2.11.1 State Transition rules

The rules that determine the current state for the state machine are implemented in ladder logic. Compare Figure 3 Main Loop State Machine Diagram with the ladder logic in Figure 4 Main State Machine Transition Rules. Compare the circle and arrow diagram against the ladder logic and you will find that for every arrow in one figure there is a bit of ladder logic that ensures that the PLC code will match the diagram.

The rules are written backwards. The rule for state 31, Seq1:31, is written first. The rule for State 30 will follow and the last rule is how we get to State 0. This is done to ensure that the ladder logic scanner will set the current state once and only once per scan of the ladder logic.

Look at the rule that determines when the state machine is in state 0, Seq:0. Notice that there are two possible ways to get to state 0, on the first scan of the ladder logic and from State 31, Seq:31. The first scan part is how I handle initialization of the state machine on power up and this should make perfect sense. Going from state 31 to state 0 is a bit tricky and the ScanDelay contact is added into this rung to prevent a terrible problem.

OK what is the problem? It is possible for the ladder logic scanner to evaluate the first rung and set the current state to 31 and then when it gets to the last rung change to state 0. The problem is that the the ladder logic that might be active when in state 31 will never be evaluated. The fix was to add the ScanDelay contact. What is done is that latter in the PLC program you will find that the ScanDelay RELAY is only asserted when we are in state 31. Since we just transitioned to state 31, this RELAY will not be asserted and the ladder logic scanner will not change the state to 0.

