1 TIC TAC TOE PLC ALGORITHM.

1.1 OVERVIEW

As a kid I was intrigued by a relay based Tic Tac Toe machine at the Museum of Science and Industry in Los Angeles. In high school I "borrowed" 500 relays with the intent of building my own Tic Tack Toe machine. I figured out how to build combinational logic, shift registers and sequencers from the relays, but never got much farther. Now, about 200 years later, I decided to make another attempt to build a Tic Tac Toe machine, but this time using ladder logic and virtual relays.

This document is intended to help explain got Tic Tac Toe to run on Triangle Research International's PLCs.

1.2 GAME PLAY APPROACH

To keep the machine very simple, I made the human play first as "X" and the PLC will respond as "0". The game strategy of the computer in response to each time the human claims a square is as follows:

1. Check to see if the computer can win, and if so claim that square and declare victory
2. If the computer can't win, then check to see if I can block the human from wining.
3. If the computer didn't win and didn't need to block the human then there are some special case patterns that need to be addressed. If one of these special patterns is found, then claim the square that to best handle the condition.
4. If the computer gets through the first 3 tests and hasn't made a move then pick a square based on this order:
    a. Center Square
    b. Corner Square
    c. Side Square
5. If we get here then there are no more squares available and no one won. Declare that a dead lock or "Cat's Game" has been reached. Game over.

1.3 TIC TAC TOE BOARD NAMING

The game is played on 3x3 grid. The PLC program manages several RELAYs for each cell or square. If a RELAY refers to the top left corner of the grid it will have a name like "R1C1_P" where the "R1C1" indicates that this relay is in row 1, column 1 of the grid and the "_P" indicates that this relay is controlled by the human player.

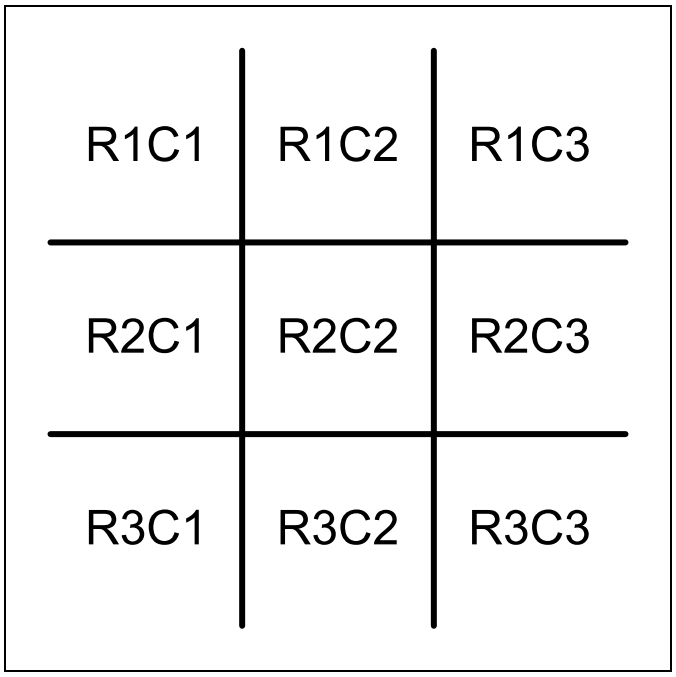

Figure 1 Tic Tac Toe Square/Cell Naming

The PLC program uses 8 RELAYS for each cell in the grid to manage the state of that cell. For example the center cell is represented by the following RELAYs:

Table 1 RELAY Naming Convention

Relay Name	Description
R2C2_P	Set by human, via HMI, to request this cell
R2C2_X	Indicates that this cell is occupied by “X” the human player
R2C2_O	Indicates that this cell is occupied by “O” the computer
R2C2_Free	Indicates that this cell is free and not claimed by “X” or “O”
R2C2_Win	RELAY set by “Win” state machine to claim this cell for the win
R2C2_Block	RELAY set by “Block” state machine to claim this cell block the human
R2C2_Special	RELAY set by “Special Move” state machine to claim this cell
R2C2_Move	RELAY set by “Move” state machine to claim this cell

As there are 9 cells on the game board and 8 RELAYS for each cell, there are 72 RELAYS just to manage the game board.

