CMPE 230 Systems Programming Project 3 - Minesweeper Game

02/06/2024

Buğra Keser 2021400144
Yusuf Anıl Yazıcı 2021400207

Table of Contents

1.Introduction

The objective of this project is to develop a Minesweeper game using C++ and the Qt framework. This involves creating a graphical user interface, implementing mine placement and detection logic, and handling user interactions. The goal is to deliver a polished and enjoyable Minesweeper game that highlights the capabilities of Qt and C++.

Minesweeper is a solo logic-based computer game played on a rectangular grid. The goal is to uncover a set number of hidden "mines" by clicking on "safe" squares while avoiding those with mines. If a mine is clicked, the game ends. Otherwise, a number from 0 to 8 appears, indicating the count of mines in the eight adjacent squares. For instance, finding an "8" means all eight neighboring squares have mines, while a "0" (shown as a blank) means no mines are nearby. Players can mark suspected mine squares with a flag.

Qt is a powerful cross-platform application development framework widely used for creating graphical user interfaces (GUIs) and cross-platform applications. Written in C++, Qt allows developers to build applications that can run on various operating systems like Windows, macOS, Linux, iOS, and Android without modifying the source code. It provides a comprehensive set of libraries and tools, including support for widgets, signals and slots for event handling, and an integrated development environment (Qt Creator). Qt's flexibility and extensive features make it a popular choice for both open-source and commercial software development.

2.Program Interface

This section explains how the user communicates with the program. Our Minesweeper game is designed to run on a macOS system. To run the program, Qt Creator should be used as the development environment. Here are the steps to activate and deactivate the program:

Activating the Program

1. **Setting up the Programming Environment:**

    - Ensure that Qt Creator is installed on your macOS system. You can download it from the Qt official website.

    - Open Qt Creator and load the Minesweeper project by selecting File > Open File

or Project and navigating to the project's .pro file.

2. **Running the Program:**

    - Once the project is loaded in Qt Creator, you can modify the game parameters such as the number of rows, columns, and mines. These parameters can be changed in the MainWindow.cpp file.

    - To build and run the program, click on the green "Run" button located at the bottom left corner of the Qt Creator interface or use the cmd + R shortcut.

    - The game will start and the graphical user interface will be displayed, allowing you to play Minesweeper.

Deactivating the Program

- To terminate the program, simply close the Minesweeper application window by clicking the close button (X) on the window.

- Alternatively, you can stop the running process using the cmd + Q shortcut.

3.Program Execution

Game Start

When the game is launched, the main window of the Minesweeper game is displayed. The user is presented with a grid of covered squares and a menu bar at the top.

Main Window Components

- **Grid:** The primary area where the game is played. Each cell in the grid can be a mine or a safe square.
- **Menu Bar:** Contains the score label and options such as 'Restart' and 'Hint'.

Setting Rows, Columns, and Mines

At the start of the game, the user can set the number of rows, columns, and mines by modifying the hard-coded values in the `MainWindow.cpp` file. To do this:

1. Open the `MainWindow.cpp` file in Qt Creator.
2. Locate the section of code where the game parameters are defined, typically at the beginning of the file.
3. Change the values of the variables representing the number of rows, columns, and

mines.

```
MinesweeperWindow::MinesweeperWindow(QWidget *parent) :
QMainWindow(parent), totalMines(20), rows(10), cols(20),
score(0), gameOver(false)
```

4. Save the changes and rebuild the project by clicking on the green "Run" button or using the cmd + R shortcut.

User Inputs

The user interacts with the game primarily through mouse clicks:

1. **Left Click:**
    - Reveals the content of the selected square.
    - If a mine is revealed, the game ends.
    - If a number between 1 and 8 is revealed, it indicates the number of mines in the adjacent squares.
    - If a blank square (0) is revealed, it indicates there are no mines in the adjacent squares, and it will automatically reveal surrounding squares.
2. **Right Click:**
    - Flags a square as suspected to contain a mine. This helps in marking potential danger zones.
    - Right-clicking again on a flagged square will remove the flag.

