CMPE 230 SPRING 2024 PROGRAMMING PROJECT 3 - MINESWEEPER GAME REPORT

Berk Göktaş 2022400039
Bora Aydemir 2021400054

June 2024

1 Purpose

Our main purpose was to implement the Minesweeper game with a simple and plain concept to maintain the code's reusability and reduce the effort needed while implementing and optimizing the code. We had a well-thought-out, structured, and organized design process to achieve this.

2 Design

As stated in the purpose part, the design concept was crucial for the project's maintainability. Before starting to code, we thoroughly considered the implementation that would simplify our work. Most of the thought process was on whether to use different classes for different aspects of the program and the hint algorithm that handles the specific task we were given without overcomplicating the algorithm itself. At first, we wanted to a different class for the cells for a smooth flow. However, we realized that this would make the project more complicated and harder to read. Since only the icon of the cell would change after every action, we thought it wasn't vital to create a cell class and hold their properties as it was insignificant to the project's functionality. We also introduced some global vectors and variables to ease our way through the maintenance of the game grid. We introduced some functions to reduce repetitive operations such as changing the icons of the buttons and setting up the environment at the beginning and after a restart. We also thought about the hint algorithm and how to implement that efficiently. After having a clear design process, we split the tasks and began implementing the project merging the changes we made on our code in a shared repository and effectively communicating with each other in key aspects that might affect the integrity of the project.

3 Implementation

We first implemented an M X N button grid. It was surprisingly one of the most challenging parts of the project. We encountered some anomalies in that process. We first wanted to implement a resizeable grid that resizes itself automatically when the window gets bigger. However, while trying to implement it, the space margins between the cells started to behave unexpectedly. Also, the default resizeEvent class was tracking all the resize events so whenever we made a change in the main window size, the icons and the buttons were taking some time to adapt and sometimes they didn't behave as we wanted as there were limits to the button sizes. After facing a lot of challenges at the very beginning of our project we decided to reconsider whether we wanted to implement a resizeable grid at the later phases of the project, after getting the game to a base functionality. We used QT's classes effectively to create the grid, and buttons and used the QRandomGenerator to randomly place the mines inside the grid. We used two grids: mineGrid and buttonGrid. The mineGrid was responsible for holding the values of the cells indicating how many neighbors of that cell had mines. The buttonGrid kept the visual changes of the buttons and displayed them according to the actions the player was taking. We kept column, row, and number of mines as global variables to be able to change them easily on demand. We also created a header file for the main window to keep things tidy. We used signals and slots to determine whether a button was clicked and override the mouseClickEvent function to introduce right clicking action. After clicking a cell that is not a mine, we wanted to make sure that if it had no neighbor mines, then the other cells near it would open recursively. We implemented a recursive function revealAdjacentCells to accomplish that. This function starts on a cell that has no neighbor mines, checks all of its neighbors, and opens them in the same manner. We kept a global variable of flagged cells as a 2D boolean vector to implement the right-click flag toggle functionality. After that, we proceeded to implement the hint button. Our hint algorithm was pretty straightforward: First, it visits all revealed cells and determines their unrevealed neighbor counts. Then, if that unrevealed neighbor count is equal to the number of the neighbor mines of that specific cell, this means all of the neighbors of that cell are mines. If this is the case, we append all of those neighbors to a global vector bombCoordinates. If the unrevealed neighbor count is bigger than the number of the neighbor mines, the algorithm will reiterate all the neighbors of that cell and check whether they are in the bombCoordinates. If so, it will decrease the number of the neighbor mines of that cell as that cell is a mine. If it can make the neighbor mines count 0, that means all of the mines around a cell are already found and all the other neighbors of that cell are safe. In this case, the algorithm just returns the first safe cell it finds through the iteration. If it can't make the neighbor mines count 0, this means we don't have enough information about that cell yet. Then the algorithm proceeds to the next cell. After all the cells are visited, the program checks whether a change is made to the bombCoordinates vector. We check this with a boolean called change. If change is true, new mines are found throughout the process, and

