CST-250 Project Milestone Guide

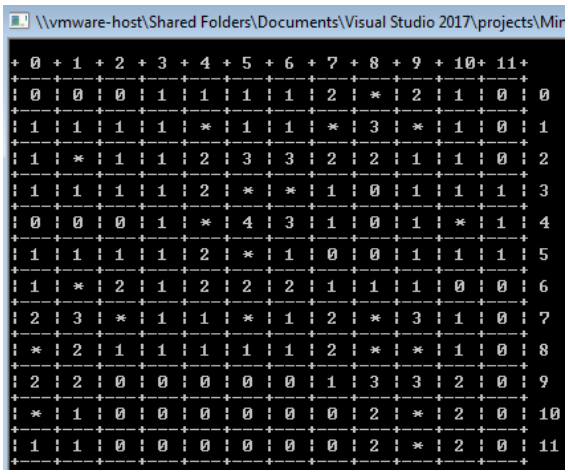
Directions: Throughout this course, students will incrementally design and build a Minesweeper Game.

Contents

Milestone 1: Console Application

Overview

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



Execution

Execute this assignment according to the following guidelines:

1. Draw a UML class diagram for the following requirements. 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. If you need a review of UML diagrams you can review video "UML Class Diagram Tutorial," located in the course materials.
2. Create a class that models a game cell. A game cell should have the following properties:
    a. Its row and column. These should initially be set to -1.
    b. Its visited boolean value. This should initially be set to false.
    c. Live boolean value. This should initially be set to false. "Live" set to true will indicate that the cell is a "live bomb" cell.
    d. The number of neighbors that are "live." This should initially be set to 0.

    The Cell class should have a constructor, as well as getters and setters for all properties.
3. Create a class that models a game board. A game board should have the following properties:
    a. Size. The board will be square, where the size includes the dimensions of both the length and width of the board.
    b. Grid. The grid will be a 2-dimensional array of the type cell.
    c. Difficulty. A percentage of cells that will be set to "live" status.
4. The Board class should have the following methods:
    a. 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.

b. setupLiveNeighbors. A method to randomly initialize the grid with live bombs. The method should utilize the Difficulty property to determine what percentage of the cells in the grid will be set to "live" status.

c. calculateLiveNeighbors. 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.

5. Program

   a. The Program class should be the console app that drives the application. This is the class that should contain a main() method. The main program should have a printBoard helper method that uses, for loops, the Console.write and Console.writeLine commands to display the contents of the Board as shown at the beginning of these instructions.

6. The main() method should:

   a. Create an instance of the Board class.

   b. Call the Board.setupLiveNeighbors and Board.calculateLiveNeighbors commands to initialize the grid.

   c. Call the printBoard method to display the contents of the grid.

Deliverables

Submit the following:

1. ZIP file containing the project folder from Visual Studio.
2. Microsoft Word document containing the UML diagram and screenshots of the application being run. Include captions beneath any images you capture.

Milestone 2: Interactive Playable Version

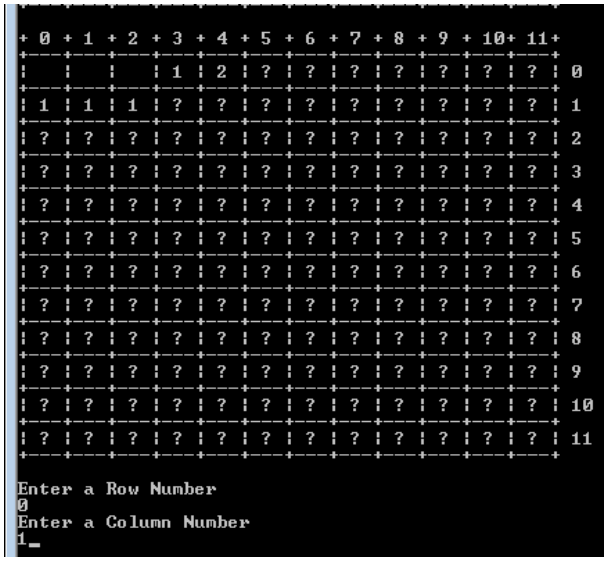
Overview

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

Execution

Execute this assignment according to the following guidelines:

1. Update the UML class diagram based on the new features described below.
2. Create a new helper function inside the Program class that will display the contents of each that have been visited. You might want to name the method PrintBoardDuringGame or something similar. This will be similar to the PrintBoard function developed in Milestone 1, but this time display either the number of live neighbors or an empty square if there are no live neighbors. If a cell has not been visited, print a question mark.