There is another important detail about the PLC implementation. The TBASIC programming language provides 2 methods to access RELAYS:

1. Single Relay at time. The SetIO, ClrIO and TestIO() statements/functions can be used to access a single RELAY at a time using the RELAY’s name. Think of this as accessing a single bit. Ladder logic handles this sort of thing easily.
2. 16-bits at a time. The programming language provides a mechanism where RELAYs can in groups of 16 RELAYs. RELAYs 1..16 can be accessed as a 16-bit signed integer as the RELAY[1]. This would require 16 rungs of ladder logic.

The RELAY, R2C2_P, is the RELAY that is set by the HMI to indicate that the human wants to take the center square in the game board. The PLC can clear this RELAY from a custom function using the following code:

```
ClrIO R2C2_P
```

The PLC can clear RELAYS R1C1_P through R3C3_P and the RELAY NewGame in a custom function like this:

```
RELAY[1] = 0
```

You will not find that sort of code in any of my PLC programs. The reason is that "RELAY[1]" doesn't give me or the next idiot that has the task to fix my code much of an ideas as to why or what the statement is attempting to do. I use the #Define mechanism in the custom function editor to allow me to assign a more more useful name to RELAY[1]. I have used the #Define mechanism to allow me to use the text "UserInput" to reference RELAY[1]. I prefer to write code in this style:

```
UserInput = 0
```

Table 2 Game Play RELAY Info

UserInput RELAY[1]		FreeSpace RELAY[2]		Xspace RELAY[4]		Ospace RELAY[5]		GameStatus, RELAY[2]	
I/O #	Name	I/O #	Name	I/O #	Name	I/O #	Name	I/O #	Name
1	R1C1_P	33	R1C1_Free	49	R1C1_X	65	R1C1_O	17	Play
2	R1C2_P	34	R1C2_Free	50	R1C2_X	66	R1C2_O	18	Draw
3	R1C3_P	35	R1C3_Free	51	R1C3_X	67	R1C3_O	19	Win
4	R2C1_P	36	R2C1_Free	52	R2C1_X	68	R2C1_O	20	
5	R2C2_P	37	R2C2_Free	53	R2C2_X	69	R2C2_O	21	
6	R2C3_P	38	R2C3_Free	54	R2C3_X	70	R2C3_O	22	
7	R3C1_P	39	R3C1_Free	55	R3C1_X	71	R3C1_O	23	
8	R3C2_P	40	R3C2_Free	56	R3C2_X	72	R3C2_O	24	
9	R3C3_P	41	R3C3_Free	57	R3C3_X	73	R3C3_O	25	
10		42		58		74		26	
11		43		59		75		27	
12		44		60		76		28	
13		45		61		77		29	
14		46		62		78		30	
15		47		63		79		31	
16	NewGame	48		64		80		32	

OK there is some more things that I could have done with being able to access RELAYS in custom functions as either single RELAYs or as groups of 16 RELAYs. There are 9 RELAYs that are used as a map of the currently available board spaces and these RELAYs are named R1C1_Free through R3C3_Free. A custom function can update all 9 of the FreeSpace RELAYs, in a single assignment statement:

```
FreeSpace = ~(XSpace | OSpace)& &h01ff
```

But, I chose not to manage the board space using custom functions. Why? My goal was to not use custom functions for this PLC program, if possible. So I solved this problem with ladder logic. Figure 2 Ladder Logic Free Space Management, shows the first 2 ladder rungs of the 9 rungs of logic that were necessary to duplicate what could have been done in a single line of TBASIC.