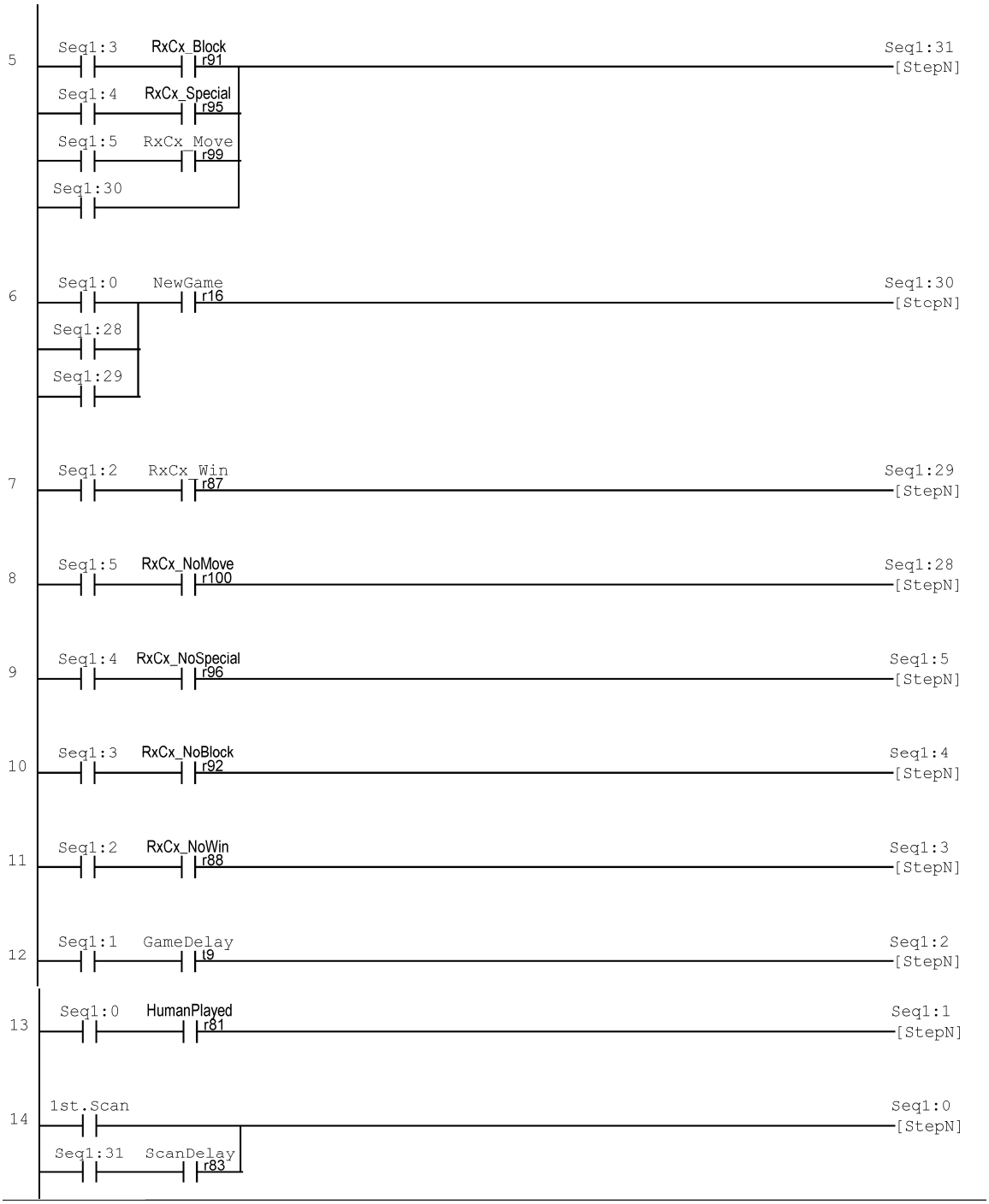

Figure 4 Main State Machine Transition Rules

2.11.2 What PLC hardware is affected by each state?

The primary task of the Main State Machine is to sequence the running of the subordinate state machines to determine which square the PLC should take.

*Error! Reference source not found.*Seq1:1 – while in this state the TIMER, GameDelay is enabled. This TIMER determines the amount of time that this state is maintained. When the TIMER's contact goes active the state will changed based on the state transition rules.

Seq1:2 .. Seq1:5 – each of these states results in a single RELAY being made active. The active RELAY enables a slave state machine that will make form 1 to 9 evaluations to determine if a move by the computer can be made. Think of the main state state machine as a benevolent overlord with the real work being done by the slaves.

Seq1:31 – when in this state, the RELAY, ScanDelay is active. This RELAY is used by the state transition logic to ensure that the state machine will stay in this state for a minimum of one full scan time.

Now look at the bottom of the ladder logic in the previous figure. This section of logic controls three RELAYs: Play, Win and Draw. These RELAYS provide the current state of the PLC to an external HMI. Later on I'll let you know that the HMI reads all 3 of these RELAYS as part of a single 16-bit transfer.

2.11.3 Custom Functions

The hardware functions were described in the previous section. Now we will look at what custom functions are invoked at each state. Please refer to Figure 5 for the custom function usage for the main state machine.

