Systems Programming

CMPE230

Project 3

Match Pairs Game

Aras Taşçı 2020400162

Osman Yasin Baştuğ 2021400021

Submitted to: Can Özturan

Introduction

This project is an implementation of a well known game where the player is expected to match pairs of cards in a given amount of tries. The program has a graphics user interface (GUI) that the player can interact with. The program is written in C++ with the Qt framework.

This documentation aims to provide a description of the program and its implementation details. It might be beneficial for the reader to be familiar with the following words in order to make full use of this documentation:

- Event-driven programming: a programming paradigm in which the flow of the program is determined by events such as user actions. [1]
- Qt: a widely used framework for creating GUIs.
- Slot (Qt): the procedure that gets invoked by the event that's connected to it.
- Signal (Qt): the specific behaviour that triggers slots that are connected to it.

Program Interface

To start the program, the user must first compile the program. As it is a Qt application, the user can run `qmake` at the root of the source code to create a Makefile. Then, running `make` at the root creates an executable `match-pairs` which can be run with the command `./match-pairs` which starts the program.

Program Execution

Gameplay

As explained in the `Introduction` section, the program is a match pairs game. The cards contain words which are initially not known by the player and the player chooses to reveal pairs of cards in order to match them. The player must match them in a given amount of tries.

User Interface

The user interface consists of clickable buttons and labels. There is a "New Game" button that ends the current game and starts a new one upon being clicked. The labels indicate the score and the number of moves left. These three components horizontally claim the first third of the GUI. Below them are the cards that are initially marked with '?'. These cards get 'flipped' upon being clicked, that is, they reveal the noun the card contains. The colors of the cards change based on the state of flipping cards and the success of the pairing: blue for the first flipped card, green/yellow (depending on the colorblind mode) for a successful match and red for an unsuccessful match. Examples with screenshots can be seen in the `Examples` section.

Overview

The program has event-driven programming at the core of its structure. Qt allows the developer to create slots (event handlers) for signals (events) coming from specialized `QObjects`.

In our code, we kept it as simple as possible. The OOP implementation complies with the SOLID principles and the classes are explained in detail in the `Implementation Details` section.

We have also used a design pattern that is very common especially in the game development world: Singleton. A singleton is basically an instance of a class that is globally accessable via its class name. The reason we went for this pattern is that there are no static classes in C++ and we needed to reach the instance of the class from within a function which has no access to it. This pattern was used for an instance of a `Game` class, one can find the implementation of it in the source code.

Implementation Details

Below are the detailed explanations of both of the classes’ functions.

Card

We implemented Card objects which implements from QPushButton which makes them interactable. Card object has a few functions which are used in the Game class. The functions are:

`setName`: Sets the name of the card object. The name is used to compare the cards.

`setColor`: Sets the color of the card object. The color is used to set the background color of the card which helps the player recognize winning or losing situations.

`enable`: Enables the card object. This function is used to enable the card object when the player clicks on it.

`disable`: Disables the card object. This function is used to disable the card object when the player clicks on it. This prevents the player from clicking on the same card twice.

`reveal`: This function is the function where the game's logic is implemented. It disables the pair and make the cards' color green if the cards are the same. If the cards are not the same, it reenables the pair and make the cards' color red. The enablePair and disablePair functions are implemented in Game.cpp which will be explained in the next section.

`justRevealName`: This function is used to reveal the name of the card object. It is used in the reveal function.

`hide`: This function is used to hide the card object. It puts '?' on the card object to let players know that the card is not revealed. It is used in the reveal function.

`remove`: This function is used to remove the card object. It is used in the disable function to delete the name of the cards after successfully matching the pair.

Game

Game is the main class of the program. It is responsible for the game logic and the GUI. In the constructor we create the necessary Qt objects like the timer, score and try label, new game button and the grid layout. We also declare the words that will be used in the game.

The class has a few functions which are:

`initialize`: This function initializes the game with the initial values like score and try times. While looping through the board, it sets random card objects to the random locations after assuring they are empty.

`isPaired`: This function checks if the cards are paired. It is used in the reveal function of the Card class.

`disablePair`: This function disables the pair of cards by blocking all signals and setting their color green for 1 second. It then disconnects the Card objects to assure they are not interactable again in that round. It is used in the reveal function of the Card class if the cards in the pair are the same.

`reenablePair`: This function reenables the pair of cards by blocking all signals and setting their color red for 1 second. The cards' color turns to blue again which means they are interactable again. It is used in the reveal function of the Card class if the cards in the pair are not the same.

`placeCard`: This function is used to place the card in an array of two named currentPair which stores the pair. This array is used in the reveal function of the Card class to be able to reach both cards' objects.

`restart`: This function is used to restart the game. It is connected to the new game button. It resets the score and try times and calls the initialize function to start a new game.

