Card Match - Documentation

Yusuf Kağan Çiçekdağ & Ömer Faruk Erzurumluoğlu

May 2023

1 Description Of The Project

The aim of this project is to write a simple card match game. In this game, the player tries to remember and match cards that has the same word written on them. If the player matches all cards in a limited number of turns, he/she wins and loses otherwise.

2 How To Use The Program

After the program is run, the game starts. In each turn, the player can open two cards one by one by pressing onto their corresponding buttons. When two cards are opened, the player's remaining number of tries decreases by one and the player can see them open for a while. If they have the same word on them, the player gets a point and the matched cards vanish. Otherwise they got closed again. The player wins the game if he/she can match all cards before running out of tries and loses otherwise.
When the player pushes the restart button, the game restarts regardless of the situation of the current game.

3 Implementation

Since this is a simple game, we used only one file to write this code. As a result, we have used lambda functions to define functions which run after getting the corresponding signal. Since this is a Qt project, we have used Qt design principles. And since this is a game, we used styles to make it more appealing to the eye.

4 Dependencies

- **QApplication:** The Qt application class that manages the application's control flow and event handling.
- **QVBoxLayout:** A layout class that arranges widgets vertically.
- **QMessageBox:** A dialog class for displaying messages or notifications.

- **QWidget:** The base class for all UI objects in Qt.
- **QGridLayout:** A layout class that arranges widgets in a grid.
- **QPushButton:** A button widget class.
- **QLabel:** A widget class for displaying text or images.
- **QTimer:** A timer class for scheduling events.
- **random:** A standard library used for random number generation, shuffling, etc.
- **QFile:** A class for reading from files and writing to files.
- **QIcon:** A class for handling application icons.
- **QResource:** A class for accessing resources compiled into the application.

5 Global Variables

- **std::array<QString, 30> shuffledArray:** Shuffled array of character names
- **QPushButton *cardsTable[5][6]:** Array of buttons (except the restart button)
- **int numberOfRemainingTries:** Number of remaining tries of the current game.
- **int score:** Score of the current game
- **QLabel *topRow[4]:** Array of top row labels (note that the restart button is not included here)
- **QLabel *middleRow:** Middle row label (which is just an empty label)
- **QPushButton *restartButton:** Restart button of the game
- **int fittx:** x-coordinate of the first opened card, which is held before pushing the second button and comparing their strings
- **int fitty:** y-coordinate of the first opened card, which is held before pushing the second button and comparing their strings
- **bool wasEnabled[5][6]:** Array to store the state of buttons. By saying "states", we indicate whether the button is enabled or not.

6 Functions

- **std::array<QString, 30> nameArrayShuffler():** Returns a shuffled version of a predefined constant array (which is the array of our character names, which are Sans,Mettaton,Remilla,... etc.).
- **void disconnectAll():** Disables all of the buttons except the restart button
- **void freeButtons():** Enables the buttons which have been enabled before using *disconnectAll()* (except the restart button).
- **void sameButtonFunc(int x, int y):** This will be called if the two pressed buttons' corresponding character names are same. This function temporarily disconnects the matched cards' buttons and makes them invisible (until restarting the game) and changes other variables accordingly.
- **void diffButtonFunc(int x, int y):** This will be called if the two pressed buttons' corresponding character names are different. In this case, we disconnect all buttons for 1 second, then we re-connect the non-matched cards' buttons and restart button, change the appropriate global variables and continue the game.
- **void restarter():** Restarts the game. In other terms, it initializes the global variables, re-shuffles the buttons' corresponding character names and starts a new game.
- **int main(int argc, char *argv[]):** Sets up the application.

7 User Interface

The user interface consists of three grid layouts arranged vertically:

1. **Grid Layout 1:** Contains four labels and a button, which are "Score" text, the score itself (as a number, not just the text "Score"), "No. of Tries Remaining" text, the numberOfRemainingTries itself (as a number, not just the text "No. of Tries Remaining"), and a restart button at the right of the grid.
2. **Grid Layout 2:** Contains a single empty label in the middle of the board.
3. **Grid Layout 3:** Contains a 5x6 grid of buttons at the lower part of the board, representing the game cards.

8 Algorithm

1. First, we start the program, and the main function starts
2. Then we create our QApplication app

3. Then, we initialize resources to use images and styles in this project.

4. After that, we initialize the labels which will be inserted at the topmost grid, create a restart button and create a function which will be called when the restart button is pushed (which is restarter function).

5. Then, we create a window and set its title and create a main layout which we will insert our grids into.

6. Then, we create the first (and indeed, topmost) layout and assign corresponding labels and restart button, which are mentioned in step 4. We also arrange the size policies, maximum height limits, column and row stretches during this process.

7. After that, we create the second layout, add middleRow inside it, and set stretches, size policies and maximum height as well.

8. Then, we create the third layout. We first initialize our buttons and create the functions which will be called when we press one of these buttons. After that, we fill in the third grid with these buttons while arranging size policies, row and column stretches of these buttons.

9. After doing all these, we fill in the mainLayout with our three grids, and call restarter() function in order to start the game.

10. Then, we execute the application.

11. At the application window, if not all cards are invisible on the board, the player can click a random button.

12. If there is an open card on the board, the player can click a visible button (except the button clicked at the previous step) and the two opened cards are shown for 1 second. During this time, all buttons are temporarily disconnected.

13. If the two open cards' texts are same, the score is increased by 1, numberOfRemainingTries is decreased by 1, the matched cards are disconnected on the board, and they got darkened. Then we re-connect other buttons. If the game does not end, we continue the game.

14. If the two open cards' texts are different, we re-connect our non-matched pair, numberOfRemainingTries is decreased by 1, and the cards' texts are turned into "?" again. Then we re-connect other buttons. If the game does not end, we continue the game.

15. If the score is equal to 15 (namely, all pairs are found), the game ends. We create a QMessageBox with the text "Congratulations brother!" and a cancel button. After we press the cancel button, the finished game board appears and the player can either restart the game (namely, start a new game) or just quit the app.

16. If the numberOfRemainingTries is equal to 0 and the score is not equal to 15, we reveal all character names of the corresponding buttons, enable the restartButton, create a QMessageBox with the text "Mission failed!" and a cancel button. After we press the cancel button, all buttons except restart button are disconnected on the board which interrupts the player to keep going. The player can either start a new game or quit the app.

17. Note that at any step, we can push the restart button. When we push the restart button, it creates a new game, sets global variables and our board accordingly and starts a new game.