Outputs

- **Revealed Numbers:** When a safe square is clicked, a number between 1 and 8 is shown, indicating the count of mines in the adjacent squares.
- **Flagged Squares:** Squares marked by the user as suspected to contain mines.
- **Game Over Message:** If the user clicks on a mine, a game over message is displayed, indicating the end of the game.
- **Victory Message:** When all non-mine squares are revealed, a victory message is displayed, indicating the user has successfully located all mines.
- **Score Label:** The score is displayed at the top of the game window and is incremented with each revealed safe cell. The initial score is 0, and it increases as the player reveals more safe squares. This score provides feedback on the player's progress throughout the game.

Menu Options

The Minesweeper game includes the following options accessible via buttons in the main window:

Restart

- The "Restart" button resets the game to start a new session. The grid is refreshed with a new random distribution of mines, and the score is reset to 0.

Hint

- The "Hint" button provides a hint to the player by revealing a safe square. This can help in progressing through the game by avoiding potential mines.

4.Input and Output

Inputs

The primary inputs in the Minesweeper game are user interactions through mouse clicks.

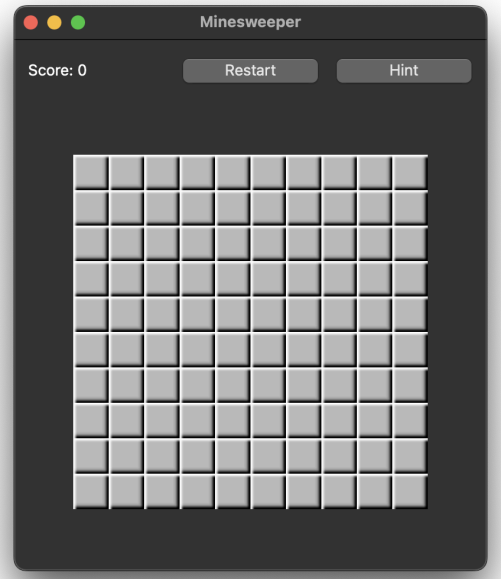
1. **Left Click on a Safe Square:**
    - **Description:** The user clicks on a square that does not contain a mine
    - **Explanation:** The square is revealed, showing a number (between 1 and 8) indicating the count of mines in the adjacent squares.
2. **Left Click on a Mine:**
    - **Description:** The user clicks on a square that contains a mine.
    - **Explanation:** The game ends, and a game over message is displayed.
3. **Right Click to Flag a Square:**
    - **Description:** The user right-clicks on a square to mark it as suspected to contain a mine.
    - **Explanation:** A flag icon appears on the square, indicating it is flagged as a potential mine.
4. **Right Click to Unflag a Square:**
    - **Description:** The user right-clicks on a flagged square to remove the flag.
    - **Explanation:** The flag is removed, and the square returns to its original covered state.
5. **Left Click on the Restart Button:**
    - **Description:** The user clicks the "Restart" button to reset the game.
    - **Explanation:** The game grid is refreshed with a new random distribution of mines, and the score is reset to 0.
6. **Left Click on the Hint Button:**

- **Description:** The user clicks the "Hint" button to receive a hint.
- **Explanation:** A safe square is revealed, helping the player to progress without hitting a mine.

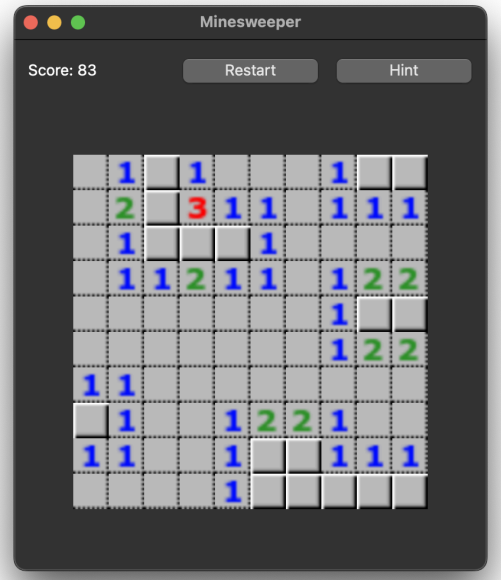
Outputs

The outputs in the Minesweeper game include revealed numbers, flagged squares, score updates, game over messages, and victory messages.

