Project 3 – Card Match Game

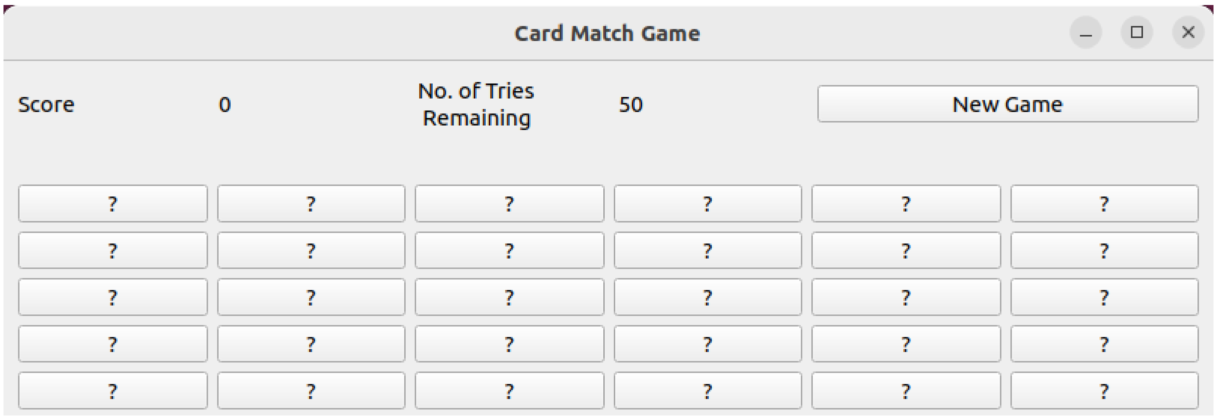
CmpE 230, Systems Programming, Spring 2023

Semih Yılmaz 2020400171

Submission Date: 23/05/2023

1. Introduction

In this project, I implemented a game known as "Card Match" or "Pairs" using QT. The objective of the game is to turn over pairs of matching cards.



The button widgets are:
- New game: starts a new game
- The 6 x 5 array of cards which can be clicked to turn over.

The **Score** field will display 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.

2. How to Use the Program

First of all, installing qt is required. To install qt on ubuntu, following terminal commands can be used:

```
sudo apt-get install build-essential
sudo apt install -y qtcreator qtbase5-dev qt5-qmake cmake
```

To create Makefile, following command can be used:

```
qmake pairsGame.pro
```

Finally, to create executable, just use make command and executable will be created. After that executable can be run:

```
make
./pairsGame
```

3. Program Structure

The implementation of the program consists of 3 source files and 2 header files.

Source Files: main.cpp, cardbutton.cpp, manager.cpp

Header Files: cardbutton.h, manager.h

a) main.cpp

```
#include <QApplication>
#include "manager.h"

int main(int argc, char *argv[]){
     QApplication app(argc, argv);
     QWidget *window = new QWidget;
     window->setWindowTitle("Card Match Game");
     Manager* manager = new Manager();
     window->setLayout(manager);
     window->show();
     return app.exec();
}
```

My main function is located here. It basically initializes and configures necessary objects. The manager class is a subclass of QLayout class with custom fields and methods. It handles most of the work required for the program.

b) cardbutton.h

```
#ifndef CARDBUTTON_H
#define CARDBUTTON_H
#include <QPushButton>
class Manager;

class CardButton : public QPushButton{
    Q_OBJECT
private:
    Manager* manager;
    QString word;
    bool found;
    bool hidden;
public:
    CardButton(const QString& text, Manager* x_manager);
    bool isHidden();
    bool isFound();
    QString getWord();
    void hideCard();
    void showCard();
    void cardFound();
public slots:
    void selected();
};
#endif // CARDBUTTON_H
```

In this header file, I defined a custom subclass of QPushButton class. It has some private attributes and corresponding public getter and setter type methods. Moreover it has a custom slot function. Their implementation was done in cardbutton.cpp.