3. Game Loop

Create a playable version of the game in a while loop within the main method of the program class. Use this pseudo code as a guide.

```
while (the game is not over yet) {
        1. Ask the user for a row and column number.
        2. If the grid contains a bomb at the chosen cell (row, column) then set
           the endgame condition to true. Display a failure message.
        3. If all of the non-bomb cells have been revealed, then set the endgame
           condition to true. Display a success message.
        4. Print the grid.
}
```

4. Game Loop Flow Chart

Create a drawing of the flowchart for the game flow. Use standard flowchart symbols. If you need a review of flowchart diagrams, review the video "Introduction to Creating Flowcharts," located in the course materials.

Deliverables

Submit the following:

1. ZIP file containing the project folder with all the source code.
2. Microsoft Word document containing updated UML diagram, flow chart, and screenshots of the application being run. Caption each image to explain what is being shown.

Milestone 3: Using Recursion

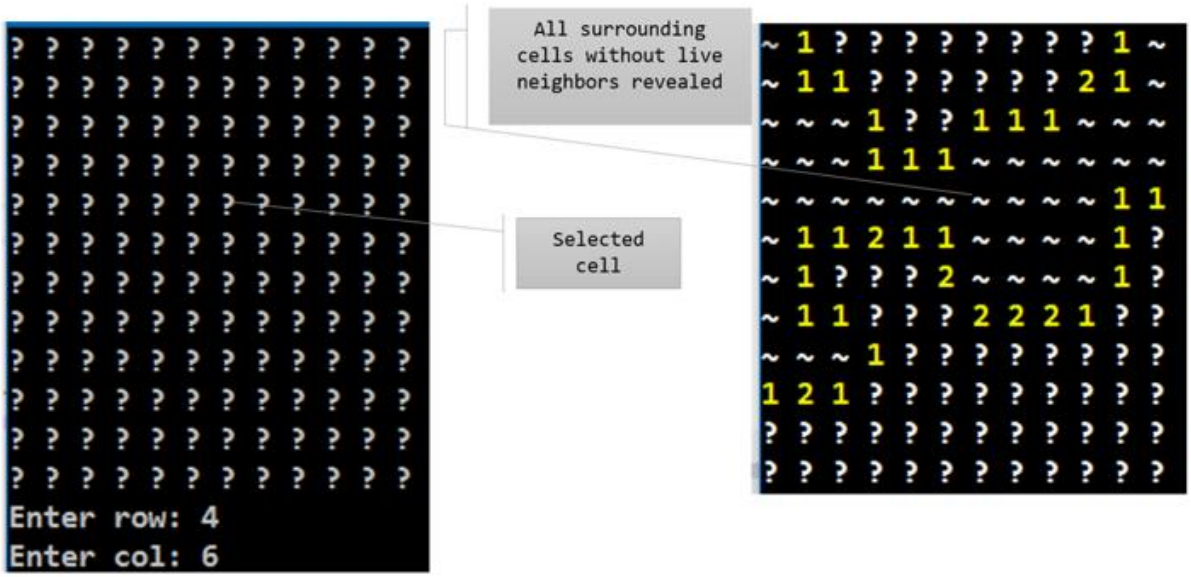
Overview

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

Execution

Execute this assignment according to the following guidelines:

1. Write a recursive definition 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.



   The pictures above show the ~ symbol represents visited cells.

1. Draw a flowchart showing how floodfill will operate.
2. Add a method to the Board class called floodFill(int row, int col) This method should be recursive. Inside the floodFill 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 required readings.
3. Update the UML class diagram to include the new method.

Deliverables

Submit the following:

1. ZIP file of the project folder containing all the source code.
2. Microsoft Word document containing the updated UML diagram, flowchart for the floodfill process, and screenshots of the application being run successfully. Caption each image to explain what is being shown. Be sure to demonstrate the floodFill method working successfully.

