CMPE 230

Systems Programming

Project 3 - Minesweeper Game

Programming Project

SUBMISSION DATE: 02/06/2024

OZGUR SAVASCIOGLU 2022400366

SENOL SOLUM 2016400396

Table of Contents

1. Introduction

This project aims to implement the Minesweeper game. The main mechanic of the Minesweeper game is to reveal all cells without hitting any mine during the process. In other words, the user will aim to find the locations of all mines to avoid them and reveal non-mine cells during the game.

Since the game has an extensive popularity among the computer user community for a very long time there are different online resources to find detailed guides about gameplaying strategy and the game's rules. In this project, we used the website: https://minesweeper.online/ as the main source for the game rules, mechanics, and solution patterns.

To implement this project, we utilized Qt Creator as the development environment, with C++ as the primary programming language.

Our implementation includes essential game mechanics like left-click(revealing), right-click(flagging), recursive revealing for empty cells, restarting the game via the restart button, showing the score (# of revealed cells) in the game window, giving hint via hint button. In addition, we designed dedicated end-game screens for game-won and lost cases.

2. Program Interface

The start executing of the Minesweeper game is straightforward. Users either can directly start the executable file named *2022400366_2016400396* or run the program in QT Creator after building it. The executable file can be found under the build folder in the QT Project directory.

To end the program, there are several options. At the beginning, the user simply press cancel on the row, column, and mine input buttons. After the start of the program, the user can press the X button at the top right to end the program.

3. Program Execution

To create the executable of the program, we use the QT Creator interface and build command to create the executable file named *2022400366_2016400396*. The executable file can be found under the build folder in the QT Project directory.

4. Input and Output

The main inputs of the program are numbers for rows, columns, and mines. The user will determine these numbers at the start of each game. Three sequential screens as can be seen below will appear to collect this input.

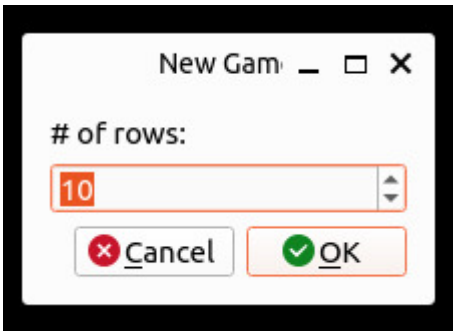

Figure 1: row input screen

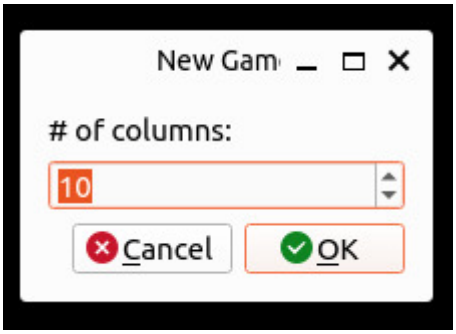

Figure 2: column input screen

Figure 3: mine input screen

Users can select the desired amount either by using the slider or directly writing the number with keyboard. The minimum allowed amount for rows and columns is 2. The maximum amount for these fields is 100. For mines, the minimum allowed is 1 and the maximum allowed is rows*columns. Each input has default values at the beginning. The default value for rows and columns is 10 and the default for mines is 4.

5. Program Structure

The program has the below listed files:

- 2022400366_2016400396.pro
- Headers
    - cell.h
    - gamegrid.h
    - gamewindow.h
    - recursionstack.h
- Sources
    - cell.cpp
    - gamegrid.cpp
    - gamewindow.cpp
    - recursion stack.cpp
    - minesweeper.cpp

The main of the program is under the minesweeper.cpp file. The main collects the initial user inputs for rows, columns, and mines. Then the main creates the corresponding *gamewindow* object and shows it on the screen.

5.1. Classes

5.1.1. gameWindow

The *gamewindow* not only include the gamegrid object under it but also includes the score label, hint and restart buttons. After the creation of initial gamewindow, the initial gamegrid is created which means creating the cells and fill them randomly with mines. Also, creating necessary connections for each cell, button, etc.

Here are the slots, their functions, and connected signals to them under the gamewindow class:

Slot Name	Function	Connected Signal(s)
changeScore	updates the score	myGrid, gameGrid::scoreUpdate
restartGame	restarts the game	restart, &QPushButton::clicked
giveHint	highlights a hint if possible	hint, &QPushButton::clicked
indirectHindEnd	if the hinted cell is revealed close the hint	myGrid, &gameGrid::indirectHintClose
disableHint	disable the hint button after the game end	myGrid, &gameGrid::closeHint

5.1.2. gameGrid

The gameGrid class, includes the allCells vector that stores the cells in the game. In addition, the class stores the data regarding # of rows, columns, mines, and unrevealed mines in the game. The class also includes the functions that are used for pattern checking for finding a hint cell.

