      /// </summary>
8      [STAThread]
9      static void Main()
10     {
11         // To customize application configuration such as set high DPI settings or default font,
12         // see https://aka.ms/applicationconfiguration.
13         ApplicationConfiguration.Initialize();
14         Application.Run(new Form2());
15     }
16 }
17

```

Figure 23 Form2 is set as the project's startup form.

6. Test the game.

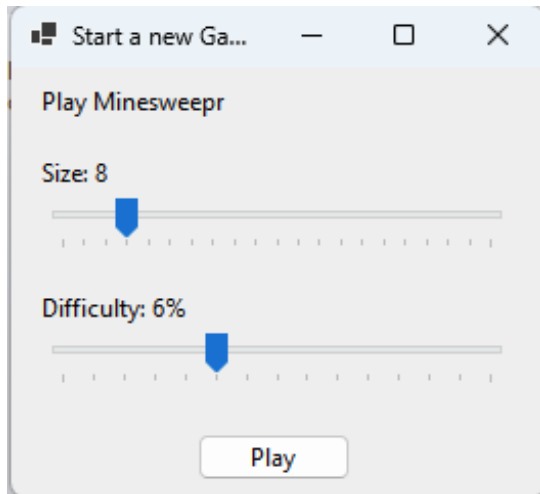


Figure 24 Form2 gets the size and difficulty values from the player.

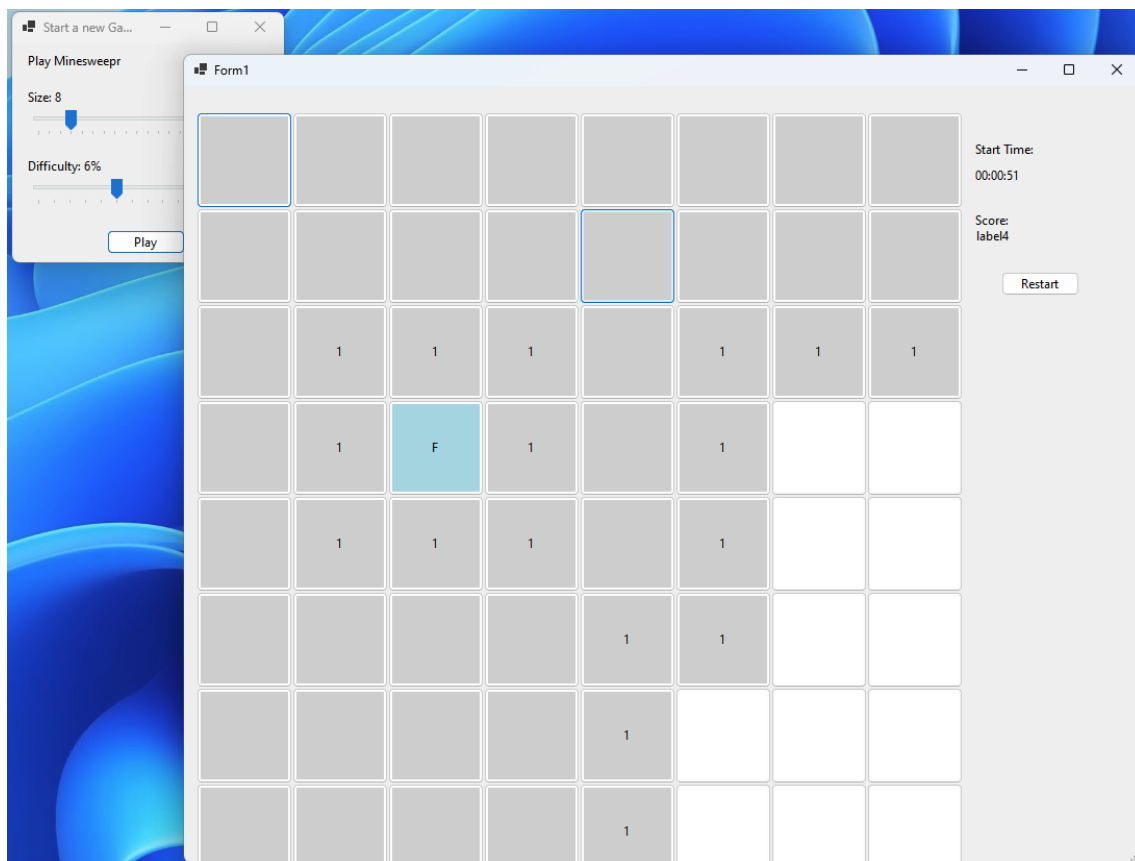


Figure 25 Game is in progress. Change button background colors and text properties to show the state of the cells.