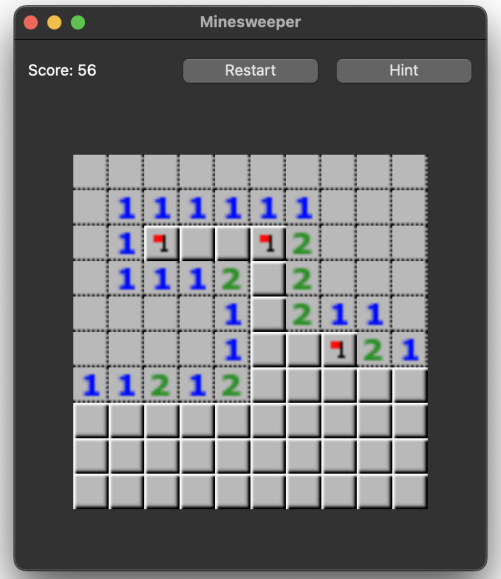
- **Initialized Game Screen:**
  - **Description:** The game starts with a grid of covered squares and a menu bar at the top.
  - **Explanation:** This is the default view when the game is first launched, showing the grid where the player will begin interacting.



- **Revealed Numbers:**
  - **Description:** When a safe square is clicked, a number between 1 and 8 is shown, indicating the count of mines in the adjacent squares.
  - **Explanation:** The numbers help the player deduce the locations of mines.

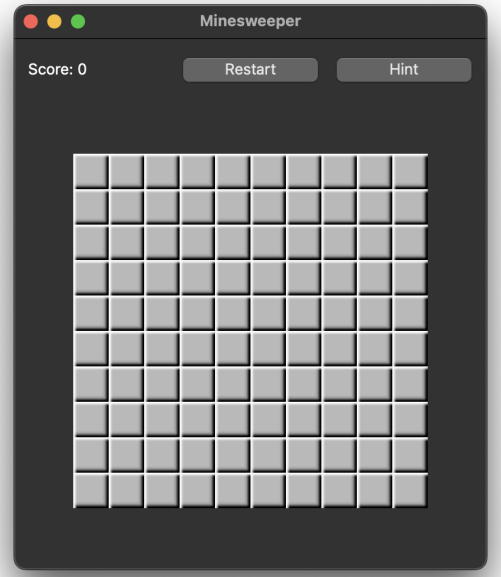


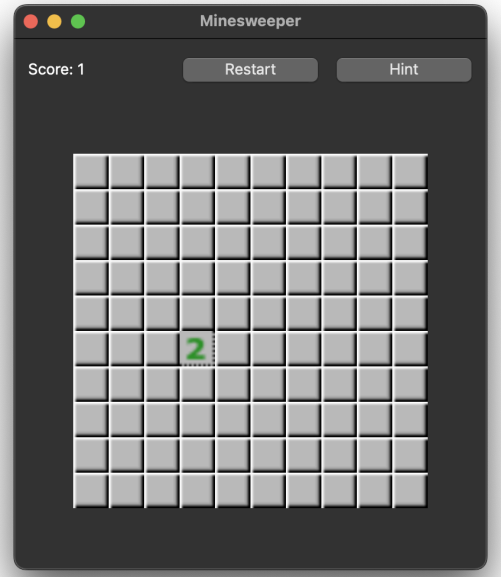
- **Flagged Squares:**
  - **Description:** Squares marked by the user as suspected to contain mines.
  - **Explanation:** Flags assist the player in keeping track of potential mine locations.



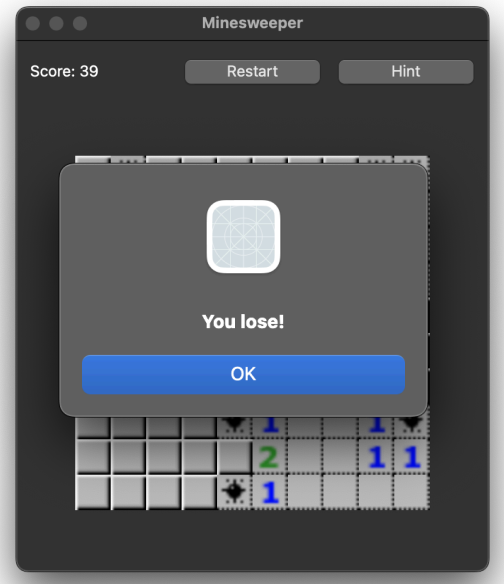
- **Score Update:**
  - **Description:** The score is incremented with each revealed safe cell.
  - **Explanation:** The score label at the top of the window updates, reflecting the

number of safe cells revealed by the player.