Here are the slots, their functions, and connected signals to them under the gameGrid class:

Slot Name	Function	Connected Signal(s)
revealAllMines	show all mines in case of end game	• &gameGrid::youWin • allCells[index], &Cell::youLose
recursionCase	Start the recursion, in case of empty cell	allCells[index], &Cell::startRecursion
loseDialog	Opens game lost dialog screen	allCells[index], &Cell::youLose
winDialog	Opens game won dialog screen	&gameGrid::youWin
decreaseCounter	Decrease the number of remaining cells	• allCells[index], &Cell::showNumber • allCells[index], &Cell::showEmpty

Pattern checkers

Under the gameGrid class there are several pattern checker functions. Below you can find the table that explains each pattern with the corresponding visual representation.

Although there are different pattern-checking functions almost all of the test runs the first function finds the pattern if it is available. Kindly note that we run the first function twice to use the data from the first run for the upper cells on the second try.

Function Name	Pattern Reference from https://minesweeper.online/help/patterns	Visual
basic1Pattern	• B1 • B2	• Figure 4 • Figure 5
pattern121	1–2–1	Figure 6
pattern1221	1–2–2–1	Figure 7
pattern_One_One	• 1-1 • 1-1+	• Figure 8 • Figure 9

The below pattern visuals are taken from the https://minesweeper.online/help/patterns website:

Figure 4: B1 Pattern

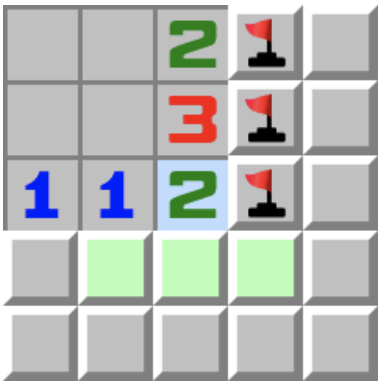

Figure 5: B2 Pattern

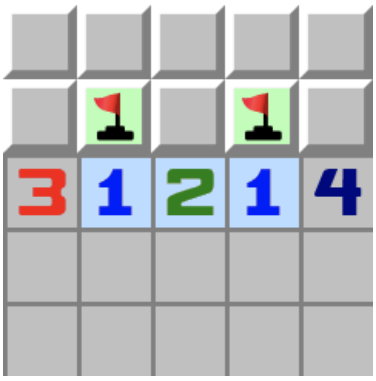

Figure 6: 1-2-1 Pattern

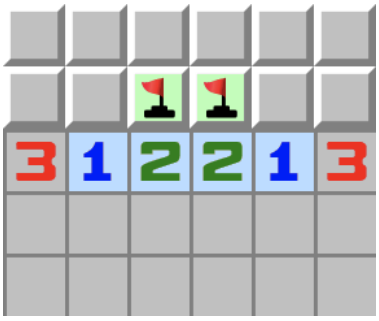

Figure 7: 1-2-2-1 Pattern

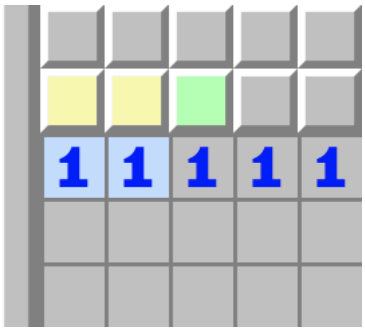

Figure 8: 1-1 Pattern

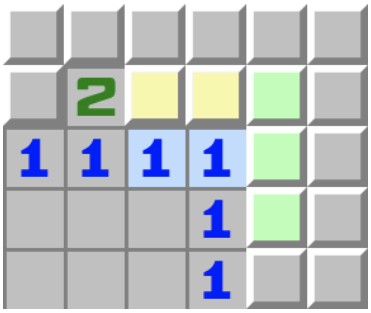

Figure 9: 1-1+ Pattern

5.1.3. Cell

The Cell class includes the data about the Cell objects in the game which are customized QPushButtons. These data fields include basic data: whether the cell has mine, flagged, total neighbors with mine, whether the cell is revealed or not, and whether the cell is a known mine by the user.