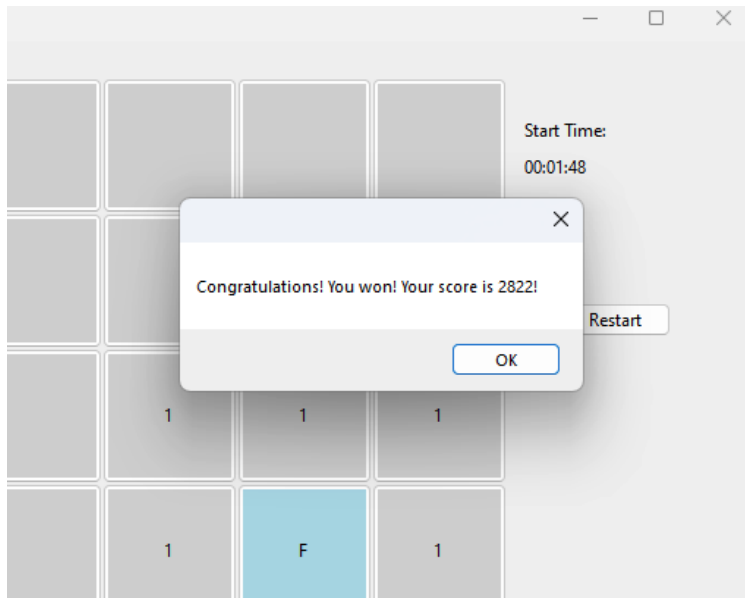


Figure 26 Game is over. Score is calculated based on size, difficulty, and time.

7. Add some game assets for Minesweeper graphics.

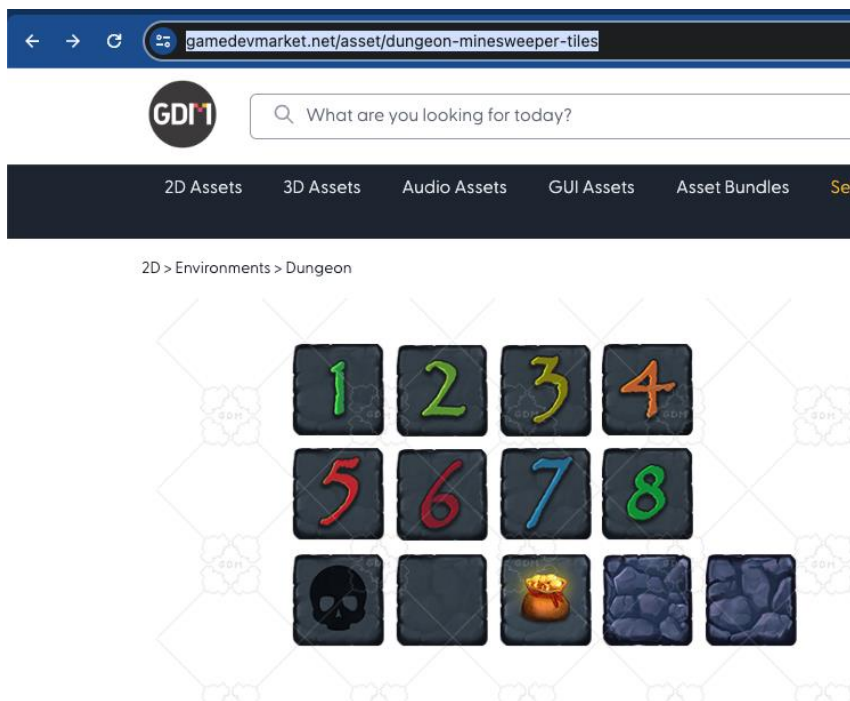


Figure 27 Free assets for a minesweeper game at GameDevMarket.com

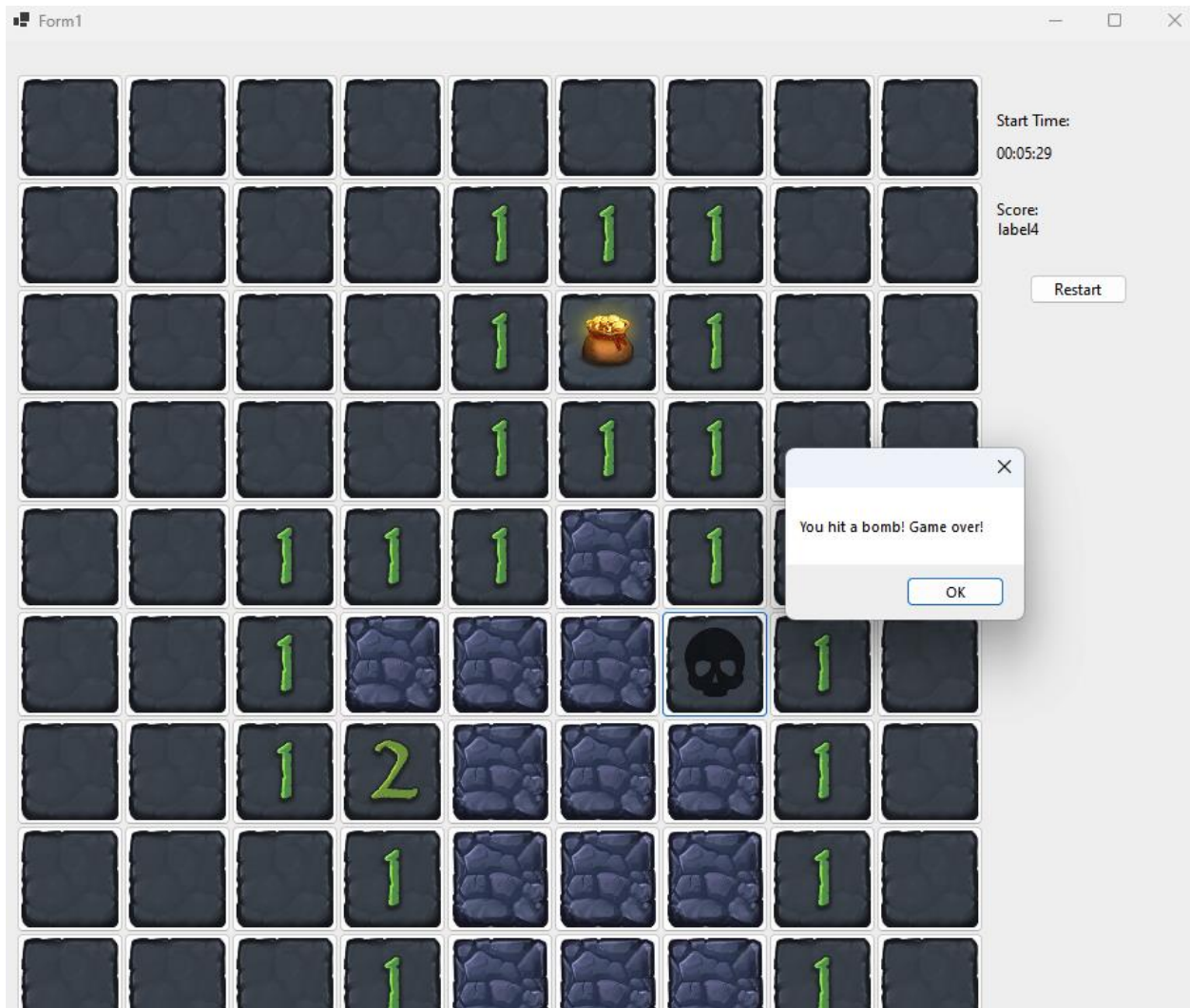


Figure 28 Game assets make the game look much better.

GitHub

1. Review the "GitHub Tutorial," located in Class Resources, to learn how to create a git repository and use a version control system.
2. Create a commit of this Milestone and push it to the GitHub repository.
3. Your instructor will see the updates if you provided access.
4. Include the URL for the repository in the project Microsoft Word document.

Deliverables:

Submit the following:

1. ZIP file containing the project folder containing all the source code.
2. Create a Microsoft Word document containing the wireframe design and screenshots of the program being run. Caption each image with a description of what is being demonstrated, then convert it to a PDF document.

Milestone 5: GameStat Class

Overview

In this milestone, you will manage game stats for winning games.