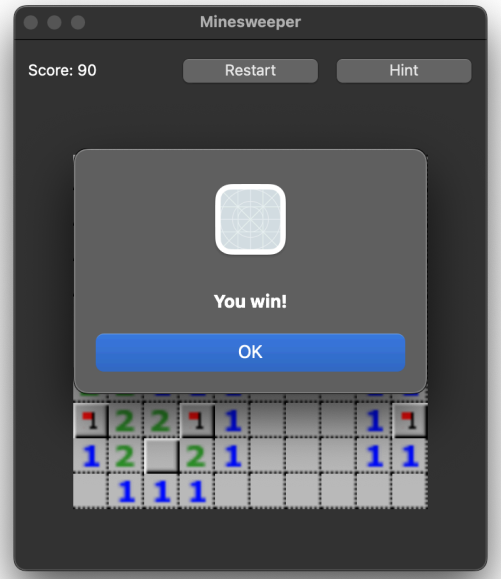


- **Game Over Message:**
  - **Description:** If the user clicks on a mine, a game over message is displayed.
  - **Explanation:** The game ends, informing the player that they have clicked on a mine.

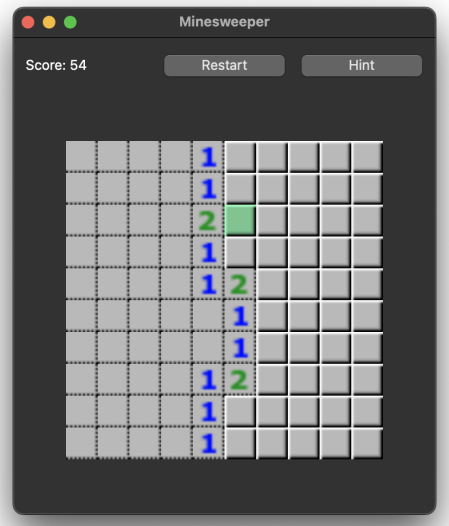


- **Victory Message:**
  - **Description:** When all non-mine squares are revealed, a victory message is displayed.

- **Explanation:** The game congratulates the player for successfully locating all mines.



- **Hint:**
  - **Description:** When the "Hint" button is clicked, a safe square is revealed to the player.
  - **Explanation:** The hint provides assistance by uncovering a safe square, helping the player progress without hitting a mine.



5.Program Structure

Design

The Minesweeper program consists of the following modules:

Cell.h & Cell.cpp

The `Cell` class represents each cell in the Minesweeper grid with the following data fields:

- **`mine`:** Boolean indicating if the cell contains a mine.
- **`revealed`:** Boolean indicating if the cell has been revealed.
- **`flag`:** Boolean indicating if the cell is flagged.
- **`hint`:** Boolean indicating if the cell is marked by a hint.
- **`adjacentMines`:** Integer representing the number of mines adjacent to the cell.
- **`imagePath`:** QString holding the path to the cell's icon image.
- **`icon`:** QIcon representing the visual state of the cell.

Key methods include:

- **`reset()`:** Resets the cell to its default state.
- **`updateIcon()`:** Updates the cell's icon based on its current state.

CustomButton.h & CustomButton.cpp

The CustomButton class is a subclass of QPushButton that can differentiate between left and right clicks. It includes:

- **Signals:**
  - **`leftClicked()`:** Emitted when the button is left-clicked.
  - **`rightClicked()`:** Emitted when the button is right-clicked.
- **Protected Method:**
  - **`*mousePressEvent(QMouseEvent event)`:** Handles mouse click events and emits the appropriate signal based on whether the click was left or right.

MainWindow.h & MainWindow.cpp

The MainWindow class manages the game interface and logic with the following components:

Data Fields:

- **`totalMines`:** Number of total mines in the game.
- **`minesIndexes`:** QVector containing the indexes of the mines.
- **`cells`:** QVector containing all the cells.

