# CMPE 230

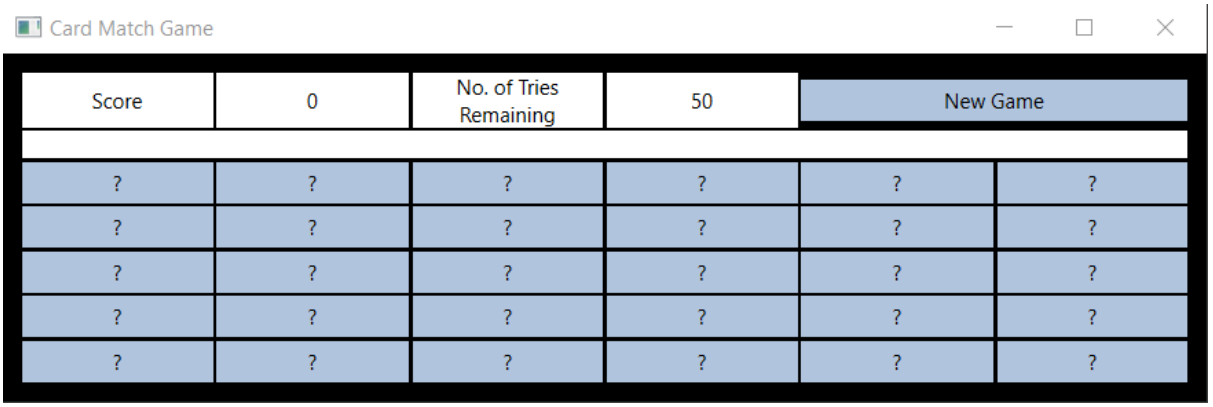
**Students:**
Hanaa Zaqout 2020400381
Deniz Bilge Akkoç 2020400135