all the cells should get checked again, but if change is false, this means there are no new mines to find so the information is not enough to find a hint and returns. This hint algorithm and the hint-giving process were the hardest parts of the project and we faced many difficulties implementing them. For example, as we wanted the second click to the hint button to reveal the cell if there were no changes made to that cell, we had to check if a hint was given and active before finding and revealing a cell. Within this process, the boolean values got very complicated and we faced a non-ending recursion in specific cases which caused the program to force quit. We fixed this problem by determining the part causing the infinite recursion and made the code more readable so as not to face the same kinds of issues again. This algorithm has a flaw: For a safe cell to be determined, we have to make sure some cells are mine. However from a human perspective, in some situations, we can find two cells where one of them is definitely a mine but we don't know which, in that case, if those cells are the neighbors of a cell that has a neighbor mine count of 1, we can determine the third neighbor is safe as the mine is in one of the other cells. However, this algorithm can't find this as it does not assume that, for this algorithm to find that safe cell, it has to determine the specific cell of the two cells with the mine and proceed to calculate the safe cell with that information. Therefore it can't assume and try possibilities. This algorithm can be improved by adding those flaws into consideration, however as it was not expected to be a perfect algorithm and it was perfectly within the guidelines, we wanted to make it simple. After completing the functionalities, we returned to the resize topic we discussed at the beginning of the project. In the end, we decided to implement a fixed-size grid to keep things simple.

4 Examples and How To Use The Program

After opening the project files on QtCreator, and clicking the run button at the bottom left, the program will compile and the Minesweeper game will pop up as a window. You can left-click the restart button to restart and left-click the hint button to receive a hint when it's possible. Left-clicking a cell will reveal it, and right-clicking a cell will flag it. You will lose if you click a mine and you will win if you reveal all the cells without a mine. If a hint is already given, clicking the hint button again will reveal the hinted cell. Here are some examples:

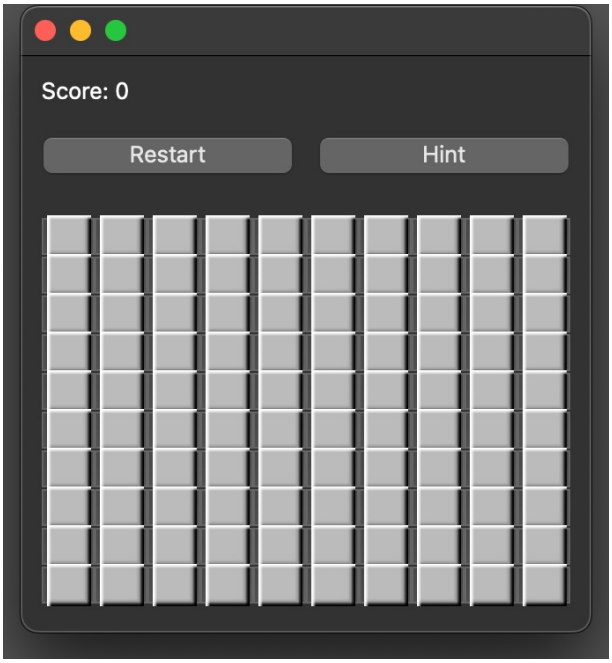

Figure 1: A 10x10 starting grid

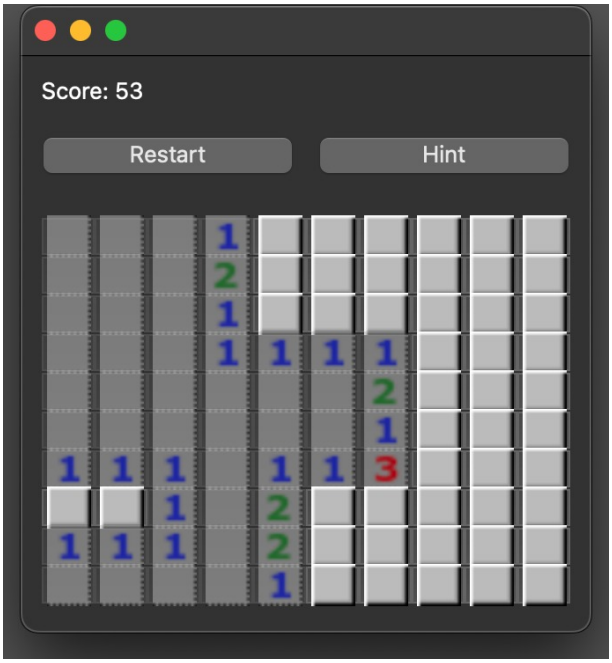

Figure 2: When an empty cell is clicked, all the empty cells near it are opened recursively

