CMPE 230

İrem Nur Yıldırım

Başak Tepe

Pairs Card Game

Systems Programming Course Project

Submission: 23.05.2023

1. Introduction

In this project, we were expected to implement a simple matching card game using the QT framework for C++ language. The game consists of 15 card pairs and it is played through a GUI in a 6x5 card layout.

2. Program Interface and Program Execution

For the activation of the program, the source code can be compiled with the "qmake -makefile" command in the terminal. Afterwards, the Makefile can be run with the "make" command, and an executable is formed. Executable name is "MatchingCardsGame". After running the executable with the command "./MatchingCardsGame", a new window opens, displaying the card pairs game.

3. Gameplay

When a new game starts, the user first picks a card and the card back is revealed. Then, the user gets to pick a second card, whose backside is also displayed shortly after the clicking event. If the backsides of these two cards match, the user has made a correct guess and therefore gains +1 score. If the two cards do not match, they are turned over again and the user gets 0 points for the round. The user is given 50 tries for each game round. In other words, the user gets to make 50 pair guesses. In each guess, whether the guess is correct or not, the remaining attempt count decreases. The total number of pairs is 15, thus the condition for winning the game is making 15 correct attempts and obtaining a score of 15. After the winning display, the user is prompted to play again. If all attempts are used and the score is below 15, the game is lost and the user is prompted to play again.

4. Program Structure

4.1 Widget (QWidget)

We first created our application with QApplication, then we supplied a QMainWindow to this app, where there is a connected QWidget that contains all of our game components. In short, the widget is responsible for holding the grid thus the game components.

4.2 Score And Remaining Tries Labels

These labels serve the purpose of displaying the total score a user has made and their remaining attempts at flipping card pairs. They are of the QLabel type and they are set next to next on a horizontal line which is a part of the whole grid the game sits on.

4.3 NewGameButton

The newGameButton is of the type QPushButton, and it simply serves the purpose of starting a new game during another one. Just as the rest of the game, we chose a simple color palette and font to distinguish the button. This button has its place in the same horizontal grid as labels. New game button is responsible for calling the resetGame function in a click event.

4.4 ResetGame Function

This function is responsible for two properties. First is resetting the game's status: the score and the remaining tries. The second responsibility of this function is ensuring the reshuffling of the cards for the new game round, as well as resetting their enable & checked properties along the way.

4.5 Card Object

This object is created at the beginning of each loop and is an instant of QPushButton class, although it is not implemented as a separate class structure, it has an attribute “RealText” due to the setProperty function of QPushButton. Every card's name is set to “?” by default and turns back to this default value if it is revealed but could not be matched. We assign their respective random names by iterating through a random name list called “shuffledNames”.

4.6 Timer

We have not created a timer object because we simply needed a 1000 millisecond delay for an only specified instant where the two different cards are opened. Therefore, we ended up having a code block in a for loop where we call QTimer class’s singleshot function, in order to wait for 1 second to supply the user enough time to realize whether cards match or not. Afterwards we flip the cards back.

