IT6253 Assignment 2

Project: Tic Tac Toe

Table of Contents

[Purpose 2](#_Toc463553430)

[The GUI 2](#_Toc463553431)

[Designing the app 3](#_Toc463553432)

[Creating the app 3](#_Toc463553433)

[‘btnClick’ method 4](#_Toc463553434)

[‘checkIfWin’ method 4](#_Toc463553435)

[‘disableButtons’ 4](#_Toc463553436)

[‘newGame’ 4](#_Toc463553437)

[Running the code 4](#_Toc463553438)

[Appendix 5](#_Toc463553439)

[TicTacToeForm.cs 5](#_Toc463553440)

[Figure 1: The Tic Tac Toe Window 2](#_Toc463553441)

[Figure 2: selecting project type 3](#_Toc463553442)

[Figure 3: adding a method to an event 3](#_Toc463553443)

# Purpose

This project is to demonstrate some of the skills and knowledge acquired during the course. It was decided that a simple game app would be created; and although the code in itself is quite simple, it was developed mainly to provide an easy to use graphical user interface, with minimal uncontrolled input from the user.

# The GUI

The game is designed with only buttons within a form. This means that the user will only be able to click on buttons. The idea behind this is that it prevents any erroneous entry, thus no exception handling from the system.

Referring to the picture below, it can be seen that there are a total of 11 ‘objects’, which are the 11 buttons in the game. The first 9 displayed are where the game is played, meaning that they will be the ones displaying ‘X’ or ‘O’. The bottom ones are for the player to start a new game or exit the app.



Figure : The Tic Tac Toe Window

# Designing the app

As this app is solely a GUI oriented game, just the TicTacToe class can be used to create and store variables and methods.

As mentioned previously, there are 11 objects, or 11 buttons. For the sake of making things easy to follow, the playing buttons are named by row and column, for example a1 is the top left most one, and b2 is the middle one. And the 2 bottom ones are for game options.

We also need the following methods:

* display ‘X’ or ‘O’ when one of these buttons is pressed
* check if someone has won
* create a new game
* exit the game
* disable the play buttons when game is over

We also need 2 variables to store values used throughout the app, and are:

* turn (Boolean used to differentiate whose turn it is to play)
* turnCount (integer used to check how many turns have been played. When 9, game over).

# Creating the app

We have to first create a new Form Application.

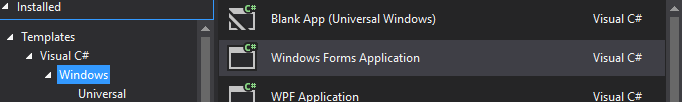


Figure : selecting project type

From there after adding the 11 buttons, we rename each of them as a1, a2, a3, b1, b2, b3, c1, c2, c3 (to store ’X’ or ‘O’) and ‘new game’ and ‘exit’. We also have to change the displayed text on each to nothing for a1 to c3.

For ease of use, one method will be used by all 9 play buttons. It will therefore be responsible of displaying the appropriate text, if someone wins, and whose turn it is. On top of that, we need to ensure that once a button has been pressed and assigned a value, it is not possible to click on it again.

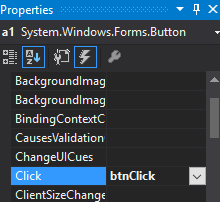
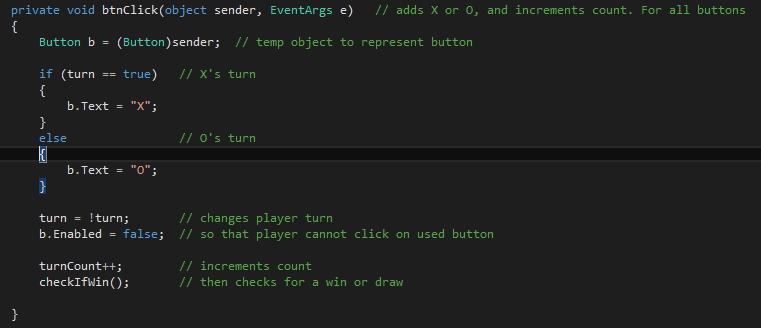


Figure : adding a method to an event

## ‘btnClick’ method

The ‘btnClick’ method is added to the event handling of each button, and when we double-click on one of the buttons, it is added to the TicTacToeForm.cs. The code is as follows:



As can be seen, depending on whose turn it is, it changes the text displayed on the button. It then reverses the Boolean value of ‘turn’ and disables the button. The turn counter is incremented and another method to check for a winner is called.