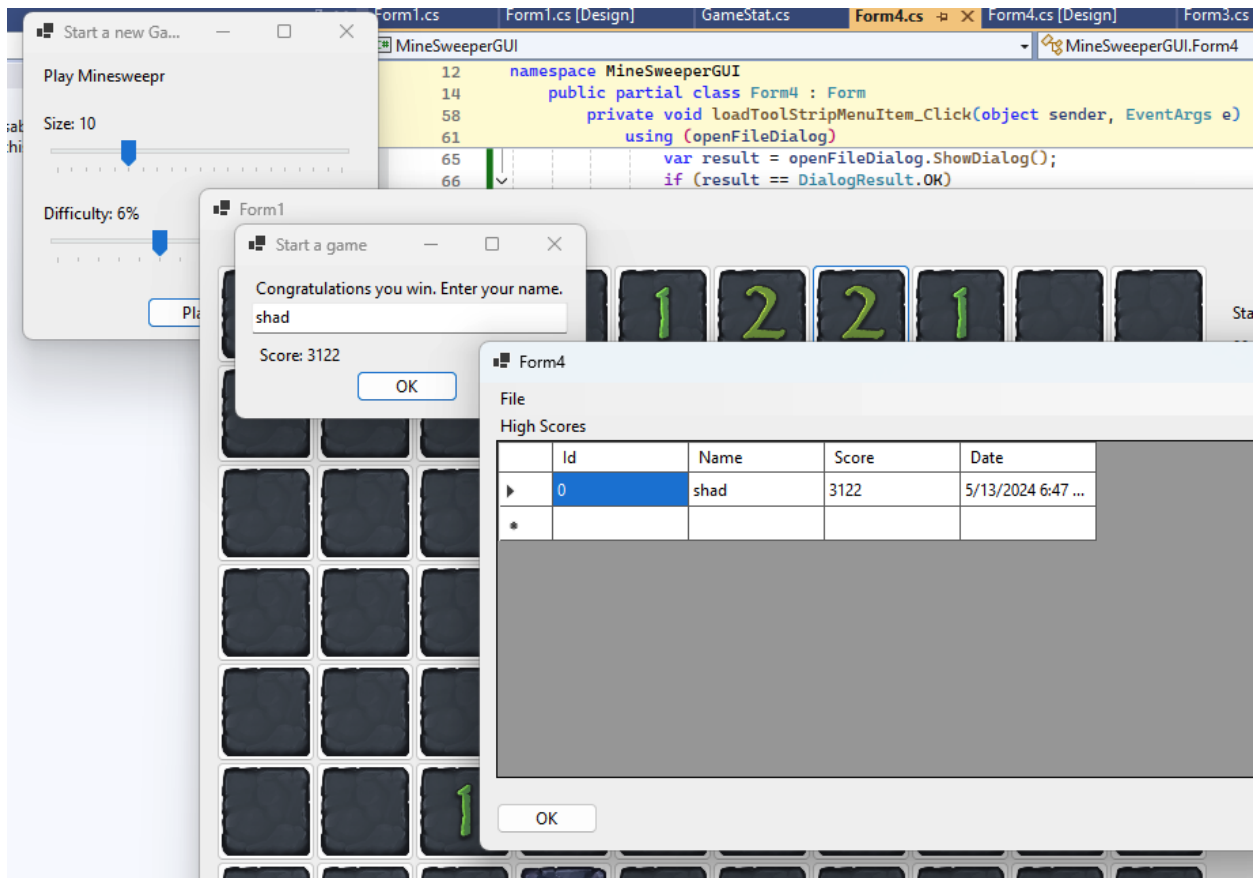


Figure 29 Form3 is used to capture the user's name. Form4 is used to show a list of scores in a gridview.