- **`buttons`:** QVector containing all the custom buttons.
- **`rows`, `cols`:** Number of rows and columns in the grid.
- **`score`:** Current score of the game.
- **`scoreLabel`:** QLabel displaying the current score.
- **`gameOver`:** Boolean indicating if the game is over.
- **`knownMines`:** QVector containing indexes of confirmed mines.
- **`safeCells`:** QVector containing indexes of confirmed safe cells.

Key Methods and Slots:

- **`giveHint()`:** Provides a hint by revealing a safe cell.
- **`findSafeCells()`:** Identifies cells that are safe based on a simple algorithm.
- **`revealAllMines()`:** Reveals all mines when the game ends.
- **`resetGame()`:** Restarts the game.
- **`distributeMines()`:** Randomly distributes mines on the grid.
- **`calculateAdjacentMines()`:** Updates the number of adjacent mines for each cell.
- **`revealCell(int index)`:** Reveals the cell at the specified index.
- **`attachFlag(int index)`:** Attaches a flag to the cell at the specified index.
- **`checkWinCondition()`:** Checks if the game has been won.
- **`updateScore()`:** Updates the score based on the number of revealed safe cells.

The class initializes the game interface, sets up the grid, handles user interactions, and manages the game state.

main.cpp

Initializes the Qt application and sets up the main window, starting the game. It serves as the entry point of the application.

Implementation Details

Initializing the Game

The `MainWindow.cpp` creates the specified grid with the given row and column values, randomly distributes the specified number of mines across the grid using the `distributeMines()` function, and assigns the relevant `adjacentMines` values to each cell using the `calculateAdjacentMines()` function.

Gameplay

- **User Interaction:**
  - **Left Click:** Handled by the `mousePressEvent(QMouseEvent *event)` function in `CustomButton.cpp`. When the user clicks a cell, the `revealCell()` function is called and it checks if the cell contains a mine. If it does not, the cell is revealed and updates the icon using `updateIcon()`. If the

cell has no adjacent mines, for all the adjacent cells, the `revealCell()` function is called to recursively reveal neighboring cells. If the cell contains a mine, the `gameOver` boolean is set to true and the revealAllMines() function is invoked.

- **Right Click:** Also handled by `mousePressEvent(QMouseEvent *event)`, a right-click on a cell calls the `attachFlag()` function, which attaches a flag to the cell, marking it as a suspected mine. A second right-click invokes the `attachFlag()` function again and removes the flag.

- **Score System:**
  - The `updateScore()` function in `MainWindow.cpp` increments the score for each revealed cell that is not a mine. The score label is updated in the user interface to reflect the current points earned by the user.

End Game

- The `gameOver` boolean determines if the game ends when all cells except mines are revealed or the user clicks on a mine.
- Upon game over, the `revealAllMines()` function is called, which reveals all mines.
- A new game is initialized with default values when the restart button is clicked, triggering the `resetGame()` function again.

Hint Feature

The hint feature is implemented in `MainWindow.cpp` through the `giveHint()` function. It provides suggestions for safe cells based on the user's current perspective. It uses basic patterns such as:

1. **Mine Identification:**
   - If a number is touching the same number of cells, then those cells are identified as mines.
   - If a number is touching the same number of known mines, then all adjacent cells are identified as safe and can be opened.
2. **Safe Cell Suggestion:**
   - After identifying safe cells using the `findSafeCells()` function, the `giveHint()` function suggests one of them to the user.
   - A second click on the Hint button reveals the suggested safe cell.

6.Examples

To understand the program better, here are some gameplay examples with video links:

Example 1: Basic Gameplay

Input: Start a new game, click on several cells, flag suspected mines.

Output: Revealed cells show numbers or are blank, flagged cells show flags, score updates.

Video Link:
https://drive.google.com/file/d/1uSuKYsgk03SGCs8B5K910sqwo36R8uRA/view?usp=share_link

Example 2: Game Over

Input: Start a new game, click on a cell containing a mine.

Output: Game over message, all mines revealed, no more clicks allowed.

Video Link:
https://drive.google.com/file/d/18f9ulOnAyJZPPOk-2GXV2JIWugp9-wQC/view?usp=sharing

Example 3: Using Hints

Input: Start a new game, click the hint button.

Output: Safe cell revealed, score updates.