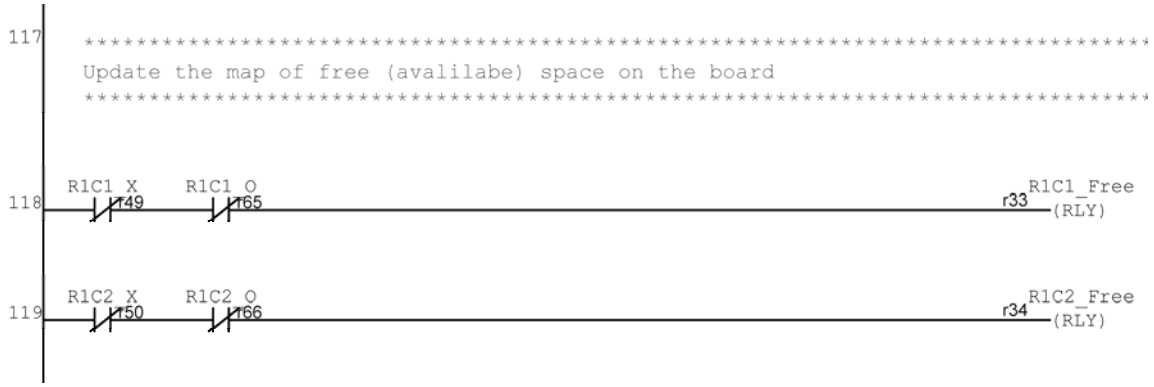

Figure 2 Ladder Logic Free Space Management

OK. Why am I telling you all of this stuff? You need to know that the position of these RELAYS in the I/O table is very critical. If you insert or delete a RELAY in the I/O table this will shift the RELAYs and and they may not be in the same position as the PLC and HMI expect them to be. There is nothing in the PLC programming that can detect that you have done this and if you are cavalier with making changes, you will spend endless hours trying to figure out why the program quit working. You have been warned!

1.4 PLC ALGORITHM

In response to a move by the human, the PLC evaluates the current state of the board to determine the best possible move for the computer. The algorithm that I used is dumber than a box of rocks and can make up to about 36 different evaluations before setting on the best possible move.

The approach that I have taken with the PLC code implementation is to write it as a series of state machines. I have a master state machine that handles the overall game. Four slave state machines that handle the evaluations required to determine the best possible move for the computer.

The use of state machine solves a bunch of problems:

1. Provides the mechanism to allow the PLC ladder logic to work through a complicated sequence of evaluations.
2. Helps to debug the PLC program before I have written it. I draw the state diagram on a big piece of paper for each state machine and then try to break it before writing any PLC code.
3. Documentation for the PLC program. My entire working experience has been involved in writing computer programs. In my experience, ladder logic is probably one of the best programs to use to ensure that no one will ever understand what you were intending to do. I can explain the state diagrams (circles and arrows) years later.
4. Helps debug the running PLC program. The state number that shows up in the state diagram is the same as the current value of a PLC counter
5. Allows for the easy addition of new features without breaking the working code.
6. Eliminates the necessity to be sober while programming. If you get the state machine designed right on paper, the PLC coding is trivial enough to do while drinking.

If you are not an aging computer science student, then you can google Moore State Machines or (Mealy State Machines). Trust me, I have not invented anything here, I am just using what was published in the 1950s and still being taught at universities.

Here's the short version of the state machine design that you missed in school:

1. Each state is unique. The state machine is only in a single state at a time.
2. There are unique rules to enter and exit each state. These are the state transition rules.
3. RELAYS and OUTPUTs are controlled by the current state and other inputs to the state machine.
4. Custom Functions are executed on entry to a state.
5. Each state is active for a minimum of one complete scan of the ladder logic engine. This is required because of the way that the PLC ladder logic scanner operates.

2 MAIN LOOP STATE MACHINE.

Have a peek at Figure 3 Main Loop State Machine Diagram. The circles are the states. Note that in each state is a label in the form of "Seq1:n" where the Seq1 is the name of the COUNTER that manages the state and the "n" is the state number. The TRI PLCs support eight specialized COUTNERs that they refer to as sequencers. Each of these sequencers can support 32 states. It is trivial to translate a state diagram to ladder logic.