Milestone 4: GUI Version

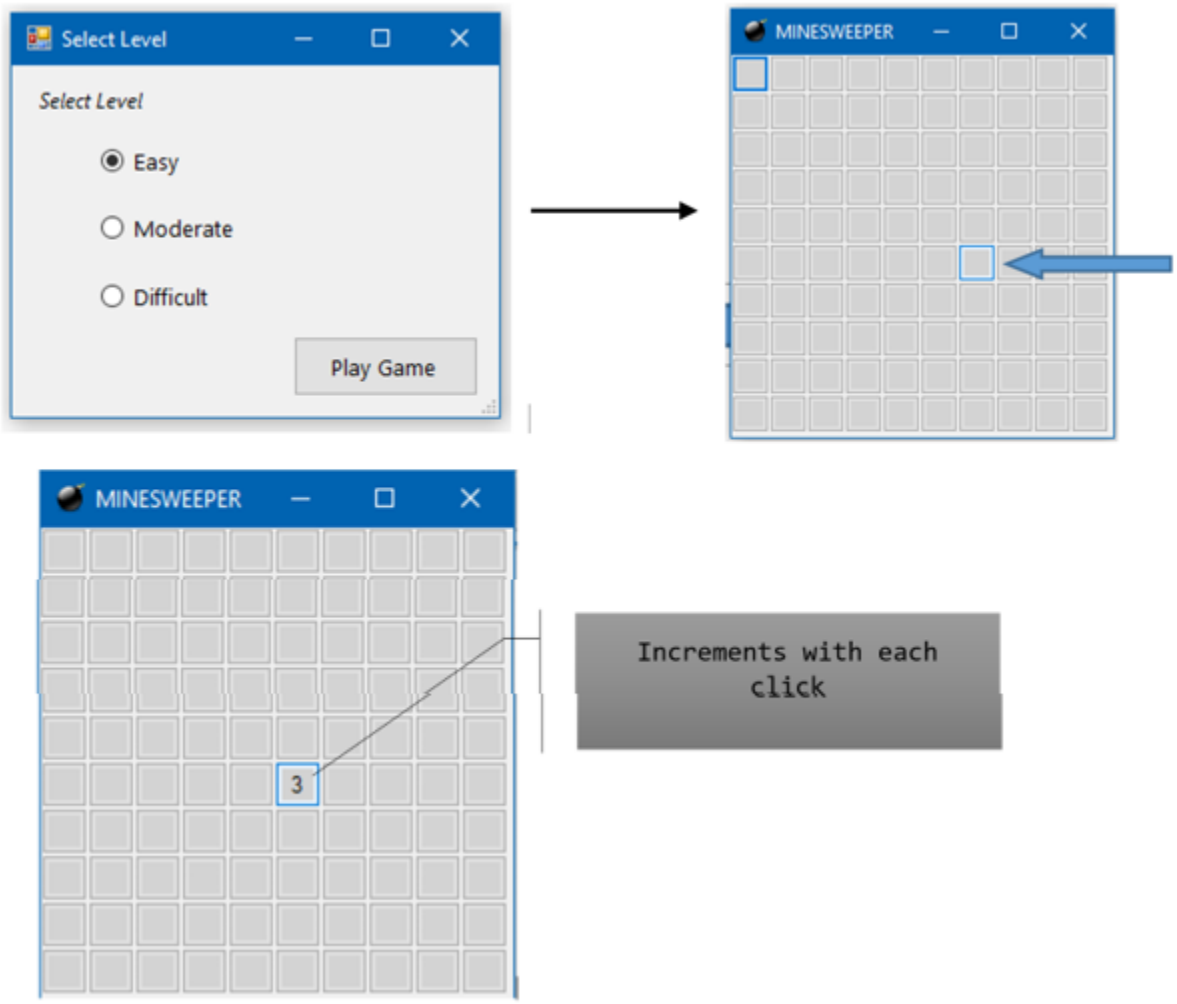
Overview

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

Execution

Execute this assignment according to the following guidelines:

1. Using the chessboard activity as a model, create a Windows Form that displays a grid of clickable cells.
2. For this phase of the project, you do not have to incorporate the Board or Cell classes. Focus your attention on building an organized GUI. In the following milestone, we will integrate the Cell and Board classes into the solution. Create two forms.
3. Create a form that prompts the user for a level of difficulty. Open a second form after the user selects an option.
4. On the second form, dynamically create a grid of buttons when the form loads. Associate a single method to handle click events on all of the buttons in the grid. The method should do something visual when the button is clicked such as change the color of the button or increment a counter.
5. Create a wireframe design of the application forms. Use any drawing tool of your choice.