**Project:**
Card Match Game
Implemented in Qt

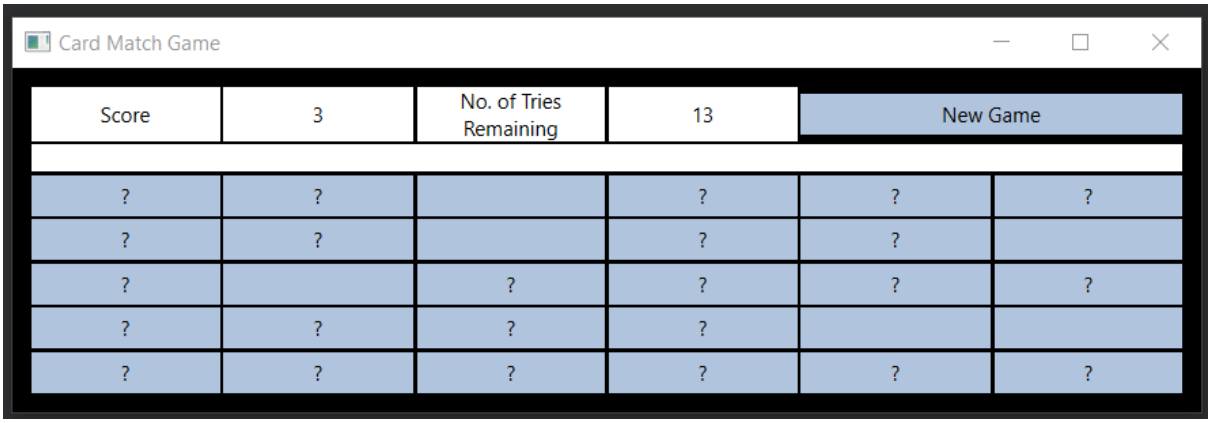
**Submission date:**
22nd. May 2023

**Purpose:**

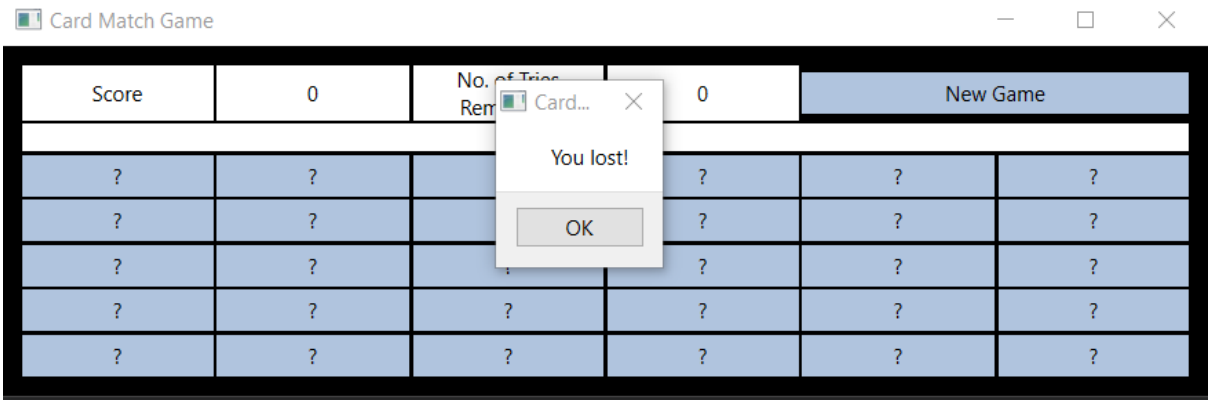
In this project, we created an interactive GUI for the famous card match, pairs game. In our game, the Score field displays the number of matchings for the player. At the beginning, the user will be given 50 tries. No. of Tries Remaining will display the remaining number of tries. The New Game button restarts the game. Below, you can see screenshots of our game in different cases.




(The initial screen in the game: score=0, no. of tries remaining=50, and all cards are closed)



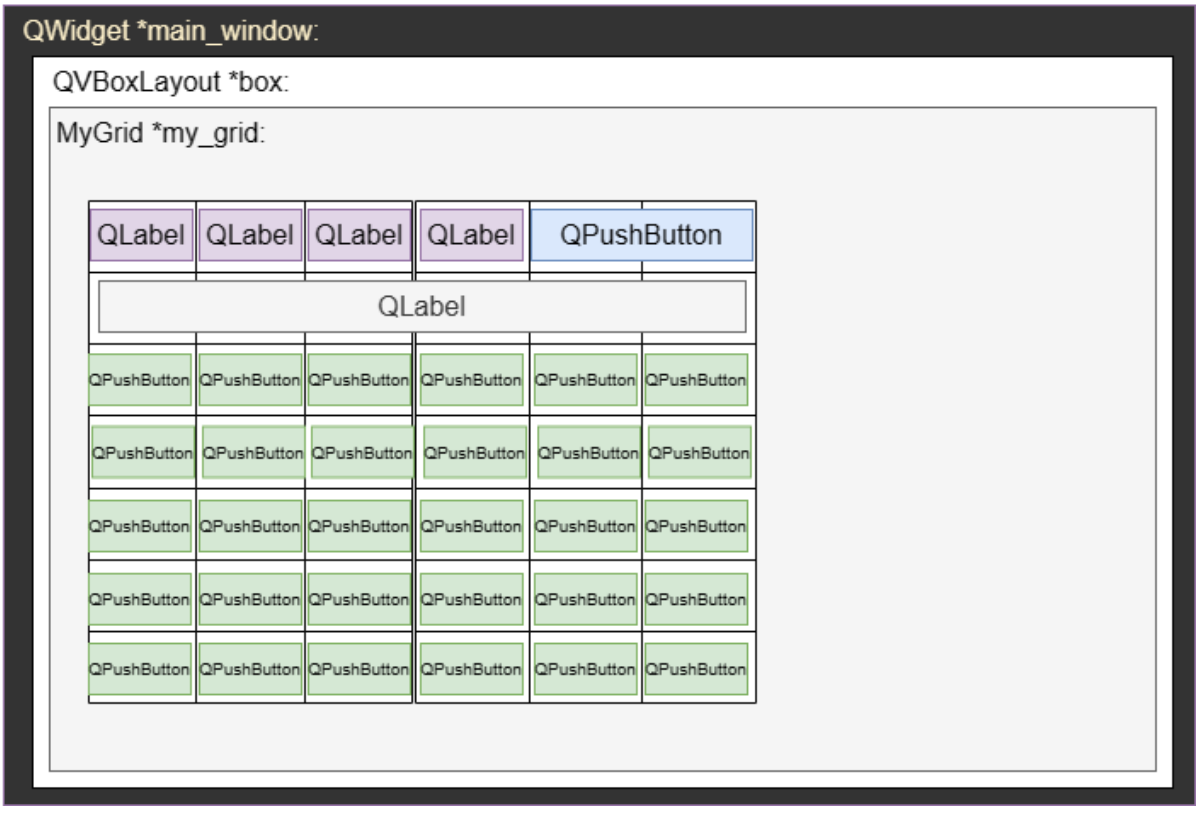

(while playing the game: here score=3 and there are 6 empty cards in the grid, 3 matched pairs. Also, no. of tries remaining =13 which means that the player has opened 50-13=37 pairs till now)