Video Link:
https://drive.google.com/file/d/1tslnkzK1pvB8MK5d63Y3FZSaVmu8pQP9/view?usp=share_link

Example 4: Winning the Game

Input: Start a new game, reveal all safe cells without hitting a mine.

Output: Victory message, final score displayed.

Video Link:
https://drive.google.com/file/d/1uSuKYsgk03SGCs8B5K910sqwo36R8uRA/view?usp=share_link

These examples demonstrate the game's functionality and different outcomes.

7.Improvements and Extension

In this section, we discuss areas of the program that need improvement and potential future extensions. We also identify the strengths and weaknesses of the program, along with any deviations from the initial project plan.

Areas for Improvement

1. **Hint Algorithm:**
    - **Current State:** The current hint algorithm provides basic hints by revealing a safe cell based on simple patterns.
    - **Improvement:** The hint algorithm can be expanded to recognize and handle more complex patterns, providing more accurate and helpful hints to the user.
2. **Cross-Platform Compatibility:**
    - **Current State:** The application is designed to run on macOS.
    - **Improvement:** Extending the application to be cross-platform, so it can run seamlessly on Windows and Linux, would increase its usability and reach.

By addressing these areas for improvement and extending the program's capabilities, we can enhance the overall user experience and broaden the application's applicability.

8.Difficulties Encountered

1. **Implementing Right-Click Functionality:**
    - **Challenge:** Detecting both left and right clicks was essential for the game's functionality. However, `QPushButton`, the standard button class in Qt, does not inherently support this differentiation.
    - **Solution:** To address this, the `CustomButton` class was created as a subclass of `QPushButton`. This subclass overrides the mousePressEvent function to detect and emit signals for both left and right clicks.
    - **Impact:** This solution allowed for effective flagging and unflagging of cells, enhancing the user interaction with the game.
2. **Algorithm for the Hint Button:**
    - **Challenge:** Developing an algorithm to provide useful hints was complex. The goal was to implement a pattern recognition system that could identify and suggest safe cells.
    - **Solution:** A basic pattern recognition algorithm was implemented, which recursively identifies safe cells until no further safe cells can be found.
    - **Impact:** Although the hint functionality is basic, it provides helpful guidance to the player. However, the limitations of the current algorithm mean that it does not recognize all possible safe cell patterns, highlighting an area for future improvement.

9.Conclusion

The development of the Minesweeper game using Qt and C++ has been a valuable educational project, providing practical experience in software design, implementation, and problem-solving. Throughout the project, we successfully created a functional and user-friendly game that demonstrates key programming concepts and techniques.

Key Achievements

- **Functional Gameplay:** The core gameplay mechanics, including mine distribution, cell revealing, flagging, and game-over conditions, were implemented effectively.
- **User Interface:** A clean and intuitive user interface was developed, enhancing the user experience and making the game easy to play.
- **Modular Design:** The program's modular structure, with clearly defined classes and methods, facilitates maintenance and potential future extensions.

Challenges and Learning

- **Right-Click Functionality:** Implementing the `CustomButton` class to handle both left and right clicks was a significant challenge that was successfully overcome.
- **Hint Algorithm:** Developing a basic pattern recognition algorithm for the hint feature provided valuable insights into recursive algorithms and pattern detection.

Areas for Improvement

- **Advanced Hint Functionality:** Expanding the hint algorithm to recognize more complex patterns and provide more accurate hints is a key area for future development.
- **Cross-Platform Support:** Extending the application to support multiple operating systems would broaden its usability and reach.

Final Thoughts

Overall, this project has been a rewarding experience, offering practical insights into game development, user interface design, and algorithm implementation. The lessons learned and skills gained will be valuable for future projects and professional development. The Minesweeper game, as it stands, is a solid foundation that can be built upon and enhanced in future iterations.

10.Appendices

The complete source code and assets folder for the Minesweeper game is included in the same folder as this documentation. The source code files are:

1. **Cell.h**
2. **Cell.cpp**
3. **CustomButton.h**
4. **CustomButton.cpp**
5. **MainWindow.h**
6. **MainWindow.cpp**
7. **main.cpp**
8. **assets.qrc**
9. **Minesweeper.pro**

These files together form the complete implementation of the Minesweeper game. For detailed examination and reference, please refer to these files directly.