Deliverables

Submit the following:

1. ZIP file containing the project folder containing all the source code.
2. 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.

Milestone 5: Combining GUI and Game Logic

Overview

In this milestone, students will add the game play logic to their GUI application.

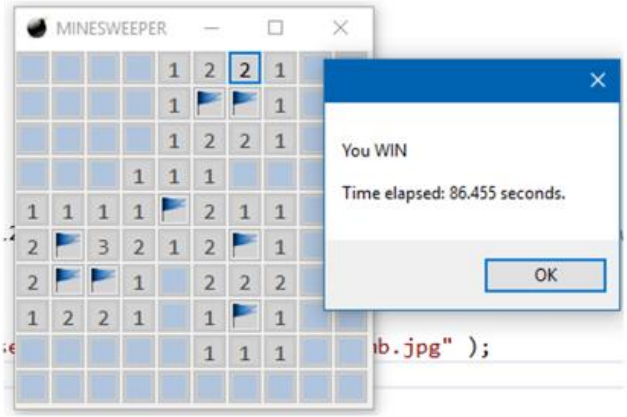
Execution

Execute this assignment according to the following guidelines:

1. Create a new Winforms solution in Visual Studio.
2. Import the console app as a second project to the new solution. The second project should be a copy of the existing console app project, which includes the Board and Cell classes. The main program console app file is no longer needed.
3. The new form in the GUI application will accomplish the tasks of the console app did including getting input and displaying the board. However, the new interface uses buttons instead of printed text to communicate with the user. The text or image on each button will depend on the value stored in the Board class.
4. There are several "expected" features in the classic Minesweeper game. Be sure to implement these into the project:
    a. If a user clicks on a mine, the entire game board is revealed with "bomb" pictures on the mines and a "Game Over" message is displayed.
    b. If a user successfully exposes all squares without clicking on a mine, the entire game board is revealed with "flag" pictures on the mines and a "You Win" message is displayed.
    c. If a user right-clicks on a square, a "flag" picture is placed on the square. Right-click has not been demonstrated in previous tutorials, so you will have to research how to do the right-click event.
5. Add a stop watch to the application and record the length of time it takes a player to win the game. Display the elapsed time with the "You Win" message.
6. Refer to the examples below. Your specific design should be similar.

Example 1:

Example 2:



Deliverables

Submit the following:

1. ZIP file containing the project folder containing all the source code.
2. Create a video screencast demonstrating all of the features of the application. Submit a text document containing a URL to the video.

Milestone 6: PlayerStats Class

Overview

In this milestone, students will create a Playerstats class that models a player's game data, as well as adds a "high score" page to the application.

Execution

Execute this assignment according to the following guidelines:

1. In Minesweeper, a player's score is the time taken to successfully complete the game. The player that takes the least amount of time has the highest score. However, you should implement a scoring formula to reward higher levels of difficulty. A player who gets the minimum time on the hardest level should receive the most points. A player who gets a lower time on an easy level should not be at the top of the high score list.
2. Begin this milestone by implementing PlayerStats class.
3. This class should implement the IComparable interface and should contain attributes for the player's initials (or names), the level of difficulty played, and the time elapsed. You may add additional data if needed.
4. Next, add a high score form to the project.
5. After every game, the high score page should display the top 5 scores for the corresponding game level. Player game time should be read from and written to an external file for persistent storage.

6. When the application begins, populate the collection PlayerStats objects. The PlayerStats objects are created with data read from the external file. If there is no external file yet, then create an empty list of scores.
7. Use LINQ statements to query the collection to retrieve only the appropriate data for the high score form.

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 7: Final Project Presentation

Overview

In this milestone, students will make any revisions and corrections to get the Minesweeper project into its best state and then prepare a video demonstration of the project.

Execution

Execute this assignment according to the following guidelines:

1. Prepare a video demonstration of the project.
2. The video should be between 5 and 10 minutes long, and should include the following:
    a. A demonstration of the Minesweeper game
    b. A brief discussion of the challenges encounters and how they were resolved
    c. An explanation of the recursive algorithms used to open up spaces with no live neighbors
    d. What you learned

Deliverables

Submit the following:

1. ZIP file containing the project folder containing all the source code.
2. A link of your video presentation to LoudCloud.