c) cardbutton.cpp

```
#include "cardbutton.h"
#include "manager.h"

CardButton::CardButton(const QString& text, Manager* x_manager):
    QPushButton(text, 0)  {
    manager = x_manager;
    this->word = text;
    hideCard();
    found = false;
    QObject::connect(this, SIGNAL(clicked()),this, SLOT(selected()));
   }

void CardButton::hideCard(){
    hidden = true;
    this->setText("?");     }

void CardButton::showCard(){
    hidden = false;
    this->setText(this->word);   }

void CardButton::cardFound(){
    found = true;
    this->hide();
    this->setText("");      }

// slot
void CardButton::selected(){
    manager->selected(this);  }

bool CardButton::isHidden(){   return hidden;  }
bool CardButton::isFound(){    return found;   }
QString CardButton::getWord(){ return word;    }
```

This source file contains implementation of the methods of the CardButton class. hideCard and showCard methods change the status of “hidden” attribute and shows/hides the word on the button. cardFound method hides the button in GUI. The slot function “selected” is bound to clicked signal of the button in the constructor. Thus, it passes the button object as parameter to the “selected” method of the corresponding manager object when the button is clicked.

d) manager.h

```
#ifndef MANAGER_H
#define MANAGER_H
#include <QVBoxLayout>
#include <QGridLayout>
#include <QLabel>
#include <algorithm>
#include <random>
#include <QTime>
#include <QMessageBox>
#include <QEventLoop>
```

```
#include <QCoreApplication>
#include "cardbutton.h"

class Manager: public QVBoxLayout{
    Q_OBJECT
private:
    int score, nofTries;
    bool firstCardSelected, busy;
    CardButton *firstCard;
    QLabel *labelScore, *labelNofTries;
    QGridLayout *buttons;
    void showMessageBox(QString);
public:
    Manager();
    void selected(CardButton*);
private slots:
            void startNewGame();
};

#endif // MANAGER_H
```

Here a subclass of QVBoxLayout is defined. This custom class has necessary attributes and methods for the game, and it manages what happens when buttons are clicked.

e) manager.cpp

i. Constructor

```
    #include "manager.h"

Manager::Manager(){
    QGridLayout *row1 = new QGridLayout;
    QLabel *label1 = new QLabel("Score");
    labelScore = new QLabel();
    QLabel *label3 = new QLabel("No. of Tries\n Remaining");
    labelNofTries = new QLabel();
    QPushButton *newGameButton = new QPushButton("New Game");
    QObject::connect(newGameButton, SIGNAL(clicked()), this,
SLOT(startNewGame()));

    row1->addWidget(label1,0,0,1,1);
    row1->addWidget(labelScore,0,1,1,1);
    row1->addWidget(label3,0,2,1,1);
    row1->addWidget(labelNofTries,0,3,1,1);
    row1->addWidget(newGameButton,0,4,1,2);

    this->addLayout(row1);
    QSpacerItem *si = new QSpacerItem(0, 30);
    this->addSpacerItem(si);
    buttons = new QGridLayout();
    startNewGame();
    this->addLayout(buttons);
}
```

The constructor defines required labels and buttons, and adds them to the layout. It calls startsNewGame method to handle card buttons.