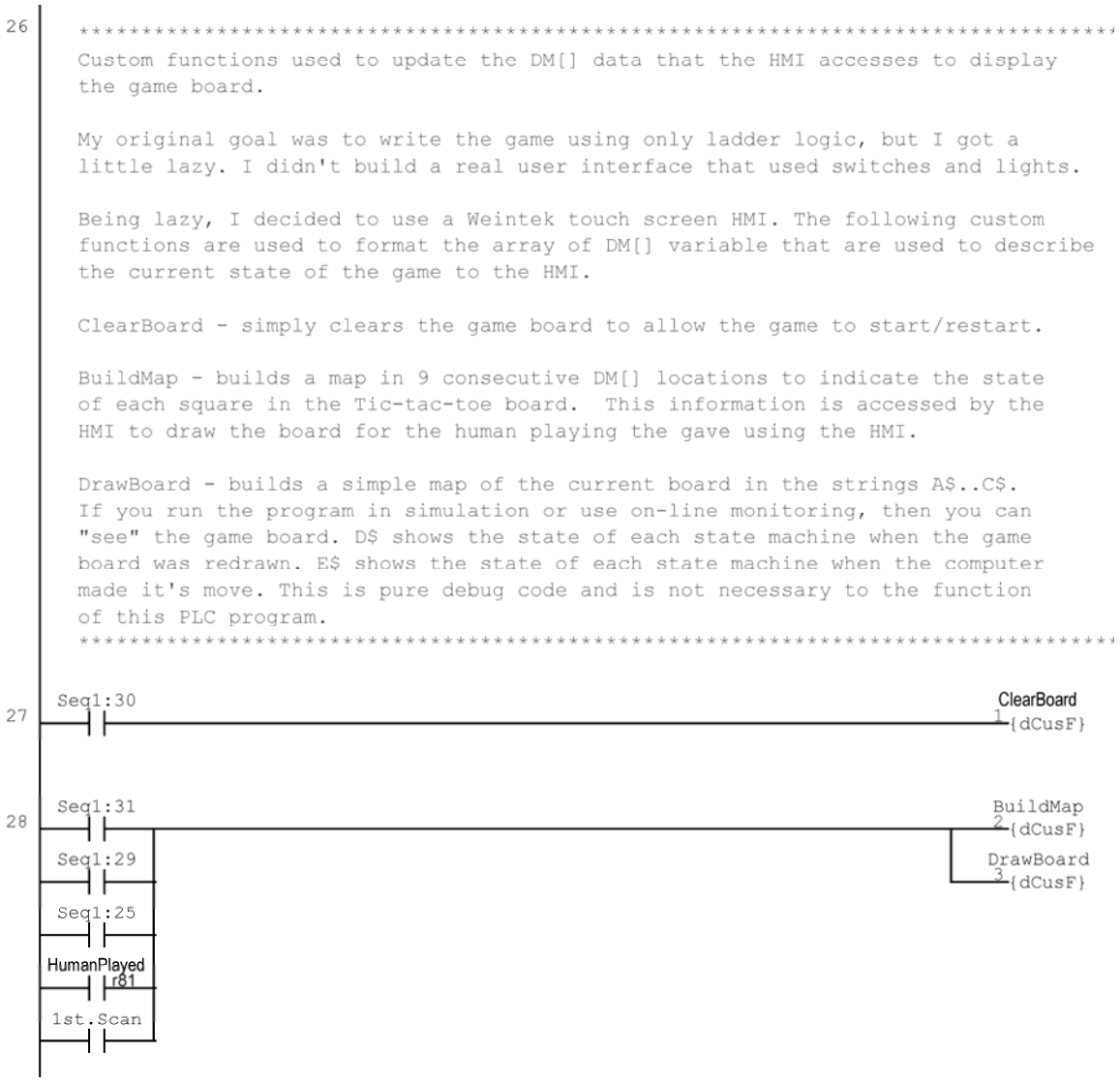

Figure 5 Main Machine Custom Functions

A quick count will reveal that there are only 3 custom functions used by the main state machine.

The ClearBoard custom function initializes all of the RELAYS that are involved in both the game board and any input requests from the human. This is all that happens in Clear Board:

```
UserInput = 0    ' clear all relays that the HMI can set
XSpace = 0       ' clear map of spaces occupied by "X"
OSpace = 0       ' clear map of spaces occupied by "O"
```

The BuildMap CF builds a map of 9 memory locations in DM[]. The state of these 9 DM[] locations is used by the HMI to draw a human-readable version of the game board. The code for this custom function is bordering on pretty cryptic:

```
for i = 0 to 8                    ' for each of the 9 squares...
     if TestBit(XSpace,i)
```

```
            DM[RxCxBase + i] = 1  ' X is in this square
      elif TestBit(OSpace,i)
            DM[RxCxBase + i] = 2  ' O is in this square
      else
            DM[RxCxBase + i] = 0  ' This square is available
      endif
next
```

The DrawBoard CF is pure debug and serves only to allow one to play the game without an HMI and to play the game from the simulator without any actual hardware. It builds the game board. The following figure is a screen shot of what you can see if you inspect the string variables using On-line Monitoring. Notice that A$..C$ forms the 3x3 game board. D$ and E$ report the state of all 5 state machines at various points in time.

Figure 6 Game Play Screen for Debug

2.11.4 Notes on state machine implementation that are worth mentioning

The state transition rules are designed so that the ladder logic scanner will only make a single state transition per scan of the logic.

The state transition rules are the first block of ladder logic that is scanned. No actions are taken until the current state is 100% known.

The “hardware” actions always follow the rules section.

Custom functions are the last group of things that get scanned. Most of my custom functions are only called on the transition to the state. This is simply my style. I can give you a host of reasons why I decided to do this, but you probably don’t need a lecture.