# Tic-Tac-Toe

**CGL MINIPROJECT**

## PROBLEM DEFINITION

Design and implement a game of Tic-Tac-Toe or X and O, using concepts of computer graphics and Qt Creator in C++.

## CORE FUNCTIONALITIES

1. DDA Line Drawing Algorithm
2. Flood Fill polygon filling algorithm
3. Qt Signals and Slots
4. QMenu and Actions
5. C++ lambdas

## CONCEPTS

### 1. DDA LINE DRAWING ALGORITHM

The Digital Differential Analyzer (DDA) line drawing algorithm starts by calculating the smaller of dy or dx for a unit increment of the other. A line is then sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for the other coordinate.

Considering a line with positive slope, if the slope is less than or equal to 1, we sample at unit x intervals (dx=1) and compute successive y values as

$$y_{k+1} = y_k + m \qquad x_{k+1} = x_k + 1$$

Subscript k takes integer values starting from 0, for the 1st point and increases by 1 until endpoint is reached. y value is rounded off to the nearest integer to correspond to a screen pixel.

For lines with slope greater than 1, we reverse the role of x and y i.e. we sample at dy=1 and calculate consecutive x values as

$$x_{k+1} = x_k + \frac{1}{m} \qquad y_{k+1} = y_k + 1$$

## 2. FLOOD FILL POLYGON FILLING ALGORITHM

Flood fill, also called seed fill, is an algorithm that determines the area connected to a given node in a multidimensional array. The flood-fill algorithm takes three parameters: a start node, a target color, and a replacement color. The algorithm looks for all nodes in the array that are connected to the start node by a path of the target color and changes them to the replacement color.

Stack-based recursive implementation (four-way):

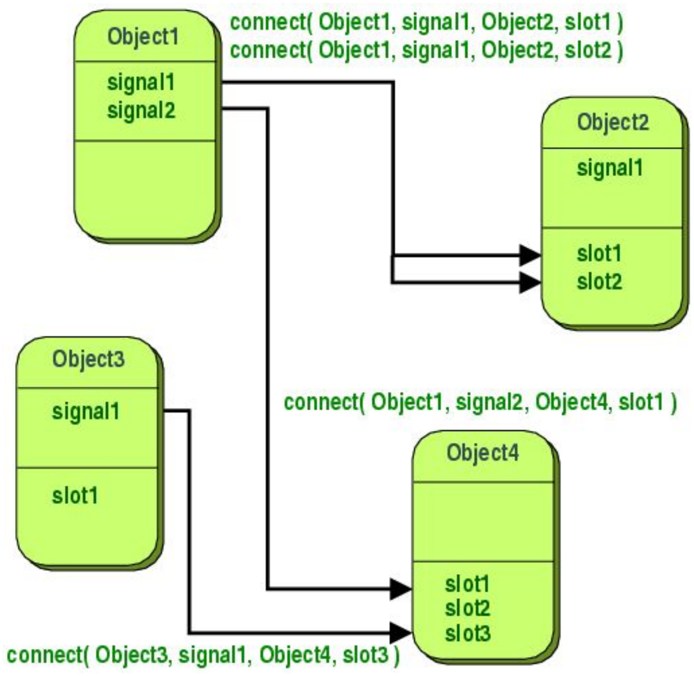
```
floodFill(float x, float y, QRgb fillcolor, QRgb bgcolor){
  If (image.pixel(x,y) == bgcolor) {
     image.setPixel(x,y,fillcolor);
     floodFill(x+1,y,fillcolor,bgcolor);
     floodFill(x,y+1,fillcolor,bgcolor);
     floodFill(x-1,y,fillcolor,bgcolor);
     floodFill(x,y-1,fillcolor,bgcolor);
  }
}
```

## 3. Qt Signals and Slots

Signals and slots are used for communication between objects. The signals and slots mechanism is a central feature of Qt and probably the part that differs most from the features provided by other frameworks. Signals and slots are made possible by Qt's meta-object system.

A signal is emitted when a particular event occurs. Qt's widgets have many predefined signals, but we can always subclass widgets to add our own signals to them. A slot is a function that is called in response to a particular signal. Signals and slots are connected using connect( ) function. Syntax:

QMetaObject::Connection QObject::connect(const QObject **sender*, const char **signal*, const QObject **receiver*, const char **method*, Qt::ConnectionType *type* = Qt::AutoConnection)


Object1
signal1
signal2
connect( Object1, signal1, Object2, slot1 )
connect( Object1, signal1, Object2, slot2 )
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect( Object1, signal2, Object4, slot1 )
Object4
slot1
slot2
slot3
connect( Object3, signal1, Object4, slot3 )

4. QMenu and Actions

The QMenu class provides a menu widget for use in menu bars, context menus, and other popup menus. They can be executed either asynchronously with popup() or synchronously with exec(). Menus can also be invoked in response to button presses; these are just like context menus except for how they are invoked.

