Kevin Riggs Module 3 Assignment Write Up

Requirements:

Design and implement a TicTacToe game in C++. Unless you want to learn MFC or some other GUI on your own, the game is a Console application and redraws the board at the beginning and again after each move.
The board is a 3x3 grid of horizontal and vertical lines
The user is the person playing TicTacToe and the game automatically determines and plays moves. The user always plays first as "X" and selects the location by the row and col and the game plays by placing an "O" in a cell. There is no requirement for the game to have any "smarts" as to where to place the "O", although you might have it loop looking for any move that would "win" or otherwise select a random unoccupied cell.
For example,
1 3
Would place the users "X" in the first row, third column
The game determines the next move and places it's "O" at the location determined
The game evaluates if either player has won at the completion of each move
If either player has won, the winner is displayed and the game ends
The game determines if all squares are filled at the completion of each move
If all squares are filled

A "Tie" is displayed, and the game ends

Demo:

I only checked these with Google Chrome and Firefox.
http://oopcpp.rpkdesign.tech/TicTac/TicTac.html
http://oopcpp.rpkdesign.tech/statistics/statistics.html
(I didn't put a link the statistics app in the last write up. The statistics app can be passed a text file, or csv, of doubles separated by newlines, not commas, only splitting on newlines. See ~/statistics/dist.txt for an example file)

Design:

Notice, I apologize for the bad design. This was intended to just be a spike but I did not have time this week to reimplement the code properly.

I built off a Qt Designer plugin example which are designed to be built, saved in a folder Qt creator can access, and are linked to for building GUIs. They are supposed to be reused.

Interestingly, I did find a glaring fatal error in the original example implementation. Essentially the original example implementation set the turn number to 9 without having initialized the string that saves the state of the game. It then tried to iterate through positions 0 to 8 on that empty string. This only failed when running a debug build.

Big flaw current implementation. The paint event handler is taking responsibility for determining whether someone has won the game. All it should do is paint the board given information from the state of the game. This causes quirks when playing the game.

For testing purposes, I set the computer player to mark the next available square, not a random move or intelligent move. If I set it to random, I would not be able to easily reproduce results when testing manually. If I had an intelligent design it would have been more reproducible but I did not have time to implement it.

Test Design and Output:

I did purely exploratory manual testing.
Results –

QMessageBox does not work correctly targeting web assembly. We get a notification that the game is over and we can click out of it, but the text does not display who won/tie. Works as intended when targeting traditional platforms like windows and linux.

Due to the coupling of state with painting, we can reproduce scenarios where visually we get two winners but the message box displays who one first. This is a consequence of not de-spiking after the proof of concept.

Lessons learned:

Not so much lessons learned but obvious ways to reimplement this program.
First, I should fully de-spike and include a testing framework.
Next, implementation-wise I just need to take the winner check logic out of the paint function. I can stick that in the mouse click event handler.
Last, I want a proper message to display when there is a winner. Clearly QMessageBox doesn't work with WASM, so I should just add a header with a lineedit that updates given the state of the game(x's turn, o's turn, x wins, tie etc). This will work with WASM.

Source Code:

https://github.com/riggskevinp/oopcpp/tree/master/TicTac

```
/****************************************************************************
**
** Copyright (C) 2016 The Qt Company Ltd.
** Contact: https://www.qt.io/licensing/
**
** This file is part of the examples of the Qt Toolkit.
**
** $QT_BEGIN_LICENSE:BSD$
** Commercial License Usage
** Licensees holding valid commercial Qt licenses may use this file in
** accordance with the commercial license agreement provided with the
```

```
#ifndef TICTACTOE_H
#define TICTACTOE_H

#include <QWidget>

QT_BEGIN_NAMESPACE
class QRect;
class QSize;
QT_END_NAMESPACE
```

```
//! [0]
// Holds the state and visualisation
// for a tictactoe game
class TicTacToe : public QWidget
{
   Q_OBJECT
   Q_PROPERTY(QString state READ state WRITE setState)

public:
        // Constructor for the object
   explicit TicTacToe(QWidget *parent = nullptr);
        // Used for painting,
        // window will want to know what to set
        // the size to.
   QSize minimumSizeHint() const override;
   QSize sizeHint() const override;
        // update the state of the game
   void setState(const QString &newState);
        // return the state of the game
   QString state() const;
        // reset the game
   void clearBoard();

protected:
        // handle when a click is received on the widget
   void mousePressEvent(QMouseEvent *event) override;
        // how to paint the board
   void paintEvent(QPaintEvent *event) override;

private:
        // char used to denote the state of the game
   enum : char { Empty = '-', Cross = 'X', Nought = 'O' };

        // Used to calculate how big to make visuals on the
        // board.
   QRect cellRect(int row, int col) const;
   int cellWidth() const { return width() / 3; }
   int cellHeight() const { return height() / 3; }

        // data that holds the state of the game.
   QString myState;
   int turnNumber = 0;
        int winnerPosition = -1;
};
//! [0]

#endif
```

```
**
****************************************************************************/

#include "tictactoe.h"

#include <QMouseEvent>
#include <QPainter>
#include <QMessageBox>

// When the game has just started, no Xs or Os
static inline QString defaultState() { return QStringLiteral("---------"); }

// Constructor, parent widget will give responsibility to the main window.
TicTacToe::TicTacToe(QWidget *parent)
   : QWidget(parent)
{
   turnNumber = 0;
          myState = defaultState();
}

// size hints for initial size
QSize TicTacToe::minimumSizeHint() const
{
   return QSize(200, 200);
}

// size hints for initial size
QSize TicTacToe::sizeHint() const
{
          return QSize(200, 200);
}

// Given a new state, update the current state accordingly
void TicTacToe::setState(const QString &newState)
{
   turnNumber = 0;
   myState = defaultState();
   int position = 0;
   while (position < 9 && position < newState.length()) {
      QChar mark = newState.at(position);
      if (mark == Cross || mark == Nought) {
         ++turnNumber;
         myState.replace(position, 1, mark);
      }
      position++;
   }
   update();
}
```

```
// Return the current state
QString TicTacToe::state() const
{
   return myState;
}

// Reset the game
void TicTacToe::clearBoard()
{
   myState = defaultState();
   turnNumber = 0;
   update();
}

// Handle mouse clicks
// Logic includes checking for a winner in order to display a message
// Displaying a QMessageBox doesn't work as intended targeting WASM
void TicTacToe::mousePressEvent(QMouseEvent *event)
{
   if (turnNumber == 9) {
                        if(winnerPosition >= 0){
         QString winner = myState.at(winnerPosition);
         clearBoard();
         update();
         QMessageBox::information(this->parentWidget(), tr("Game Over"), tr("%1 wins").arg(winner),
QMessageBox::Ok | QMessageBox::Close);
                        } else{
         clearBoard();
         update();
         QMessageBox::information(this->parentWidget(), tr("Game Over"), tr("Result: tie"),
QMessageBox::Ok | QMessageBox::Close);
                        }

   } else {
      for (int position = 0; position < 9; ++position) {
         QRect cell = cellRect(position / 3, position % 3);
         if (cell.contains(event->pos())) {
            if (myState.at(position) == Empty) {
                                                // modify to just be cross when computer plays Nought
               if (turnNumber % 2 == 0)
                  myState.replace(position, 1, Cross);
               else
                  myState.replace(position, 1, Nought);
               ++turnNumber;
               update();
            }
         }
```

```
    }
                for (int position = 0; position < 9; ++position) {
                        if(myState.at(position) == Empty){
                                myState.replace(position, 1, Nought);
                                ++turnNumber;
                                update();
                                break;
                        }
    }
  }
}

// Logic for painting the board
// Notice there is too much responsibility in this class
// It currently decides who won.
void TicTacToe::paintEvent(QPaintEvent * /* event */)
{
   QPainter painter(this);
   painter.setRenderHint(QPainter::Antialiasing);

   painter.setPen(QPen(Qt::darkGreen, 1));
   painter.drawLine(cellWidth(), 0, cellWidth(), height());
   painter.drawLine(2 * cellWidth(), 0, 2 * cellWidth(), height());
   painter.drawLine(0, cellHeight(), width(), cellHeight());
   painter.drawLine(0, 2 * cellHeight(), width(), 2 * cellHeight());

   painter.setPen(QPen(Qt::darkBlue, 2));

   for (int position = 0; position < 9; ++position) {
      QRect cell = cellRect(position / 3, position % 3);

      if (myState.at(position) == Cross) {
         painter.drawLine(cell.topLeft(), cell.bottomRight());
         painter.drawLine(cell.topRight(), cell.bottomLeft());
      } else if (myState.at(position) == Nought) {
         painter.drawEllipse(cell);
      }
   }

   painter.setPen(QPen(Qt::yellow, 3));

   for (int position = 0; position < 9; position = position + 3) {
      if ((myState.at(position) == Cross || myState.at(position) == Nought)
          && myState.at(position + 1) == myState.at(position)
          && myState.at(position + 2) == myState.at(position)) {
         int y = cellRect((position / 3), 0).center().y();
         painter.drawLine(0, y, width(), y);
         turnNumber = 9;
```

```
                winnerPosition = position;
        }
    }

    for (int position = 0; position < 3; ++position) {
        if ((myState.at(position) == Cross || myState.at(position) == Nought)
            && myState.at(position + 3) == myState.at(position)
            && myState.at(position + 6) == myState.at(position)) {
            int x = cellRect(0, position).center().x();
            painter.drawLine(x, 0, x, height());
            turnNumber = 9;
                winnerPosition = position;
        }
    }
    if ((myState.at(0) == Cross || myState.at(0) == Nought) && myState.at(4) == myState.at(0)
            && myState.at(8) == myState.at(0)) {
        painter.drawLine(0, 0, width(), height());
        turnNumber = 9;
            winnerPosition = 0;
    }
    if ((myState.at(2) == Cross || myState.at(2) == Nought) && myState.at(4) == myState.at(2)
            && myState.at(6) == myState.at(2)) {
        painter.drawLine(0, height(), width(), 0);
        turnNumber = 9;
            winnerPosition = 2;
    }
}

// Figure out how big the boxes are.
QRect TicTacToe::cellRect(int row, int column) const
{
    const int HMargin = width() / 30;
    const int VMargin = height() / 30;
    return QRect(column * cellWidth() + HMargin,
                 row * cellHeight() + VMargin,
                 cellWidth() - 2 * HMargin,
                 cellHeight() - 2 * VMargin);
}
```