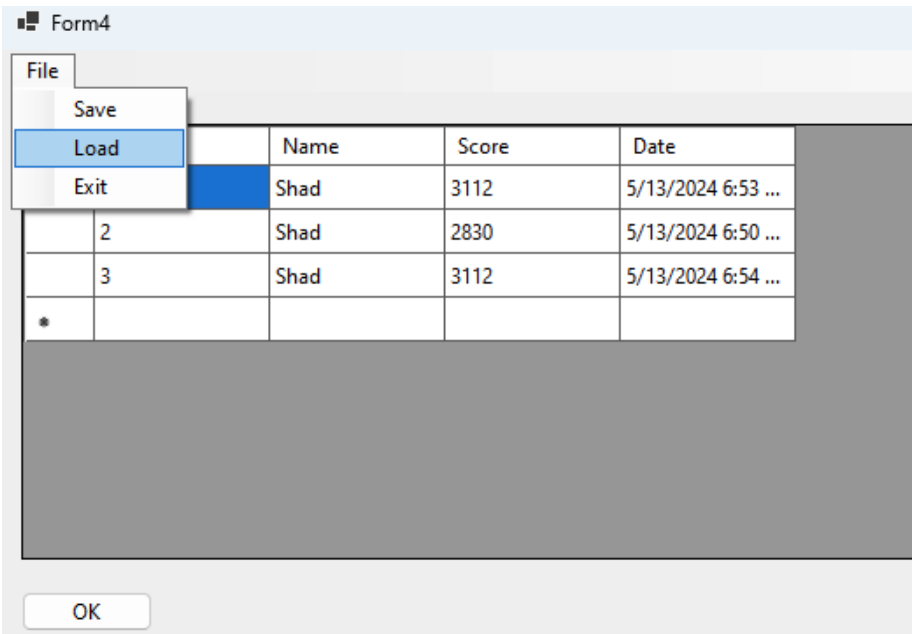


Figure 30 Form4 has a Save and Load option to manage a text file that contains scores.

Execution

Execute this assignment according to the following guidelines:

1. Create a GameState.cs class with properties id, name, score, and game time.
2. Create Form3 in order to capture the winner's name. Show Form3 after a winning game.

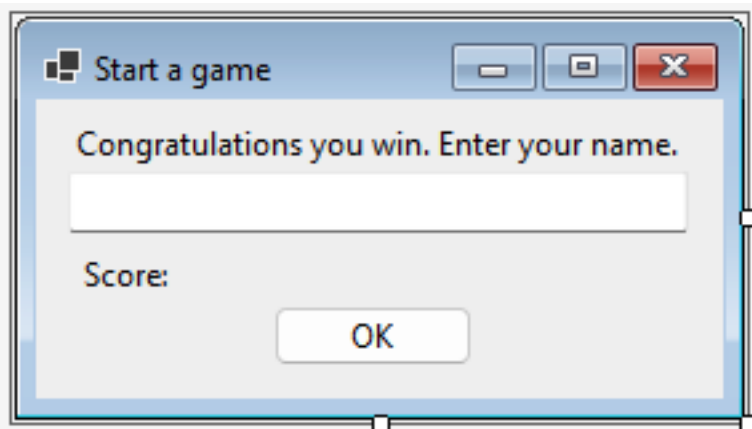


Figure 31 Get the user's name after winning.

3. Pass the **name** and **score** to Form4, which is used to display a list of winner's scores.

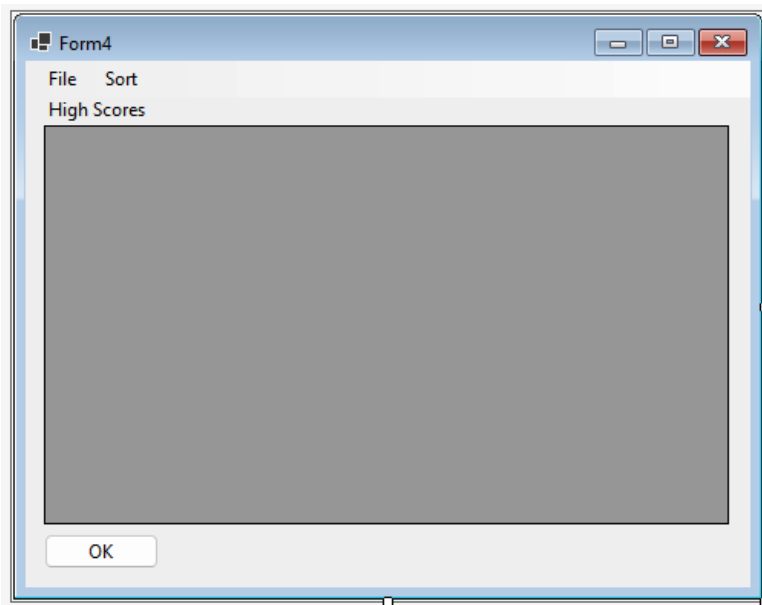


Figure 32 Form4 has a datagrid view and two menu options.

