CMPE230, Assignment 3

Abstract:

This is the documentation paper for the third assignment of the CMPE230 class. The program we implemented is a good old mine sweeper game. We have used C++ and the QT Framework to implement the game and design a GUI. The user can interact with the program by clicking on the cells and he/she can play the game just like it is played in its original version.

Structure and Initialization:

We have used the QT Creator environment to implement the program and we run it on there. We have started by creating the basics of the minesweeper game. Then we decided to use the QGridLayout object since the minesweeper field resembles a grid. We have 2 classes and a header file in our project (main.cpp, grid.cpp, grid.h). We have created a Grid class that inherits the QWidget object since we want our grid to behave like a window. After adjusting the properties of our grid and loading the assets, we create the layout for the info and action buttons above the game. After that, we initialize the board with mines, add a button in each coordinate of our grid, and set the icons of those buttons to provided cell images. By doing these, we end up a clone of the original game.

Taking Inputs and Sending Signals:

The next part is taking inputs from the user. We had to achieve this by sending signals to necessary functions based on the button the user clicks. The signal can be one of the following:

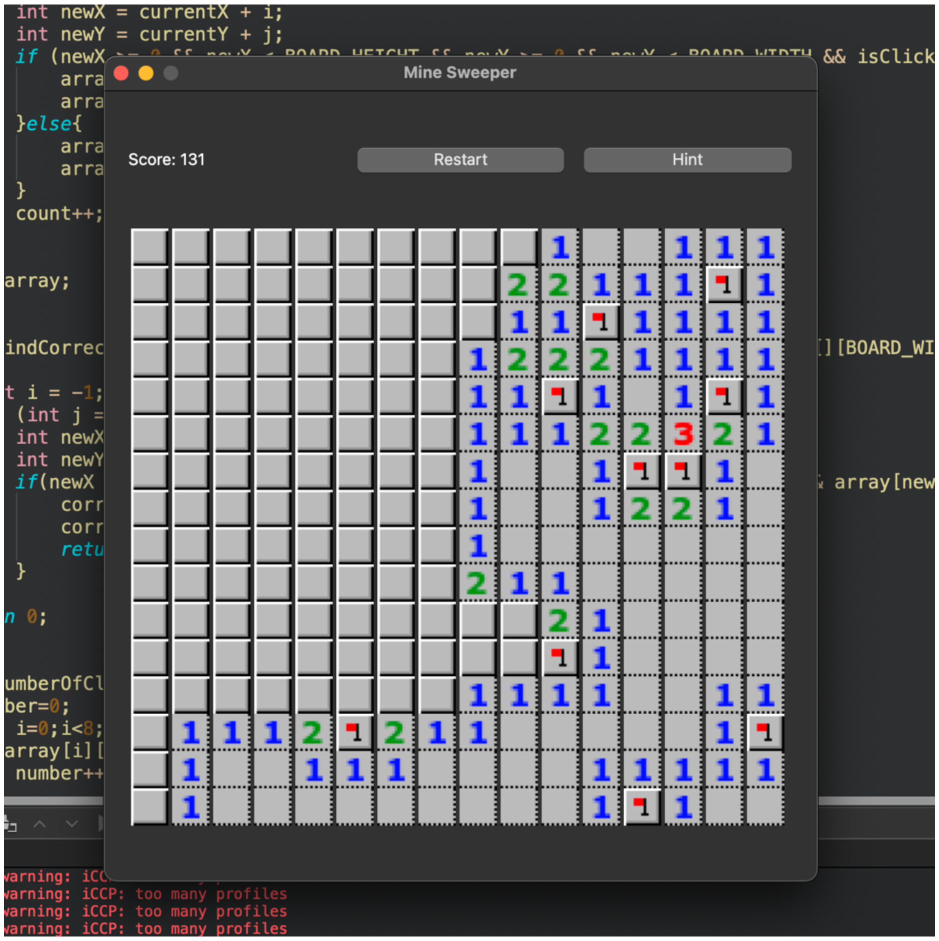
- Restart Button: to restart the game.
- Hint Button: to blink a cell (if possible) as a hint.
- Buttons on the Grid: the user can left-click or right-click on cells to play the game. We allow the user to click only on the unrevealed cells. Clicking left reveals the cell (neighboring cells as well if there is no mine nearby) and clicking right marks the cell with a flag. The user can toggle the flag by right-clicking again.

We have connected the buttons with those signals and each time the user tries to click on a button, our program takes the necessary action (if the cell has not already been clicked before). While we could write a signal class to take the right-click input from the user, we choosed to use the built-in functionalities of the QtGridLayout class and used the contextMenuPolicy property of the buttons. This way we could arrange the signal which will be sent when the user tries to access to the context menu for that button by right clicking.