4.7 ShuffleNames Function and AnimalNames Lists

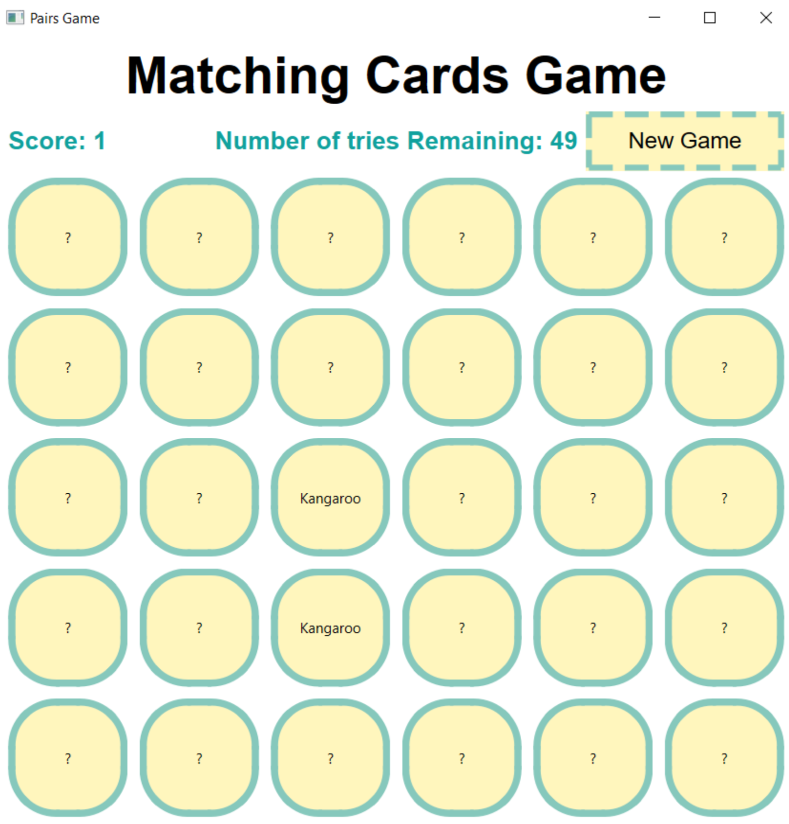
This function and the name list serves the purpose of randomizing card places in each game. At first, we used a list of 30 animal names, which got shuffled at the initialization of the program, then we tried to shuffle it once again when the NewGameButton was clicked. However, we encountered some problems and we were not able to implement our game in this manner. So we decided to switch to a different model: where we had to create a copy of the initial list in the resetFunction to shuffle the animalNames list at each call without changing that list directly. So we can think of AnimalNames as a never-changing real entity, whereas shuffleNames Function has its list in a reflective and constantly randomized version.

4.8 Win/Lose Messages

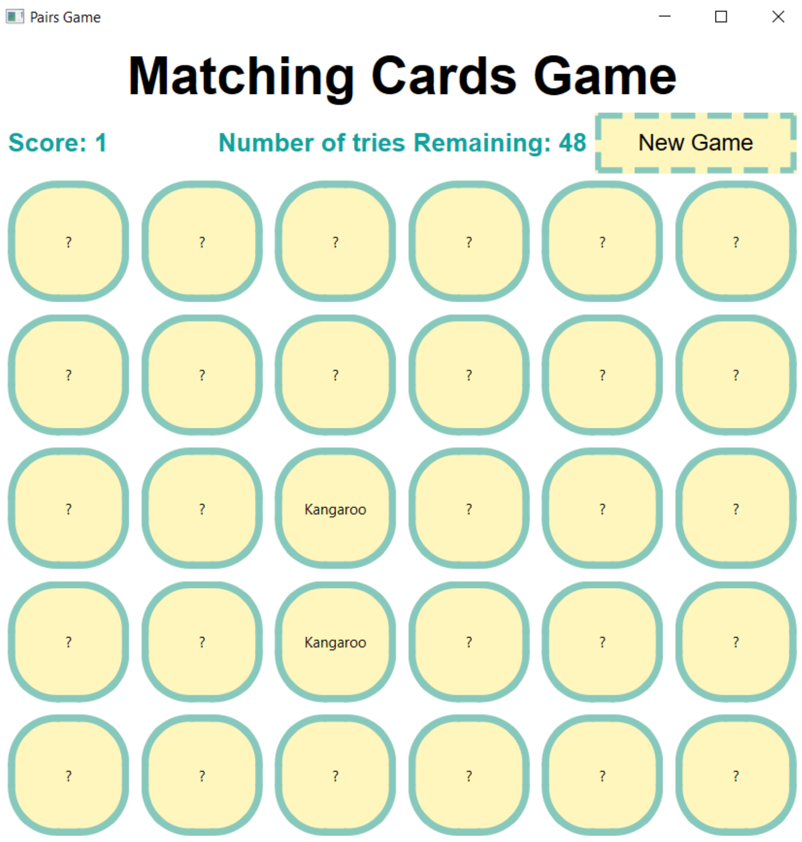
These message boxes pop up when the player has either won or lost the game. A Win Message designed with QMessageBox pops up when the user has made 15 correct guesses, ending up with a score of 15 and all cards matched. The user gets the Lost Message Box if they used all of their attempts at flipping card pairs, when there is no remaining tries left and the score is still below 15. Both of the boxes have a Play Again and OK button. The OK button simply discards the message box, whereas the Play Again button is the same as New Game Button in that it starts a new game by resetting game status and reshuffling.