`timeToEnable`: This function is a slot function that is called when the timer times out. It is responsible for enabling the cards that were previously disabled after a failed attempt to match two cards. If the two cards were successfully matched, it disables them and updates the score. If all cards have been matched, it calls the `win` function. If the player has used up all their tries, it calls the `lose` function.

`win`: This function is called when the player has successfully matched all the cards in the game. It disconnects all the signals from the cards, reveals all the cards, and displays a message box with the text "You won!". It sets a 'cancel' button. If the player clicks on the 'cancel' button, the program exits.

`lose`: This function which is called when the player has used up all their tries and has lost the game. It disconnects all signals from the cards, reveals all the cards, and displays a message box with the text "You lost...". It sets a 'cancel' button. If the player clicks on the 'cancel' button, the program exits.

`revealAllCards`: This is a function that reveals all the cards in the game. It takes a boolean
parameter `isAWin` which determines the color of the revealed cards. If `isAWin` is true, the cards are revealed in green color, otherwise they are revealed in red color.

`blockAllSignals`: This function takes a boolean flag as input. It loops through all the cards in the grid and calls the `blockSignals` function on each card, passing in the flag as an argument. If the flag is true, this will block all signals emitted by the card, effectively disabling it. If the flag is false, this will unblock all signals emitted by the card, effectively enabling it. This function is used to disable or enable all the cards in the grid at once, depending on the state of the game.

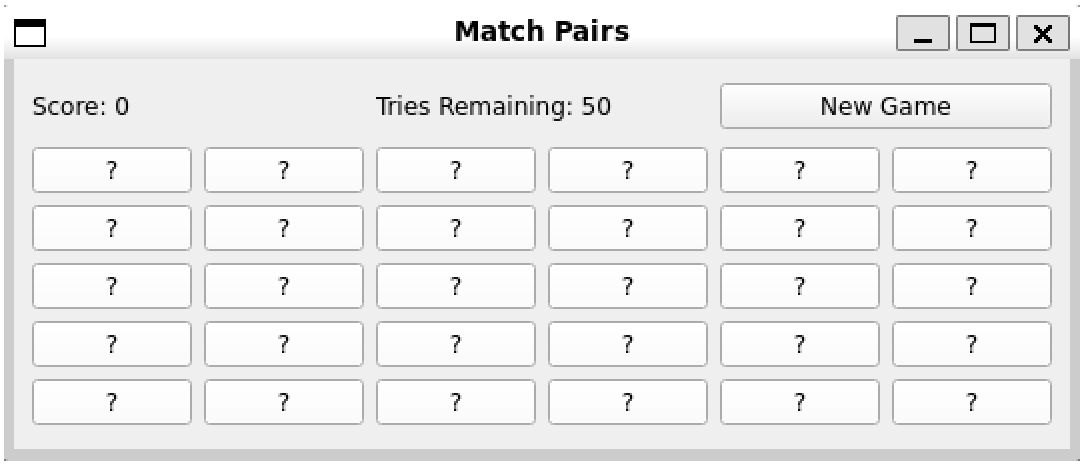
`disconnectAll`: It disconnects all signals from the cards in the grid by iterating through all the cards in the grid and calling the `disconnect` function on each card, passing in the `clicked` signal and the `reveal` slot as arguments. This ensures that no further signals will be emitted from the cards and that their slots will not be called.

main()

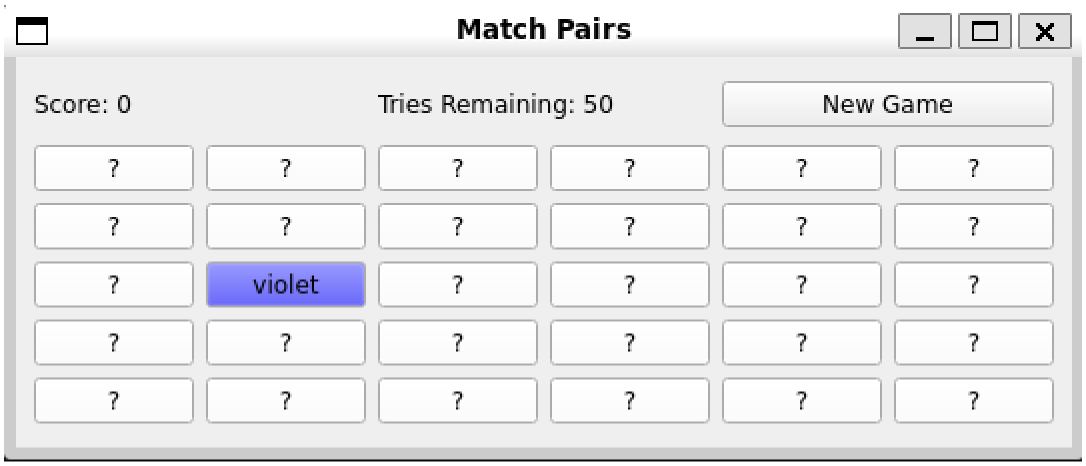
The `main` function is the entry point of the program. It creates a `QApplication` object, a `QMainWindow` object, and a `QWidget` object. It sets the title of the main window to `MatchPairs`. It creates a `Game` object and initializes it. It creates a `QVBoxLayout` object and a `QHBoxLayout` object, and adds the score label, try label, and new game button to the horizontal layout. It adds the horizontal layout and the grid layout of the `Game` object to the vertical layout. It sets the central widget of the main window to the vertical layout, and shows the main window. Finally, it starts the event loop by calling `a.exec()`.