ii. void Manager::startNewGame()

```
    void Manager::startNewGame(){
    firstCardSelected = false;
    busy = false;
    score = 0;
    nofTries = 50;
    labelScore->setText(QString::number(score));
    labelNofTries->setText(QString::number(nofTries));

    QString arr[] = {"lion", "lion", "tiger", "tiger", "elephant",
"elephant",
                     "giraffe", "giraffe", "zebra", "zebra", "monkey",
"monkey",
                     "bear", "bear", "kangaroo", "kangaroo",
"penguin", "penguin",
                     "snake", "snake", "hippo", "hippo", "crocodile",
"crocodile",
                     "koala", "koala", "panda", "panda", "wolf",
"wolf"};

    // Shuffling the array
    int size = sizeof(arr) / sizeof(arr[0]);
    std::random_device rd;
    std::mt19937 gen(rd());
    std::shuffle(arr, arr + size, gen);

    // first initilazition of buttons
    if(buttons->count() == 0){
        for(int row=0; row<5; row++){
            for(int col=0; col<6; col++){
                CardButton *randButton = new
CardButton(arr[row*6+col],this);
                buttons->addWidget(randButton, row, col, 1, 1);
            }
        }
    }
    // replacing of buttons (when new game button is pressed)
    else{
        for(int i = 0; i < buttons->count(); ++i){
            QWidget *widget = qobject_cast<QWidget*>(buttons-
>itemAt(i)->widget());
            CardButton *randButton = new CardButton(arr[i],this);
            buttons->replaceWidget(widget, randButton);
            widget->hide();
        }
    }
}
```

This method sets some attiributes to what they should be at the start of a new game. Then, it shuffles the array which holds words of cards. Then it checks whether it is the first time to define buttons or not. If it is the first time, then card buttons are defined and set a position in buttons attiribute which is a QGridLayout. If not, then new buttons are again defined and replaced with old button using replaceWidget method of QLayout class.

iii. void Manager::selected(CardButton* cardButton)

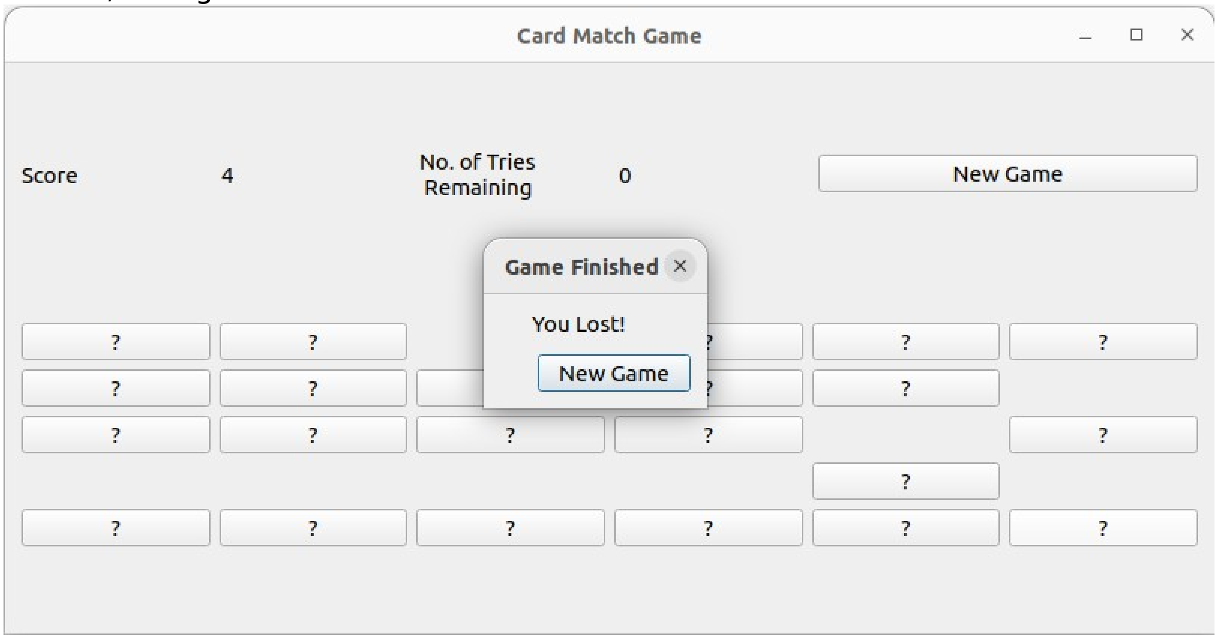
```
void Manager::selected(CardButton* cardButton){
    if((!cardButton->isFound())&&(cardButton->isHidden())&&!busy){
        if(!firstCardSelected){
            cardButton->showCard();
            firstCardSelected = true;
            firstCard = cardButton;
        }
        else{
            firstCardSelected = false;
            cardButton->showCard();
            busy = true;
            // 1 second delay without freezing screen
            QTime dieTime= QTime::currentTime().addSecs(1);
            while (QTime::currentTime() < dieTime)
                QCoreApplication::processEvents(QEventLoop::AllEvents,
100);
            // pairs are matched
            if(!cardButton->getWord().compare(firstCard->getWord())){
                score++;
                labelScore->setText(QString::number(score));
                cardButton->cardFound();
                buttons->replaceWidget(cardButton, new QLabel());
                firstCard->cardFound();
                buttons->replaceWidget(firstCard, new QLabel());
                if(score >= 15){ // won
                    showMessageBox("You Won!");
                    return;
                }
            }
            // not matched
            else{
                nofTries--;
                labelNofTries->setText(QString::number(nofTries));
                cardButton->hideCard();
                firstCard->hideCard();
                if(nofTries <= 0){  // lost
                    showMessageBox("You Lost!");
                    return;
                }
            }
            busy = false;
        }
    }
}
```