6. Examples

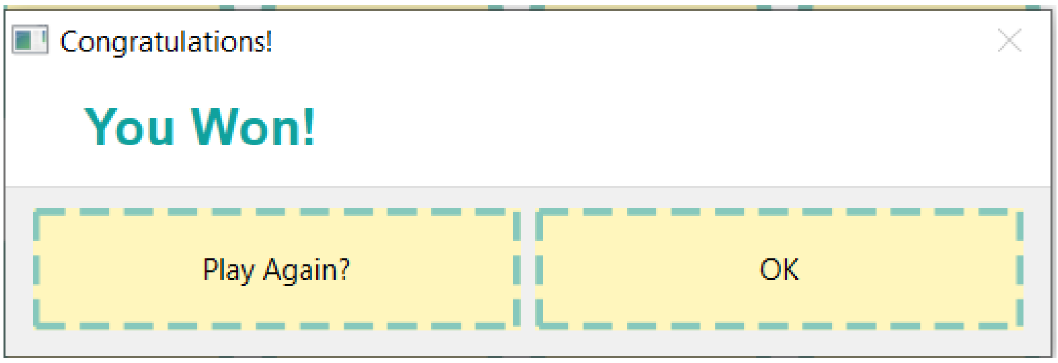
6.1 Game state after a correct guess (First Attempt)



6.2 Game state after a wrong guess (Second Attempt)

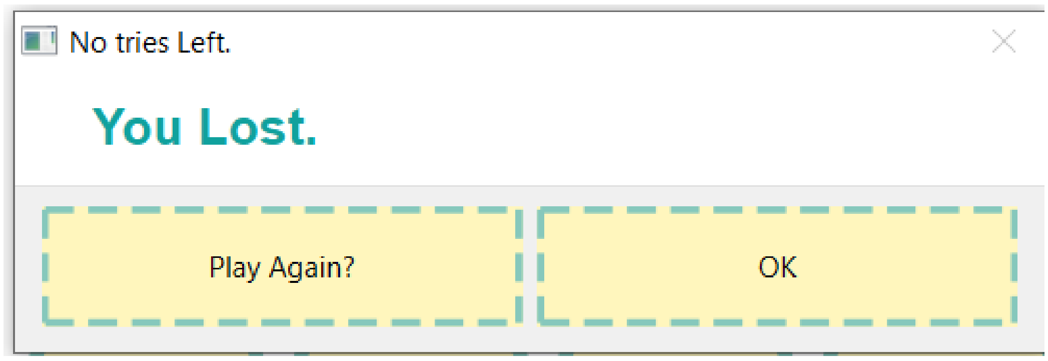


6.2 Game Won Message Box


Congratulations!

You Won!

6.2 Game Lost Message Box


No tries Left.

You Lost.

7. Improvements and Extensions

Although not sure about whether it is really an advantage, we have not created a separate class for the objects (Grid, Button or Timer). Instead we assigned their object attributes in a nested for loop for each card, and connected them to their according signal-slot pairs in the loop. Each card in the grid is a PushButton object - which is initiated at the beginning of each loop. In addition, each card has a “RealText” attribute to store its random name value. When the card is flipped, its text is set to “RealText”, otherwise it is “?”. We believe this loop enabled us to have a compact code for our simple pairs game.

Timer is called whenever two cards are opened, and their names are not the same. To achieve this, we are checking in each loop whether our previously created two dummy cards (firstCard and secondCard) are both not null and compare their names. If the names are the same, then we increase score and decrease timesRemaining by one. Then we set the two dummy card objects to unabled (in other words close or reversed) and their names to default “?”. If the names are different, we create the Timer object to wait for one second before flipping back the cards.

8. Difficulties Encountered

Due to having not coded in C++ before and not having used Qt Framework, both of the partners encountered a learning curve on designing a GUI project with Qt. Although we had a small experience on GUI’s with Javascript, getting accustomed to Qt and C++ usage took a little time due to differences in syntax, event listening or grid layouts.

9. Conclusion

After all, this was a satisfying and enjoyable project for us especially because designing a game has not often been a project subject in our classes. We had the chance to experiment with the C++ language, the Qt framework, Qt tools such as Qt creator, usage of qmake, and so on. Being able to visually perceive the changes we make in the code was a quite practical and convenient experience. Being somehow familiar with GUI concepts such as buttons and events, and the PS lecture that is held on Qt Framework gave us a more intuitive sense of how to code the project step by step and tackling each game component one by one.

10. Appendix

Appendix A

GitHub https://github.com/basak-tepe/Qt-Card-Game

Appendix B

Our source code and qmake file can be found within the same zip file as this documentation.