Here are the slots, their functions, and connected signals to them under the cell class:

Slot Name	Function	Connected Signal(s)
updateButtonText	show the number of neighbor mines	&Cell::showNumber
changeToEmpty	show the empty cell visual	&Cell::showEmpty
startMineCheck	check if there is a mine in the cell	&Cell::leftClick
insertFlag	insert a flag	&Cell::rightClick
closeExistingHint	Makes an hinted cell empty	&Cell::unclick

5.1.4. RecursionStack

This class is used to create the global stack that is used for the recursion case. The class has only one data field globalStack. It is an extern QStack<int> that stores the cell indexes.

6. Examples

Below you can find the screenshots from different stages of the game:

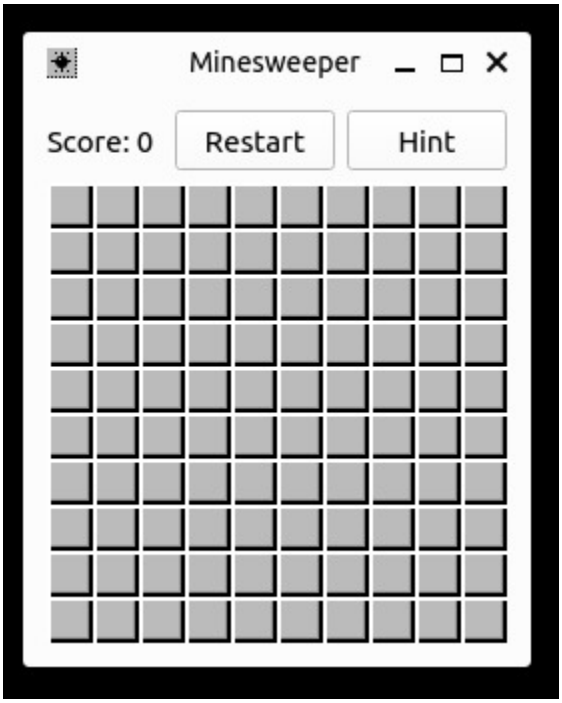

Figure 10: Sample Initial Game Screen

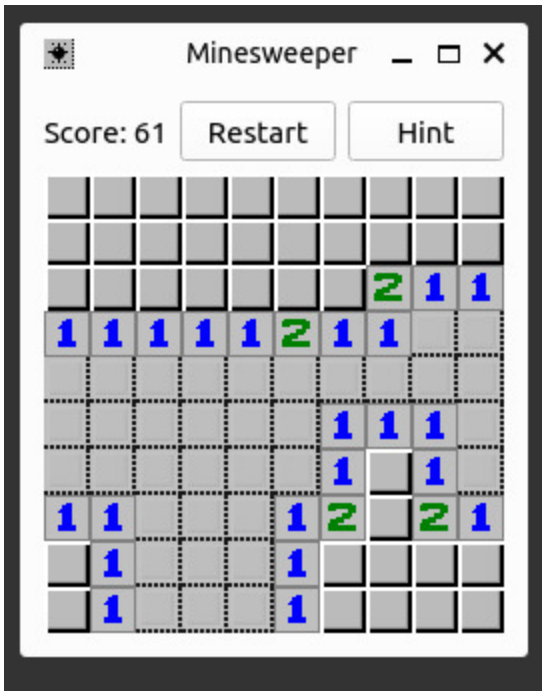

Figure 11: Sample Game Screen

Figure 12: Game Won Screen

Figure 13: Game Lost Screen

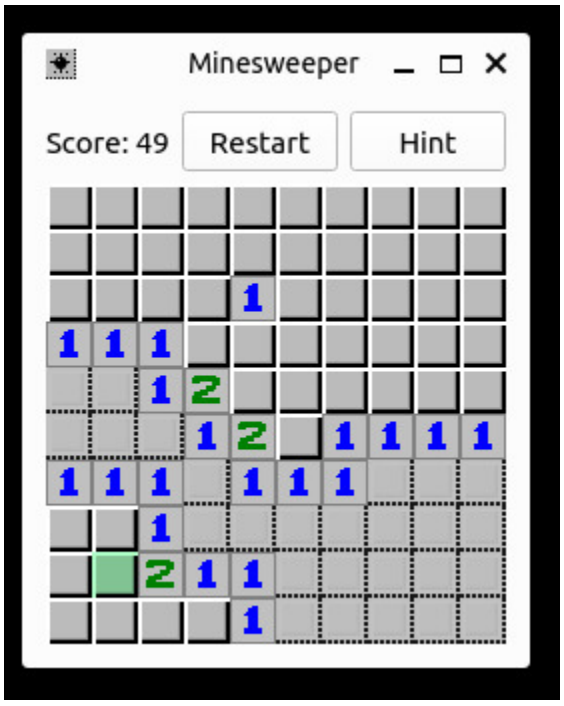

Figure 14: Hint button first press

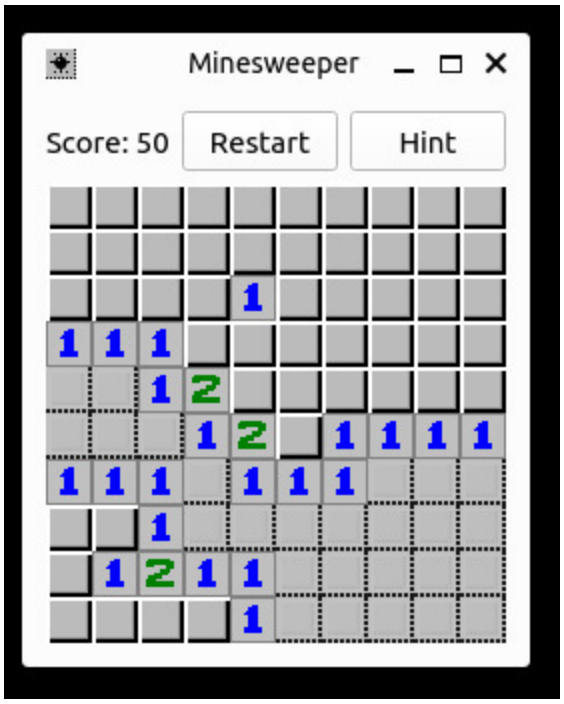

Figure 15: Hint button second press

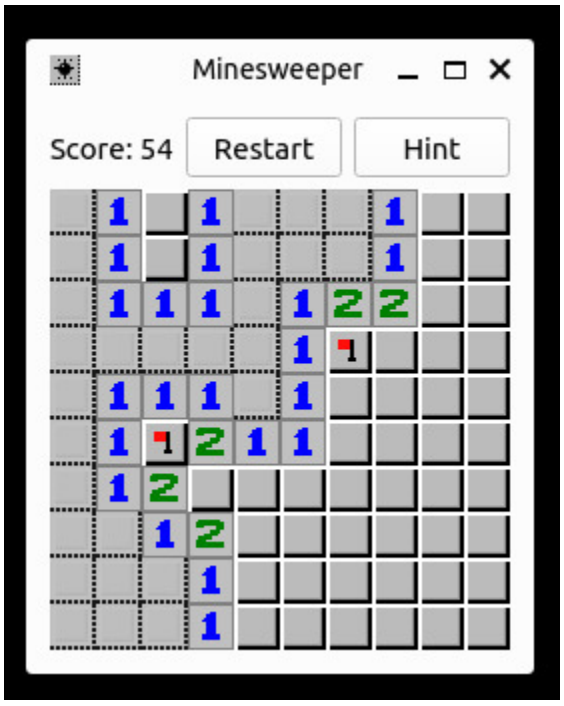

Figure 16: Flagged cells

7. Improvements and Extensions

Although the game is running as expected during the tests there is still room to improve the design visually. The main area could be creating a single row, column, and mine number input screen instead of three consecutive screens each time.

It is also possible to add a left-right click function which is not included in the project description but is a part of the classical version of the game.

8. Difficulties Encountered

Although one could think that the recursion and hint implementation are the most challenging parts to implement at first glance, these parts were not as challenging for our case as expected. On the other hand, getting used to the QT Creator environment created a significant challenge for us. The hardest problem for us is to understand the reason of grayscaling of some icons in the game. It turns out to be a result of disabling the relevant button. Our solution to this problem was overriding *paintEvent* function to keep the RGB format.

As for the edge cases, we need to be careful about the cells at the border in our calculations. This type of cells created additional checks and different types of calculations in our algorithm.

9. Conclusion

In conclusion, we successfully created our version of the Minesweeper game in this project by using the QT Creator environment and C++ programming language.

This project not only helped us to learn the general structure of the QT Creator but also increased our competence in the C++ language.

After gaining the initial understanding of the signal and slot structure of the QT Creator programming it became easier for us to implement the rest of the project.

10. Appendices

We used the below websites as references for the game rules, mechanics, and patterns:

- https://minesweeper.online/help/patterns
- https://minesweepergame.com/strategy/how-to-play-minesweeper.php#:~:text=Minesweeper%20is%20a%20game%20where,mine%20you%20lose%20the%20game!