## ‘checkIfWin’ method

This method checks who won the game by checking if an entire row/column/diagonal has the same value. For that it will have to check the 8 lines (row a, b or c / column 1, 2, or 3 / either diagonals from a1 or a3). A Boolean variable stores whether a game is won or not. The ‘turn’ variable declares whose turn it was, and therefore the winner.

However, if by the end of the game, that is all buttons are pressed, and therefore turn counter reaches 9, then the method has to display that a draw has occurred.

## ‘disableButtons’

It is a simple method to disable all buttons using a for loop. When a winner is declared, the only buttons that should be allowed to be clicked are ‘New Game’ and ‘exit’. This simple method ensures this and is therefore called at the end of ‘checkWin’ method.

## ‘newGame’

This method is quite interesting in the way that we need to ensure that everything goes back to how it was before anything was pressed. Therefore, the following have to be done:

* A for loop used to clear text from all buttons (except the bottom two)
* All disabled buttons re-enabled
* Turn reset to X’s turn (true)
* Turn counter reset to 0

# Running the code

After opening the project, a simple build and running it will load the game on screen.

# Appendix

## TicTacToeForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace TicTacToe

{

public partial class TicTacToeForm : Form

{

bool turn = true; // when true it is X's turn, if false O's turn

int turnCount = 0; // maximum of turns is 9 before end of game

public TicTacToeForm()

{

InitializeComponent();

}

private void exit\_Click(object sender, EventArgs e) // this closes the game

{

Close();

}

private void btnClick(object sender, EventArgs e) // adds X or O, and increments count. For all buttons

{

Button b = (Button)sender; // temp object to represent button

if (turn == true) // X's turn

{

b.Text = "X";

}

else // O's turn

{

b.Text = "O";

}

turn = !turn; // changes player turn

b.Enabled = false; // so that player cannot click on used button

turnCount++; // increments count

checkIfWin(); // then checks for a win or draw

}

private void checkIfWin() // checks whether player wins

{

bool gameWon = false;

foreach(Control c in Controls) // checks all 8 winning combinations

{

if ((a1.Text == b1.Text) && (a1.Text == c1.Text) && !a1.Enabled)

{

gameWon = true;

}

else if ((a2.Text == b2.Text) && (a2.Text == c2.Text) && !a2.Enabled)

{

gameWon = true;

}

else if ((a3.Text == b3.Text) && (a3.Text == c3.Text) && !a3.Enabled)

{

gameWon = true;

}

else if ((a1.Text == a2.Text) && (a1.Text == a3.Text) && !a1.Enabled)

{

gameWon = true;

}

else if ((b1.Text == b2.Text) && (b1.Text == b3.Text) && !b1.Enabled)

{

gameWon = true;

}

else if ((c1.Text == c2.Text) && (c1.Text == c3.Text) && !c1.Enabled)

{

gameWon = true;

}

else if ((a1.Text == b2.Text) && (a1.Text == c3.Text) && !a1.Enabled)

{

gameWon = true;

}

else if ((a3.Text == b2.Text) && (a3.Text == c1.Text) && !a3.Enabled)

{

gameWon = true;

}

}

if (gameWon == true)

{

disableButtons(); // since game over player can't click on buttons anymore

String winner = "";

if (turn == true)

{

winner = "O";

}

else

{

winner = "X";

}

MessageBox.Show(winner +" is the winner!");

}

if (turnCount == 9) // if no-one wins by the end of count, Draw

{

MessageBox.Show("It's a draw!!!");

disableButtons();

}

}

private void disableButtons()

{

foreach (Control c in Controls)

{

Button b = (Button)c;

b.Enabled = false; // all buttons are disabled!

}

newGame.Enabled = true; // re-activating 'New Game' and 'exit'

exit.Enabled = true;

}

private void newGame\_Click(object sender, EventArgs e)

{

foreach (Control c in Controls)

{

Button b = (Button)c;

b.Enabled = true; // re-activating buttons for new game

b.Text = ""; // to clear Xs and Os

}

turn = true; // go back to X's turn

turnCount = 0; // reset counter

newGame.Text = "New Game";

exit.Text = "Exit";

}

}

}