```
public partial class Form4 : Form
{
    GameStat gameStat;

    BindingSource bindingSource = new BindingSource();

    public List<GameStat> statList = new List<GameStat>();
    public Form4(string name, int score)
    {
        gameStat = new GameStat();
        gameStat.Name = name;
        gameStat.Score = score;
        gameStat.Date = DateTime.Now;
        gameStat.Id = statList.Count + 1;
        statList.Add(gameStat);

        InitializeComponent();
    }
}
```

Figure 33 Form4 gets the name and score values from Form3. It creates a new instance of the GameStat and adds it to a list.

4. Create a Menu for Form4 that has a File > Save File > Load and File > Exit option.

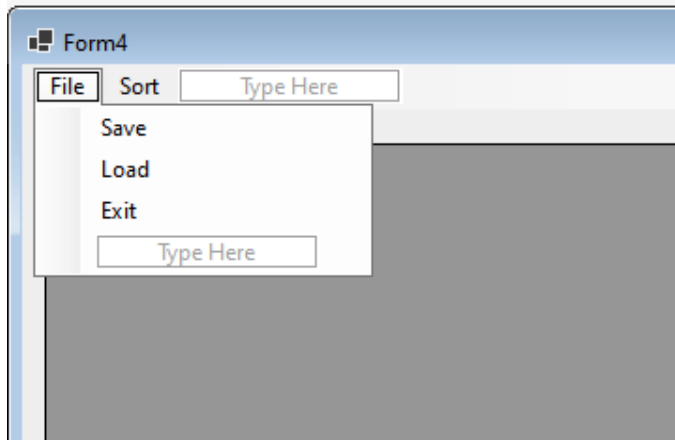


Figure 34 File menu has Save, Load and Exit commands.

5. Add a menu for Sort > By Name. Sort > By Score and Sort > By Date option.

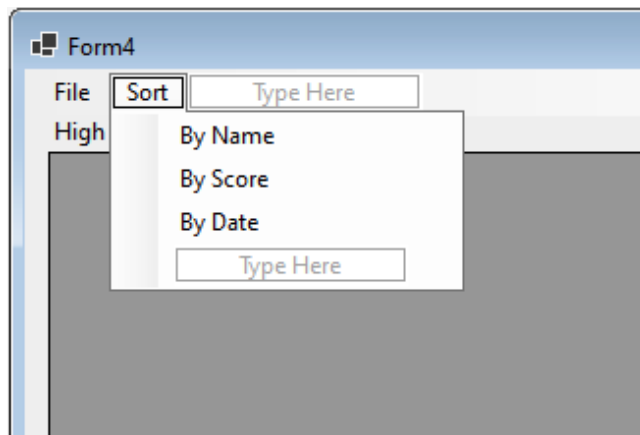


Figure 35 Sort menu has By Name, By Score, and By Date options.

6. Display the list of scores in the data grid control.

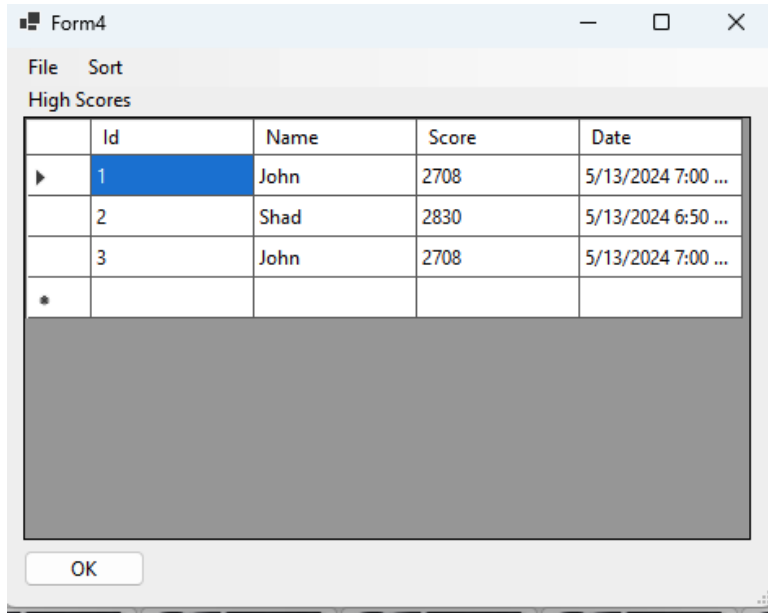


Figure 36 Three game stats have been played and saved to a text file.