Examples

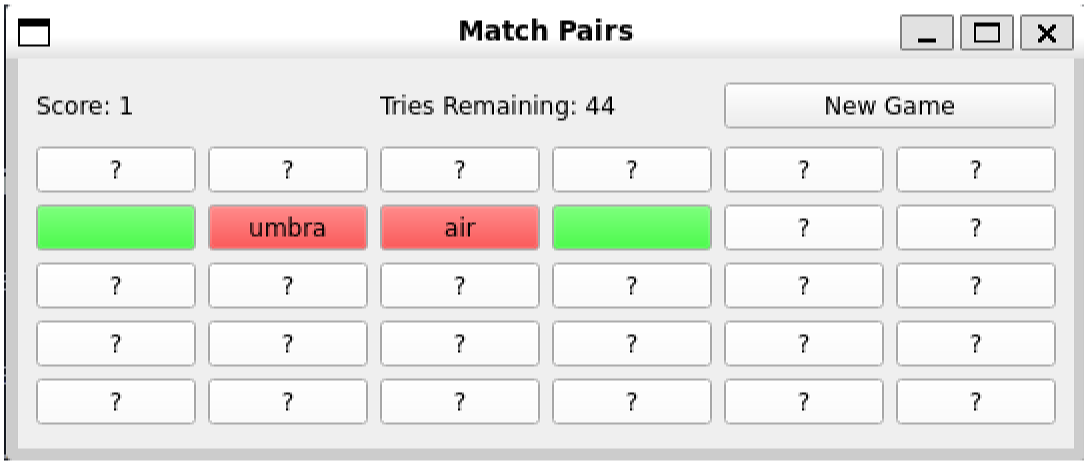
- This is how it looks when you first boot the game:



- This is how it looks when you select a single card:



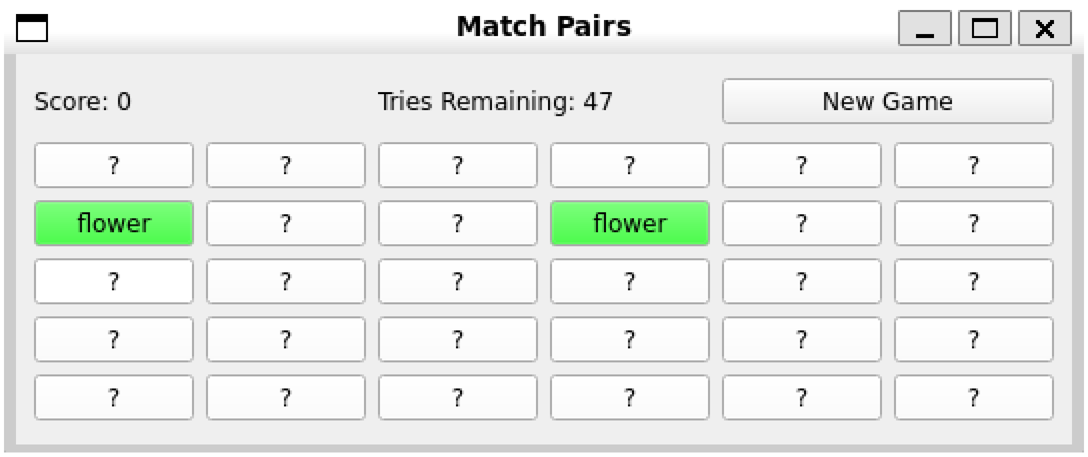
- This is how it looks when you cannot remember where the other card is:



Score: 1 Tries Remaining: 44

?	?	?	?	?	?
🟩	umbra	air	🟩	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?

