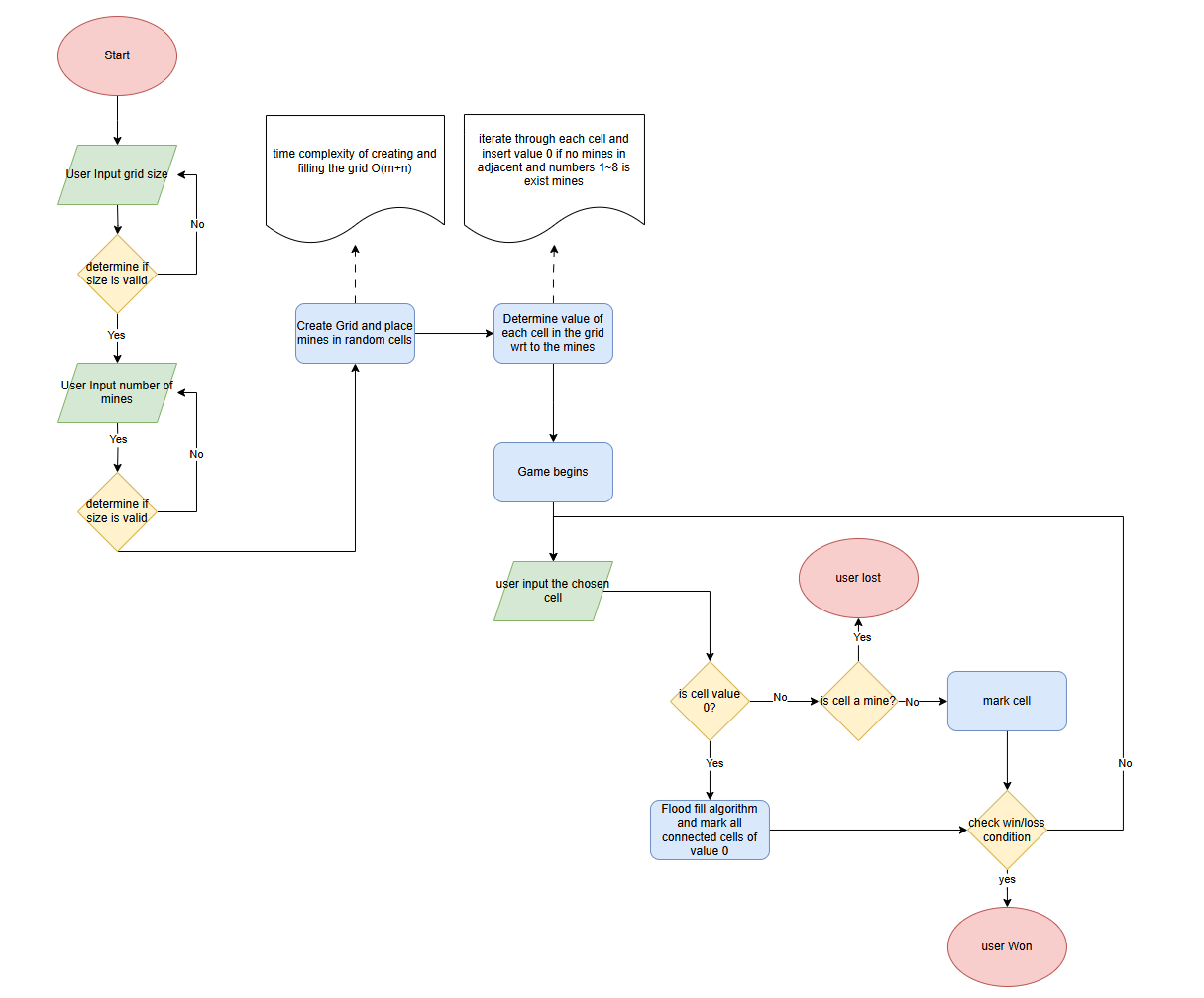
Environment: Windows

Software: Visual Studio 2022

Instructions

1. Open Minesweeper.sln
2. Compile and run

Code algorithm explanation:  
  


Assumptions:

1. Maximum grid size: 20x20
2. Orthogonal Check of connected 0

**Documentation: Minesweeper Game Algorithm (Simplified Reveal Logic)**

**1. Introduction**

This document describes the algorithm for a simplified version of the Minesweeper game, based on the provided flow diagram. The algorithm covers the game setup, initialization, the main gameplay loop driven by user input (specifically, clicking to reveal a cell), the process of checking the clicked cell, handling the cascading reveal of empty squares, and determining the win or loss state of the game. This version of the algorithm explicitly *excludes* the logic for flagging cells or using chording mechanics.

**2. Algorithm Phases**

The algorithm can be broken down into the following main phases:

* **Setup and Initialization:** Getting game parameters, creating the game board, placing mines, and calculating the numbers for safe cells.
* **Game Loop:** The continuous cycle of receiving user input, processing it, and updating the game state until a win or loss condition is met.
* **Cell Checking and Reveal:** The process of determining the consequence of clicking a specific cell and executing the appropriate reveal action.
* **Win/Loss Condition Checking:** Evaluating the current state of the board after each move to see if the game has ended.