A menu consists of a list of action items. Actions are added with the addAction(), addActions() and insertAction() functions. An action is represented vertically and rendered by QStyle. In addition, actions can have a text label, an optional icon drawn on the very left side, and shortcut key sequence such as "Ctrl+X".

Examples:

```
QMenu * menuGame = new QMenu(tr("&Game"));
ui->menubar->addMenu(menuGame);
newGame = new QAction("New Game");
newGame->setShortcut(QKeySequence::New);        //Adds Ctrl+N as shortcut
menuGame->addAction(newGame);
```

5. C++ lambdas

C++ 11 introduced lambda expressions to allow us to write an inline function which can be used for short snippets of code that are not going to be reused and not worth naming. In its simplest form lambda expression can be defined as follows:

```
[capture clause] (parameters) -> return type {  Method definition; }
```

These lambdas can also be used to connect slots and signals as shown below:

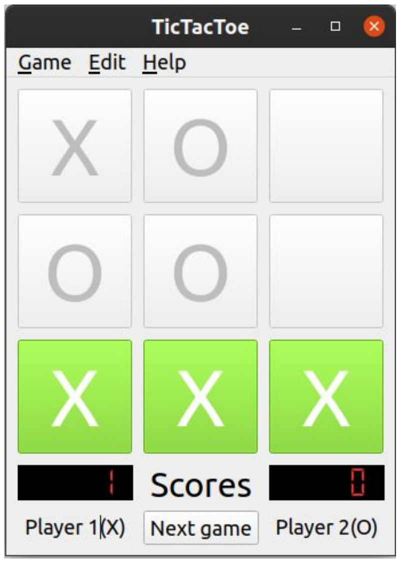
```
connect(newGame, &QAction::triggered, [this](){
    //function definition to initialize a new game
  });
```

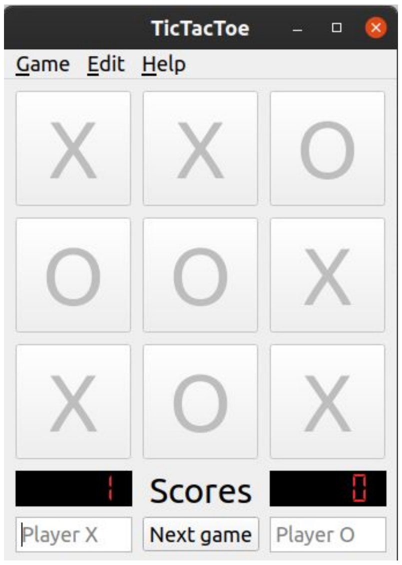
## ABOUT THE PROJECT

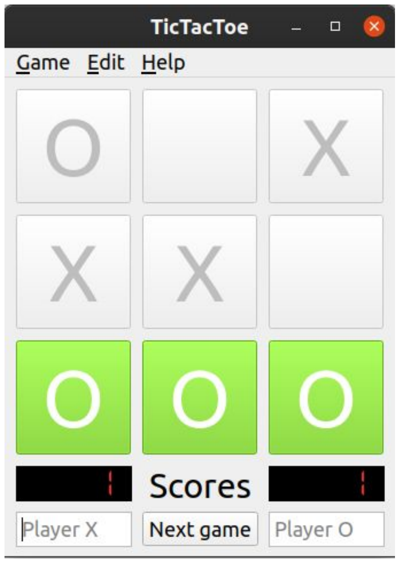
### TicTacToe

Tic-tac-toe is a game for 2 players, X and O, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. This implementation of Tic-Tac-Toe utilizes Qt widgets such as QlcdIndicator (to display scores), QButtons (to make a clickable grid), QMenu and Actions (to allow users to undo their actions and start new game) and Qt Style Sheets (to improve the appearance of the game). It uses a header file TicTacToe, which contains all the methods, variables, containers and slots required to implement clickable QButtons for accepting user input, moving to the next game, un-doing moves and to check victory, tie and re-setting the game board.

### Images


TicTacToe
Game
Edit
Help
X O
O O
X X X
1
Scores
0
Player 1(X)
Next game
Player 2(O)


TicTacToe
Game
Edit
Help
X X O
O O X
X O X
1
Scores
0
Player X
Next game
Player O


TicTacToe
Game
Edit
Help
O X
X X
O O O
1
Scores
1
Player X
Next game
Player O