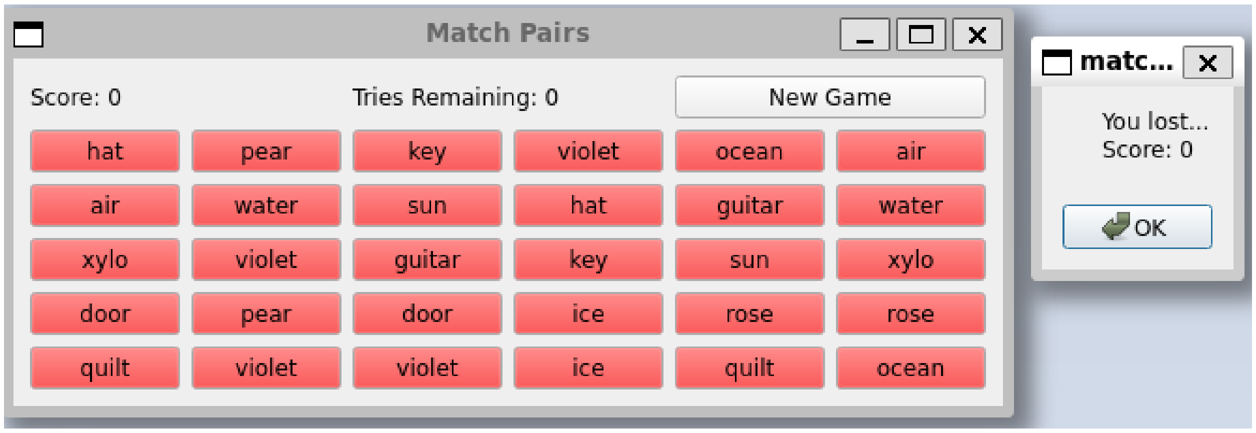
- This is how it looks when you finally remember it after failing the game (miserable):



Score: 0 Tries Remaining: 47

?	?	?	?	?	?
flower	?	?	flower	?	?
?	?	?	?	?	?
?	?	?	?	?	?
?	?	?	?	?	?

- This is how it feels to be a disappointment:

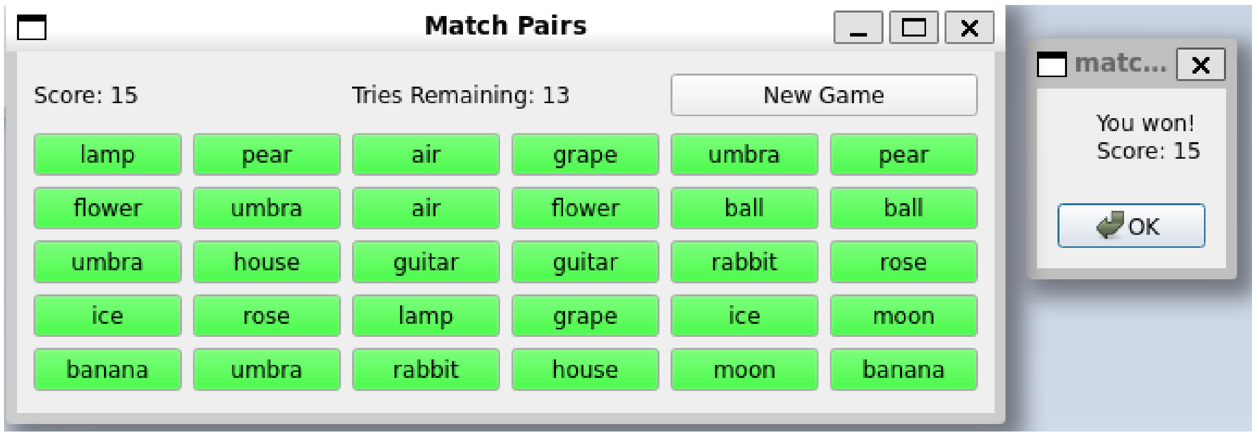


Score: 0 Tries Remaining: 0

hat	pear	key	violet	ocean	air
air	water	sun	hat	guitar	water
xylo	violet	guitar	key	sun	xylo
door	pear	door	ice	rose	rose
quilt	violet	violet	ice	quilt	ocean

You lost...
Score: 0

- This is how it feels to chew 5 gum:



lamp	pear	air	grape	umbra	pear
flower	umbra	air	flower	ball	ball
umbra	house	guitar	guitar	rabbit	rose
ice	rose	lamp	grape	ice	moon
banana	umbra	rabbit	house	moon	banana

You won!
Score: 15

Improvements

One improvement that can be made in the GUI of the program is having a colorblind mode. When we made one of our friends who is colorblind, she couldn't distinguish the difference between the failed matches and the previously succeeded ones, which are fine as she can still read the nouns but it still took away the joy of the gameplay. We actually created a version which has a colorblind button that turns the success color from green to yellow upon being clicked but have decided not to publish it, fearing stability issues.

Conclusion

Event-driven programming paradigm that the Qt framework adopts is not only mandatory for GUI programming but also generally a good practice of writing code. Event-driven programming behaviours like events and event handlers present the coder the gift to write clean and scalable code, and we have surely taken benefit from it.

Source

- [1] https://en.wikipedia.org/wiki/Event-driven_programming