**3. Detailed Algorithm Steps**

The algorithm proceeds as follows:

**Phase 1: Setup and Initialization**

1. **Start:** The algorithm begins.
2. **User Input Grid Size:** The system prompts the user to enter the desired dimensions for the game grid (e.g., number of rows and columns).
3. **Determine if Size is Valid:** The system checks if the entered grid size is within acceptable parameters (e.g., positive numbers, within program limits).
   * *If Invalid:* Return to "User Input Grid Size" to prompt the user again.
   * *If Valid:* Proceed to the next step.
4. **User Input Number of Mines:** The system prompts the user to enter the desired number of mines to be placed on the grid.
5. **Determine if Size is Valid (Mines):** The system checks if the entered number of mines is valid for the chosen grid size (e.g., non-negative, less than the total number of cells in the grid).
   * *If Invalid:* Return to "User Input Number of Mines" to prompt the user again.
   * *If Valid:* Proceed to the next step.
6. **Create Grid and Place Mines in Random Cells:**
   * A grid data structure is created with the specified dimensions. Each cell is initially in a "covered" state.
   * The specified number of mines are placed in randomly selected cells within the grid. Care must be taken to avoid placing multiple mines in the same cell if using simple random selection without tracking used locations.
   * *(Implicit Action):* A common implementation detail not explicitly shown is ensuring the *first* clicked cell is not a mine. This often involves placing mines *after* the first click or rearranging them if the first click location happens to have a mine. However, based strictly on this diagram, mines are placed upfront.
7. **Determine Value of Each Cell in the Grid WRT to the Mines:**
   * The algorithm iterates through *each* cell in the grid.
   * If a cell contains a mine, its value is irrelevant for this step (or can be considered a special "mine" value).
   * If a cell does *not* contain a mine, the algorithm counts how many of its adjacent cells (including diagonally adjacent, usually 8 neighbors) contain a mine.
   * This count (0-8) is stored as the value for that non-mine cell. A value of 0 indicates a "blank" cell with no adjacent mines.
8. **Game Begins:** The initialization is complete, and the game is ready for player interaction.

**Phase 2: Game Loop**

1. **User Input the Chosen Cell:** The system waits for the player to select a cell on the grid, typically by left-clicking it.
2. **Check Win/Loss Condition (Pre-Input):** (While the diagram places this check *after* processing the input, in some designs a check might occur here, though less common. Following the diagram, the check is primarily done *after* a cell is revealed).
3. **Check the Grid Value:** The system accesses the state and value of the chosen cell.
4. **Is it Cell a Mine??** The system checks if the chosen cell contains a mine.
   * *If Yes (It's a Mine):* Proceed to the loss state.
   * *If No (It's Not a Mine):* Proceed to check the cell's numerical value.
5. **Is Cell Value 0?** The system checks the calculated value of the chosen non-mine cell.
   * *If Yes (Value is 0):* Proceed to execute the Flood Fill algorithm.
   * *If No (Value is > 0):* Proceed to uncover the single cell.

**Phase 3: Cell Checking and Reveal Actions**

1. **(Reached from Step 12 - Is it Cell a Mine? -> Yes):**
   * **user Lost:** The game ends, and the player loses. This is a terminal state. Typically, all mines on the board are revealed at this point.
2. **(Reached from Step 13 - Is Cell Value 0? -> Yes):**
   * **Flood Fill Algorithm and Mark all Connected Cells of Value 0:**
     + This algorithm starts from the clicked cell (which has a value of 0).
     + The clicked cell is marked as "uncovered" and its value (0) is displayed (or it appears blank).
     + The algorithm then recursively or iteratively checks all of the *adjacent* cells (8 neighbors) of the current cell.
     + For each adjacent cell that is *still covered* and *not a mine*:
       - Mark the adjacent cell as "uncovered".
       - Display the value of the adjacent cell.
       - If the adjacent cell's value is also 0, the Flood Fill algorithm is called recursively or added to a queue/stack to process its neighbors as well.
       - If the adjacent cell's value is > 0, it is simply uncovered and its value displayed; the flood fill does not continue from this cell.
     + This process continues until all connected cells with a value of 0 and their immediate numbered neighbors have been uncovered.
   * After the Flood Fill completes, proceed to "Check win/loss condition".
3. **(Reached from Step 13 - Is Cell Value 0? -> No, and Step 12 - Is Cell a Mine? -> No):**
   * *(Implicit Action):* Uncover the single cell.
   * **Mark cell:** The chosen cell is marked as "uncovered", and its numerical value (> 0) is displayed.
   * Proceed to "Check win/loss condition".

**Phase 4: Win/Loss Condition Checking**

1. **(Reached after Flood Fill or Uncovering a Single Cell):**
   * **Check win/loss condition:** The algorithm evaluates if the game has ended.
   * It checks if the **win condition** is met: Have all non-mine cells on the grid been uncovered? (This means the number of cells in the "uncovered" state equals the total number of cells minus the total number of mines).
   * *(Note: A loss condition from hitting a mine is handled directly in step 12/14)*
   * *If Win Condition Met:* Proceed to the win state.
   * *If Win Condition Not Met (and not already lost):* The game is not over. Proceed back to the start of the game loop.
2. **(Reached from Step 17 - Check win/loss condition -> Win):**
   * **user Won:** The game ends, and the player wins. This is a terminal state.

**4. Conclusion**

This algorithm describes the fundamental process of setting up a Minesweeper grid, handling user clicks to reveal cells, executing the cascading reveal for empty areas, and determining if the player has won by uncovering all safe squares or lost by hitting a mine. It serves as a core logic framework for a basic Minesweeper game, focusing purely on the reveal and win/loss mechanics.