Challenges and the Hint Algorithm:

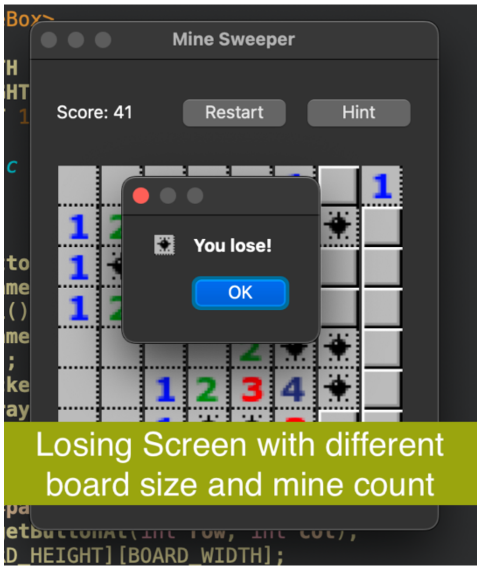
One of the harder parts of this assignment was giving hints to the user. Our hint algorithm works as follows. We iterate over each revealed cell and check the unclicked neighbors of it. Since we have the information of the neighboring mine count for each of them, we compare the mine count with the clickable cells around each cell. If the number of clickable cells around a cell is equal to neighboring mines for that cell, we mark those cells in a list since they are guaranteed to have mine. Afterward, we move on to next cells and check if all mines around them are marked. If so, the remaining cells around them which are clickable but not marked, can be given as a hint to the user since they are guaranteed to not have a mine. Since the main aim of the project is not coming up with perfect algorithms, our hint algorithm is not that complex and it doesn't evaluate the cells globally, meaning that while a player can infer the locations of a mine by looking at several cells at once and evaluating them simultaneously, our algorithm just checks the information from cells one-by-one.

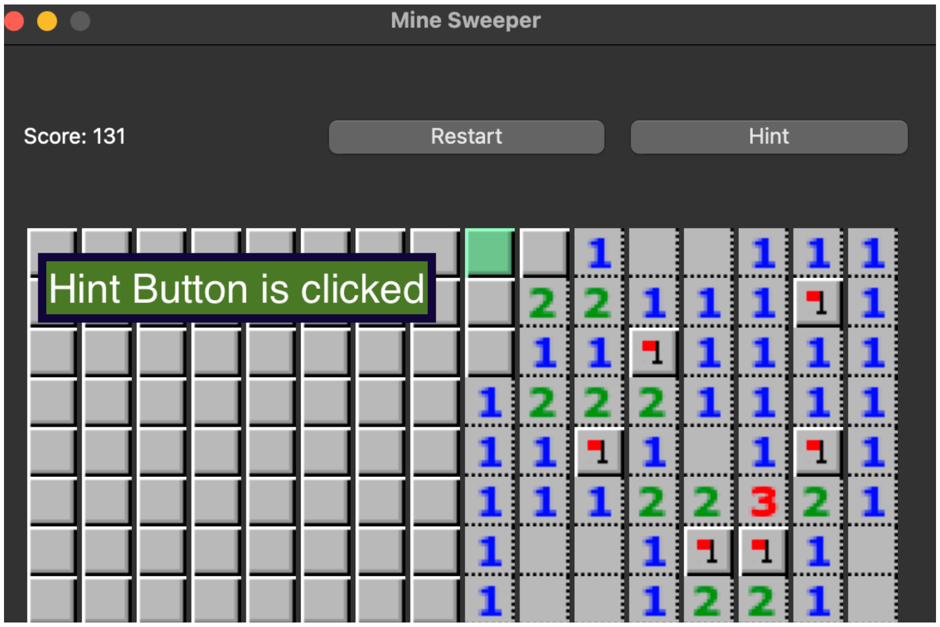
Other challenge which we have solved rather quickly was correctly identifying the coordinates of the clicked button. We needed to locate it to manipulate our 2D board array. We first couldn't manage to achieve this but then we found out that we could get the item from a grid layout object by the itemAtPosition property of it which takes x and y coordinates. Then by casting that object which is a button and comparing it with the pressed button, we could conclude the coordinates of a button. The rest is simply performing algorithms on arrays.

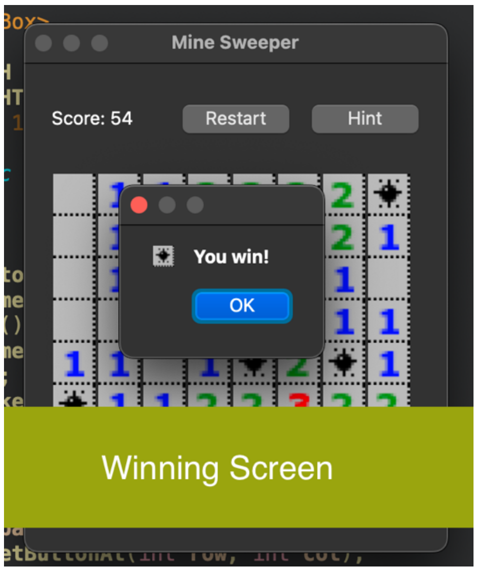


```
int newX = currentX + i;
int newY = currentY + j;
if (newX >= 0 && newY < BOARD_HEIGHT && newY >= 0 && newY < BOARD_WIDTH && isClick
    arra
    arra
}else{
    arra
    arra
}
count++;

array;

indCorrec                                                    ][BOARD_WI

t i = -1;
 (int j =
 int newX
 int newY
 if(newX                                                   & array[new
     corr
     corr
     retu
 }

n 0;


umberOfCl
ber=0;
 i=0;i<8;
array[i][
 number++
```

```
warning: iCC
warning: iCCP: too many profiles
warning: iCCP: too many profiles
warning: iCCP: too many profiles
```







This homework is done by Cengiz Bilal Sarı and Berkay Buğra Gök (2021400201-2021400258)