This method is called when a card button is clicked. Firstly, it checks whether the clicked button is a hidden one and manager is not busy. (Manager is busy while showing clicked pairs for a second) Secondly, it checks if the card clicked is the first or second card of a pair. In case it is the first card, the word of the card is shown. When the second card is clicked, second card is also shown for a second. Ultimately, it compares the words of the cards. If they are matched, score will be increased by one and card buttons are replaced with empty labels. (In order to keep the form of the grid layout which includes buttons as the form at the initial state) If cards are not matched, then number of tries remaining will be decreased by one and the words of the cards will be hidden. Moreover, this method controls if the score is reached to 15 and the number of tries remaining is reached to 0, and then calls showMessageBox method with a suitable message.

iv. **void Manager::showMessageBox(QString message)**

```
void Manager::showMessageBox(QString message){
    QMessageBox msgBox;
    msgBox.setWindowTitle("Game Finished");
    msgBox.setText(message);
    QPushButton button;
    button.setText("New Game");
    msgBox.addButton(&button,QMessageBox::RejectRole);
    QObject::connect(&button, SIGNAL(clicked()), this,
SLOT(startNewGame()));
    msgBox.exec();
}
```

This method shows a message box with a button. When the button is clicked, new game will be started.



5. Difficulties Encountered

- I wanted to add a delay for a second after the user clicks the second card of a pair. In case I used standard delay libraries, the screen had frozen and the word of the second card had never shown. To overcome the problem, I found a way to allow qt to handling its job while the program sleeps. Here is the solution:

  ```
  QTime dieTime= QTime::currentTime().addSecs(1);
  while (QTime::currentTime() < dieTime)
  QCoreApplication::processEvents(QEventLoop::AllEvents, 100);
  ```

- I used header files for the first time. Two headers had included each other, and it caused a compilation error. After research about the issue, I learned the name of the problem and how to deal with it. It was circular dependency problem, and I was supposed to use forward declarations instead of including header files. In that case, necessary headers must be included in the corresponding source files for the complete definitions.

- I tried to take parameters for a slot function. However, in qt, it is not possible to pass arguments which are not arguments in the corresponding signal function. Thus, I made a dummy slot function in the object which has corresponding signal function and called our real slot function with parameters.

  My slot function: void Manager::selected(CardButton* cardButton);

  <u>What I tried:</u>
  In manager.cpp:
  ```
  QObject::connect(randButton, SIGNAL(clicked()), this,
  SLOT(selected(randButton)));
  ```

  <u>How I solved:</u>
  In manager.cpp:
  ```
  QObject::connect(randButton, SIGNAL(clicked()), randButton,
  SLOT(selected()));
  ```

  In CardButton.cpp:
  ```
  void CardButton::selected()  {    manager->selected(this);  }
  ```