(since no. of tries remaining=0 the player has lost the game; pop up: You lost! Click OK button will close the pop up, then, the only thing that is clickable in the game is the new game button that will restart the game. )

**GUI:**

We have used Qt classes to reach the desired GUI of the game. We used QWidget, QVBoxLayout, QGridLayout, QPushButton, QLabel, and QPalette classes. Using the fact that Child widgets in a QWidget automatically appear inside the parent widget, we were able to arrange the objects on the screen. You can notice that by the difference in colors in the game; black, the background color, was used for the parent QWidget, but blue for example was used for the child QPushButton. Moreover, QGridLayout was important to arrange objects according to the column and row indices. Here is an image showing the layout we used for the screen GUI.



**Implementation:**

We used header files to declare classes, functions, and variables, but we defined classes' functions and variables in .cpp files.

- **MyCard Class**

  MyCard class inherits QPushButton. Also, the constructor of this class expects QString as a parameter when you create a new object, this text will be displayed on the button.
  - **Attributes:**
    - **int flipped**
    - **int out**

  Furthermore, as the MyCard class includes the Q_OBJECT macro, we could use objectName() and setObjectName() methods to store the animal name for each button.

- **MyGridClass**
  - **Public slots:**
    - **Void create_grid()**

      Creates QLabels that are in the first row on the screen. Moreover, it creates the 30 buttons and gives them random naming by calling **random_name_generator()**. Also,it creates a New Game button and defines the clicked slot function inline using lambda. This lambda function reset all variables in the game to their initial values (e.g. score=0, remaining tries=50, all cards are closed, generate new random naming for the cards, ..). All these objects are added to the gridlayout by addWidget method.
  - **Private Attributes:**
    - **Int available[15] and QStringList animals**
    - **Mycard *arr[30]**
    - **Mycard *flippedC**
    - **int score**
    - **int remaining**
    - **int stop**
    - **QLabel *scoreL**
    - **QLabel *triesL**
    - **QPalette smaple_palette**
  - **Private slots:**
    - **QString random_name_generator()**

      Initially, available[15] includes 2,2,2,..,2. By calling QRandomGenerator, a random number bounded by (0,15) is created. Using this random number as an index, check if available[index]>0, we choose the name from the animals list[ same index] (don't forget to decrement the available count at this index). If available[index]=0, look for a new random number.
    - **void flip_card(Mycard *card)**

      Firstly, it checks if the clicked card is already flipped (flipped attribute from Mycard) or if the clicked card has been already matched (out attribute from Mycard), or if there are now already two open cards

(stop attribute from grid), in any of these cases it ignores the signal and returns.

Otherwise, check if it is the first card to be opened in the pair, if so, show the card name on the screen. if it is the second card, wait for 1 second using QTimer (needed so the player can see the animal name on the 2nd card). While waiting input signal will be ignored (thanks to the first if-else). Now time to decide, if the open pair are matched, increment the score, decrement the number of remaining tries anyway. Also, check for a winning or losing case.

- **main**
  Run the Qapplication

I want to point out that MyCard and MyGrid objects point to QWidget as their parent, which is necessary to be able to show these objects on the screen.

**Challenges:**

When opening the second card of the pair, we needed to find a way to freeze or stop the game so the player could see the name of the animal shown on the card. We tried QThread, but it was not a good idea, because the whole game froze and even it froze before showing the name on the button (.show() was not called yet). We found a solution using QTimer that counts time and we created a stop variable, if stop=1 we ignore the input signal (just like an artificial freeze).

**To Improve:**

For larger games, we could give more attention to oop principles. For example, for MyCard class, we could make the access specifier of the attributes private and use getters and setters to achieve encapsulation (but it was not that needed since we had only 2 classes). Also, we could improve the graphical interface of the game, using more suitable colors, adding animals' pictures rather than words. Moreover, we could add a timer to make the game more challenging.

**Conclusion:**

It was a funny project, where we learned how to use Qt and applied oop in c++. It is our first GUI project, and we learned concepts and thought about stuff that we would not usually care about while running code through the terminal, e.g. the synchronous issues, delays, signals, slots, etc…