7. Perform sorting operations according to the Sort menu choices.

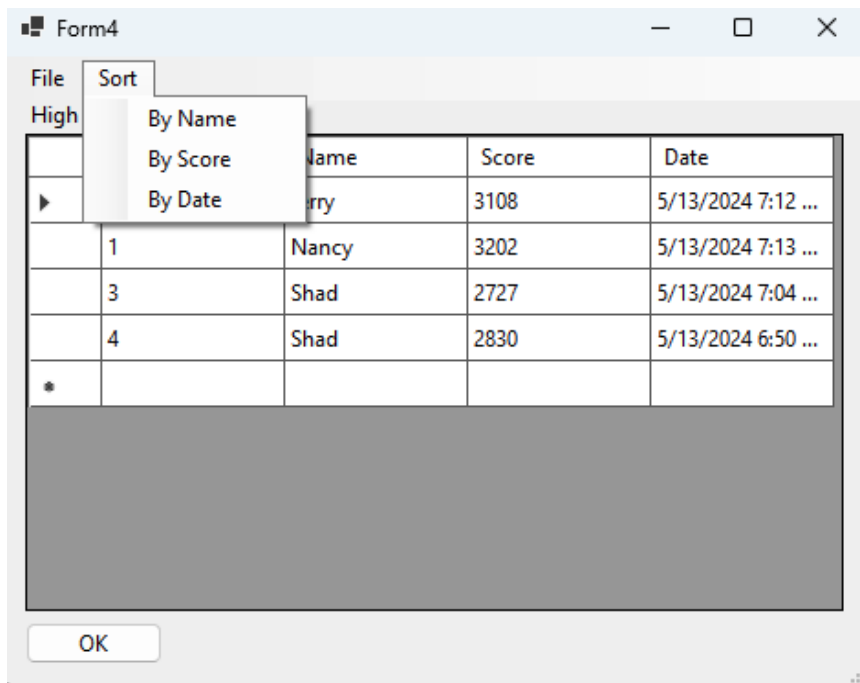


Figure 37 Sorting the scores by name, date, or score.

GitHub

1. Review the "GitHub Tutorial," located in Class Resources, to learn how to create a git repository and use a version control system.
2. Create a commit of this Milestone and push it to the GitHub repository.

3. Your instructor will see the updates if you provided access.
4. Include the URL for the repository in the project Microsoft Word document.

Deliverables:

Submit the following:

1. ZIP file containing the project folder containing all the source code.
2. A video presentation screencast to demonstrate all of the features of the game, plus a code walk-through to show the newest features of the app.

Milestone 6: Game Extensions

Objectives:

1. Apply skills from previous lessons to an open-ended problem.
2. Design and test code to implement program requirements.

Choose two of the features listed below. Implement the suggestion. Document that the features work without errors.

1. **Special Reward Cells:** Review the special reward that has been part of the Cell class since Milestone1. Implement the special power or reward that you defined.
2. **Custom Themes and Skins:** Allow players to customize the game's appearance by selecting different themes or skins for the UI, including the background, buttons, and icons. This could include seasonal themes or custom color schemes.
3. **Enhanced Stat Screen:** Enhance the PlayerStats form to include more analysis. Beneath the gridview, show average time per game, average number of points.
4. **Graphing:** Use a graphic library [like the one found here](#) to graphically display the range of scores on Form4. Show a line chart or bar chart to compare the scores earned in each game.
5. **Sound Effects and Music:** Integrate sound effects for game actions (flagging, revealing a cell, or hitting a bomb) and background music that can be toggled on or off. Provide a few music tracks to choose from, enhancing the gaming experience.
6. **Save and Resume Game:** Allow players to save their game progress and resume it later. Use the JSON serializing feature to save the Board class to a string and store in a text file. This feature would be especially useful for larger and more complex boards or when playing on higher difficulty levels.

GitHub

1. Review the "GitHub Tutorial," located in Class Resources, to learn how to create a git repository and use a version control system.
2. Create a commit of this Milestone and push it to the GitHub repository.
3. Your instructor will see the updates if you provided access.
4. Include the URL for the repository in the project Microsoft Word document.

Deliverables:

Submit the following:

1. ZIP file containing the project folder containing all the source code.
2. A video presentation screencast to demonstrate all of the features of the game, plus a code walk-through to show the newest features of the app.