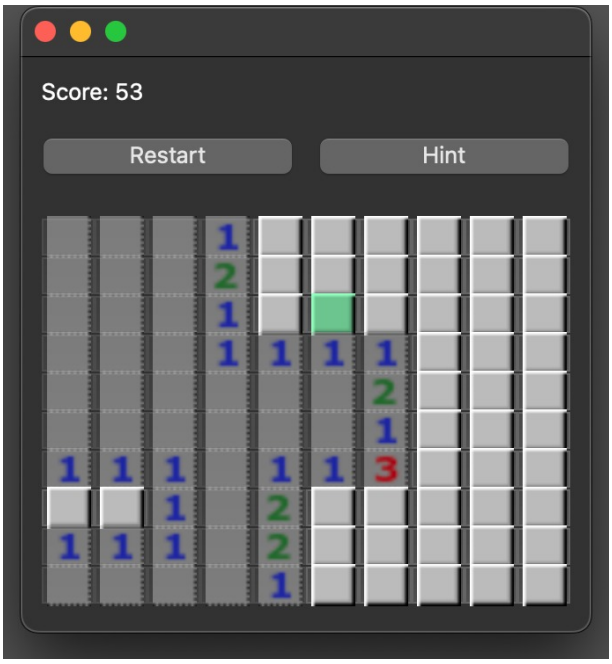

Figure 3: When the hint button is clicked, the hint cell becomes green

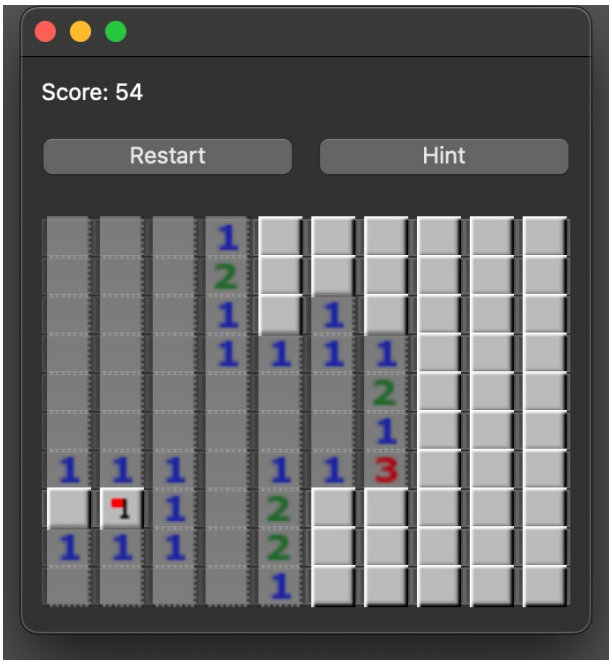

Figure 4: The hint button is clicked again, revealing the hinted cell

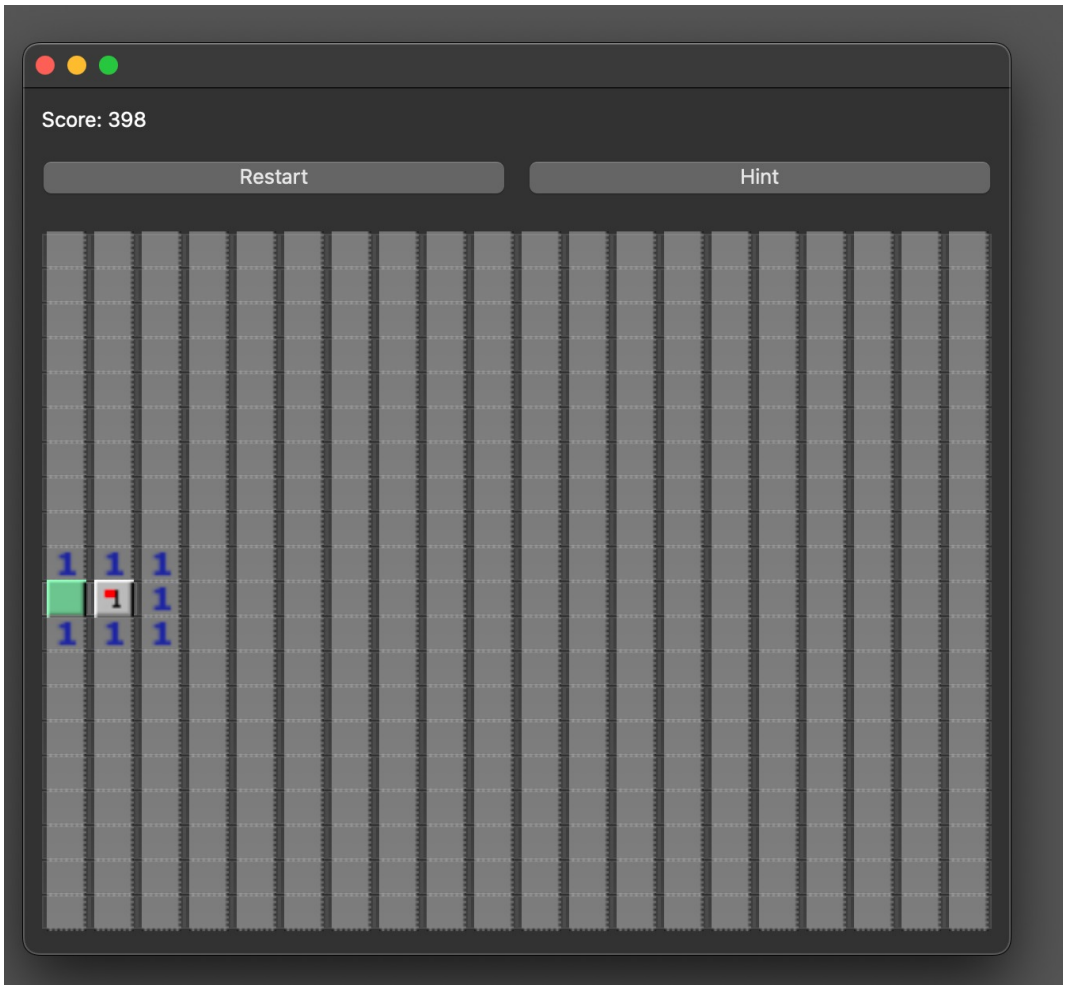

Figure 5: 20x20 grid with only 1 mine placed, the hint works correctly and all the other cells are opened recursively upon clicking an empty cell

