In this lab you will explore using Finite State machines to build a small tic-tac-toe game. The tic-tac-toe game is simplified from the usual 3x3 type of game to a 2x2 type of game (called tic-tac). For example:

<table>
<thead>
<tr>
<th></th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>X</td>
</tr>
</tbody>
</table>

The front panel of the game will be built with four ‘OK’ buttons from the Boolean Controls Palette. Place the 4 controls close together to make a matrix. The computer begins by placing an X in some location of the matrix. We then use these buttons so that we can click on them and then select which move we want to place an O into. If one gets two in a row one wins (diagonal does not count).

The Game will start with the empty matrix, like the following:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>

The Computer begins by placing an X in a location:

<table>
<thead>
<tr>
<th>X</th>
</tr>
</thead>
</table>

You then click on one location, and in doing so change the value of the button text from a null string to an O. You will also disable the button so that a subsequent click cannot change it.

<table>
<thead>
<tr>
<th>X</th>
<th>O</th>
</tr>
</thead>
</table>

The Computer then selects a place to put another X (initially set so it can always win) and this matrix location receives the X and is also disabled.

<table>
<thead>
<tr>
<th>X</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The application will pop up a two button dialog box saying 'I Have Won'. The two buttons on the dialog box should say 'Play Again' and 'quit', where 'Quit' stops the program and 'Play again' clears the board and allows you to play again. You should also add a button on the front panel for 'Restart' which will clear the board and restart.

In order to simplify the game, we will make two changes:

1) The computer starts first, and always moves in the upper left corner with an X.
2) The computer stops as soon as it wins.

To do this lab you will have to do multiple things:

1) Learn how to dynamically change the names on the OK buttons, and disable.

   a) Setup: Do this by creating property nodes for each button (Right click while on the panel view, under 'Create'. On the diagram side you need to add another element to each property node (right-click, choose 'add element'). Then set one element to 'Disabled', the other to 'BoolText.text' by searching through the 'Properties' menu on the right click. Also change both elements to write.
b) Changing strings: You should now be able to attach a string to the 'Booltext.text' and display it in the button when you run the program. To make a selection, try building an array of strings, then selecting a single element of the array with an index. Set the array of strings to be 'X', 'O', and " " (an empty string). Test with a numeric control (make it U8!); when the program runs change the values of the numeric control to select different values of the text to get displayed to the button.

c) Disabling: Add a boolean switch, and in the dialog box convert the boolean to a number using the "Boolean to (0:1)" function. Send the output of this to the Disable element of the button. If I pass a 0 to the disable element, the button behaves normally. If I pass a 1, it is disabled (FYI if I pass a 2 to the button, the button gets grayed out and is disabled). By running the application you should be able to enable or disable the button at will, as well as change its caption.

2) Design a finite state machine on paper.

a) Work through the states of a finite state machine to build to make the tic-tac game. Start by assuming an unchecked space has an 0, a space with an X has a 1, and a space with an O has a 2. Each state could then have a label abcd, as below.

i) What is the initial state? Draw a circle for this one and label with

```
00
00
```

ab

So this is state abcd=0000

ii) What are the first four possible states (primary states)? In a row below the initial state, draw a circle for each primary state and provide a label. For instance, if I chose to put an X in upper left corner the primary state is labeled as

```
10
00
```

This is therefore state abcd = 1000

Draw an arrow from the initial state to each of the four different possible first states.

iii) Now draw a row of secondary states below the row of primary states. We will simplify by following only one of the primary states in this lab, otherwise the number of states grows tremendously. So for only state abcd=1000 of the first possible initial states, define the secondary states. For instance if O chooses to move into bottom left, this would be a state abcd=1020

```
10
20
```

Each secondary state should be unique, but there may be arrows from two different primary states to a single secondary state.

iv) Now define the third level states and fourth level states. Note that the computer can win in every case, so the third level states are essentially predefined for two of the secondary states, but one of the secondary states has two different third state possibilities. There is only one fourth level state, where the computer prints out its victory message and waits.

b) Now that you have the state diagram, you have a certain number of unique states. Assign each of the above states a number. This is an arbitrary assignment and does not have to be in any order. The easy way to do this is to name them abcd, i.e state 1020, 2010, 1120, 0010, etc. In this case you have almost no work
to do. Note that state 0000 would be interpreted by the case statement in Labview as state 0. The final
victor state can be any number you have not already used; I have used 9999 in my solution.

3) Use the 'case' statement imbedded within a 'while' loop to generate the finite element machine that
jumps between the various states. In each state, assess what the input data is (the values of each of the 4
button in the matrix as well as the 'restart' switch) and decide upon the next state to jump to. Pass this state
back to the case statement using the tunnel in the While loop. If you choose to number the states as 1020,
2010, etc, then as you add each case to the case statement, change the case value to these numbers. State
0000 would be interpreted as state 0.

a) You will need to have a sort of numeric scratch pad where you can write some data and store it
as well as read it back. The strategy here will be to use a local variable. First create four numeric controls.
Each represents the states of each of the 4 matrix check boxes. Make each have a representation of U8.
You can later on hide the numeric controls from the front panel by right clicking on the numeric control
from the front panel, and clicking on 'Hide control' (located under submenu 'advanced'). Make sure these
numeric controls are outside of your case statement which controls the finite state machine sequence, but
inside the overall while loop. Use these variables also to select the caption of the individual matrix element
buttons (using the Array select, as in the first part) as well as to disable the matrix element button (check to
see if the variable is non-zero. If it is, then disable the button).

b) Go to your initial state abcd=0000 in the diagram view. Right click on the data object to select
either Create-> Local Variable. Drop the local variable for each of the 4 numerical controls buttons into the
abcd=0000 state. Assign all of these value 0, except a, which will have value 2. You can use numeric
constants to assign these values. Setup the jump to the next state abcd=1000.

iii) Go to next state abcd =1000. Again, Add local variables for bcd into this case. Now assign b,c,d from
the b, c, and matrix switch values. Then setup a new state to jump to based upon these switch values (Hint:
form a integer P= a+2*b+4*c, then pass this to another case statement within this particular case statement.
Select what the next state will be depending upon P=0, P=1, P=2 or P=4 !).

4) One state will popup a dialog box. Be sure to wire the cancel output to stop the program.

5) One state abcd=1002 will have two possible computer moves to win. Use a random number generator to
select which path will occur. The random number generator makes a number between 0 and 1.0 So
compare the random number with 0.5, if it is less, the jump to one state, if it is more, jump to each other

6) Be sure to initialize the state to 0 outside the main while loop. You probably will want to initialize each
of the numeric matrix values as well here.

7) try using the debug features of LabView as they will help you to discover problems. In particular, it can
help you to trace through the steps that your finite state machine is progressing through, and help you to
diagnose when you are not jumping to the correct state.