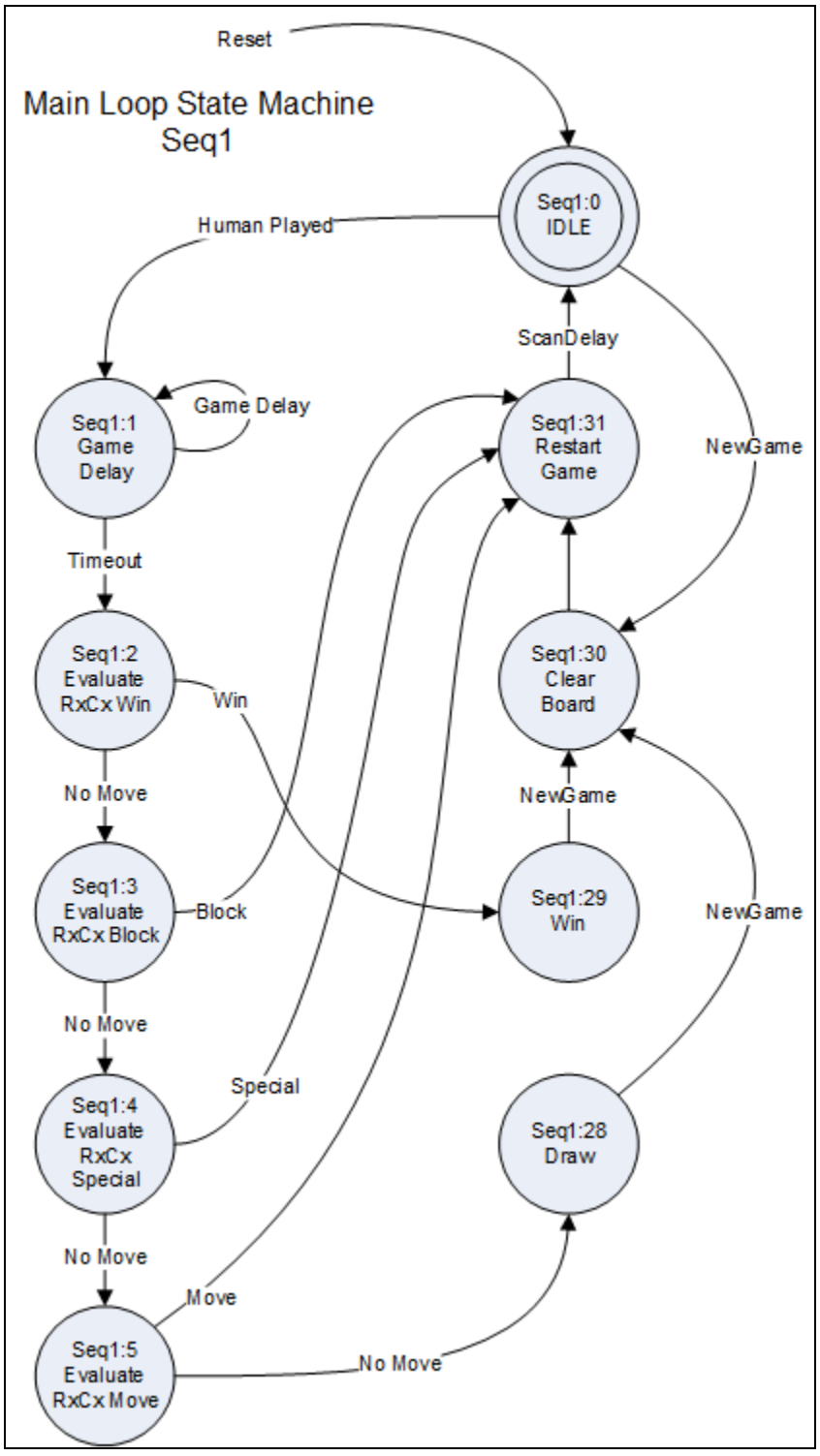

Figure 3 Main Loop State Machine Diagram

OK. We've settled on the circles as being the states. The arrows that go from state to state are the transition rules. If the state machine is sitting at any given state, it will not move from that state until there is a rule that is true and then and only then can it move to a new state.

There is one other thing to understand about my usage of state diagrams. Look at State 1, Seq:1, you will see an arrow that comes out of the circle and then points back into the circle. Indicates that this state has some sort of time dependency. In this case, the state will be maintained until some programmed times has expired.

2.1 STATE 0, IDLE

State 0, Seq1:0, is the IDLE state. This state is entered on power on reset. The only time that the PLC leaves this state is in response to the human picking a square.