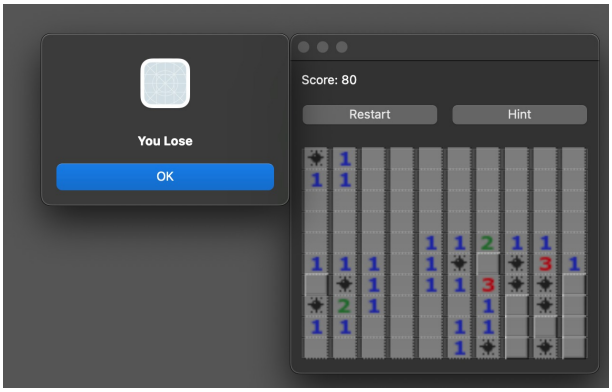

Figure 6: Upon clicking a mine, all the mines are revealed and you lose

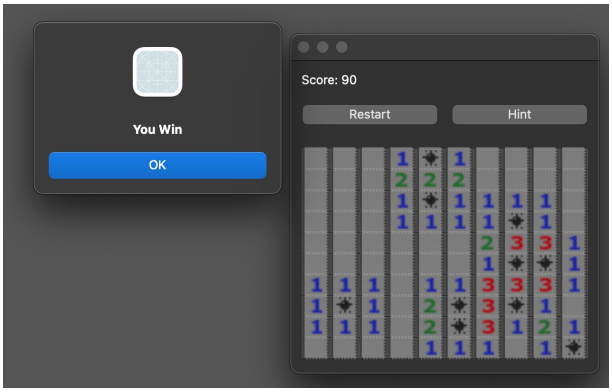

Figure 7: After revealing all the cells that are not mines, all the mines are revealed and you win

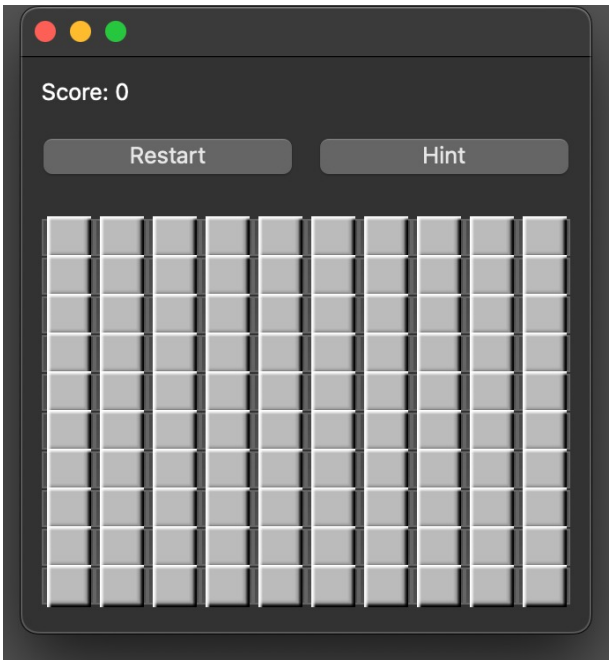

Figure 8: A new 10x10 upon clicking restart