2.2 STATE 1, GAME DELAY

State 1, Seq1:1, is entered from State 0 only when the human player makes a move. The purpose of this state is to insert a time delay. This delay provides enough time for the HMI to update its version of the game board after the human moves and before the PLC makes its move. The PLC TIMER, is enabled while in this state

2.3 STATE 2, EVALUATE RXCX WIN

State 2, Seq1:2 is entered from State 1 after the TIMER, GameDelay, is true. This state is used to enable the RxCxWin state machine to determine if the PLC can win the game by picking a single square. The results for the RxCxWin state machine determine the next state for the main state machine. If the PLC has made the winning move, then the next state will be State 29, Win. If the PLC cannot win and has made no move then the next state is State 3, Evaluate RxCx Block.is no wining move, then the

2.4 STATE 3, EVALUATE RXCX BLOCK

State 3, Seq1:3 is entered from State 2. This state is used to enable the RxCxBlock state machine to determine if the PLC must pick a square to block the human from wining. If the RxCxBlock state machine picks a square to block the human, then the main state machines next state is State 28. If the RxCxBlock state machine determines that the PLC does not have to block the human the next state is State 4.

2.5 STATE 4, EVALUATE RXCX SPECIAL

State 4, Seq1:4 is entered from State 3. This state is used to enable the RxCxSpecial state machine to determine if the PLC can pick a square that falls into a set of specialized cases to prevent the human player from being able to beat the PLC on a future move. If the RxCxSpecial state machine picks a square, then the main state machines next state is State 27. If the RxCxSpecial state machine does not pick a square, determines that the next state is State 5.

2.6 STATE 5, EVALUATE RXCX MOVE

State 5, Seq1:5 is entered from State 4. This state is used to enable the RxCxMove state machine to determine the best square for the PLC to pick. If the RxCxMove state machine picks a square, then the main state machines next state is State 26. If the RxCxMove state machine cannot pick

a square because none are available, then the next state will be State 25 not pick a square, determines that the next state is State 5 indicating that the game was a draw or a “cat’s game”.

2.7 State 28, Draw

State 28, Seq1:28, is entered from State 5 if there is no square available for the PLC to pick. Neither the human nor the PLC can win. This state is controls a RELAY that changes the game status to “Draw” to allow the HMI to display the ending game status. The only way out of this state is to either reset the PLC or for the human to set the NewGame RELAY indicating that the human wants to start a new game.

2.8 State 29, Win

State 29, Seq1:29, is entered from State 2 if there is the square picked results in the PLC winning the game. This state is controls a RELAY that changes the game status to “Win” to allow the HMI to display the ending game status. The only way out of this state is to either reset the PLC or for the human to set the NewGame RELAY indicating that the human wants to start a new game.

2.9 State 30, Clear Board

State 30, Seq1:30, is entered from States 0, 28, or 29 if the NewGame RELAY is active. This state invokes a custom function to clear the game board. I got lazy and used a custom function for this purpose as it was significantly easier to code then to add ladder logic to clear 28 RELAYS that manage the game board state

2.10 State 31, Restart Game

State 31, Seq1:31, is entered from States 3, 4 and 5 if the PLC picks a square in these states. Additionally, if the main state machine was in state 30 then this state is logically next. This state serves very little purpose other than forcing a redraw of one of the debug mechanisms that maintains the current game state in the string variables A$..E.

Probably the most important thing about this state is that the next state is Seq1:0, Idle. The rule to get from State 31 back to State is that the RELAY, ScanDelay, must be active. ScanDelay is only active when the current state is state 31. This RELAY is used to ensure that State 31 is is maintained for a complete scan of the ladder logic.

2.11 Main State Machine Implementation Notes

i-TRiLOGI supports a specialized version of the COUNTER known as a sequencer. The first eight counters can acts as a sequencer when they are named Seq1..Seq8. The magic of a sequencer is that ladder logic can directly set the current value of a sequencer with the [StepN] coil. The ladder logic can directly act on the current value of the sequencer and you will see that for sequencer #1 that the contacts are named Seq1:0..Seq1:31.