State Machine Programming Guide

Introduction:
In this guide, state machines will be explored. State machine diagrams will be explained, and steps to create a state machine in the LabVIEW™ for LEGO® MINDSTORMS® programming language will be outlined.

State Machines
A state machine is used to break up a robot’s actions into a series of modes, or states. State machines make decisions on what actions are to be performed next, based on the state the robot is in and can include other conditions. These decisions and state changes are generally made based on values input by sensors, as well as other information collected throughout the program.

In LabVIEW, a state machine is created by placing a Case structure within a While Loop. To define states, strings, or words, are generally used to label different cases in the Case structure. A string can be wired into the Case selector of a Case structure to define multiple cases, instead of just the true and false cases that can be defined by wiring in a Boolean value. To create a string constant, simply select it from the String sub-palette of NXT Programming, and type in the required text. This method makes it very easy to distinguish between states because they can be given descriptive names.

State Diagrams
A state diagram is a visual representation of a state machine. It shows program flow using arrows for the transitions, which connect the states, represented as circles. The transition arrows are often labelled with an action that happens during the transition, or an input which triggers the transition. It is possible to have multiple transitions to or from any specific state, representing different paths the program could take.

This state diagram represents a program that will stay in an idle state, or a state in which the robot is simply waiting for an event to occur, until an object is detected by the ultrasonic sensor, or the touch sensor is pressed. In the case that an object is detected, it will move onto the Forward state. In the case that the touch sensor is pressed, the program will terminate, as it goes to the Stop case. Notice that this is similar in all the cases; if the touch sensor is pressed, the program terminates. When the robot reaches the forward state and the touch sensor is not pressed, it will go forward for one second, and then go directly to the Backward state, where it will go backward for one second if the touch sensor is still not pressed. It will then return to the Idle state, and continue in the loop until the touch sensor is pressed to end the program.
State Machine Programming Guide

Create a Simple State Machine:

The following steps will create the state machine shown here.

1. Create a Case structure within a While Loop.
   a. Create a While Loop by selecting While Loop from the Structures sub-palette, and then dragging the cursor across the screen.
   b. Create a Case structure by selecting Case structure from the Structures sub-palette, and then dragging the cursor across the screen within the While Loop.

2. Add a shift register to the loop and initialize it with a string constant set to “1st State”.
   a. Add a shift register to the loop by right-clicking the edge of the loop and selecting the Add Shift Register option.
   b. Create a String Constant by right-clicking on the block diagram and selecting it from the String sub-palette of NXT Programming.
c. Type “1st State” into the string constant, and wire it into the shift register.

![Diagram showing shift register connected to a string constant labeled '1st State'.]

d. Wire the output of the shift register to the Case selector. The Case selector will turn pink, to represent string data.

![Diagram showing shift register connected to a Case selector labeled '1st State'.]

3. Create a state called “1st State” in the Case structure, and create the necessary code inside this case.
   
   a. Change the text where it says “True” to say “1st State”. It is very important that this is spelled the exact same way as in the constant that was just created.
   
   b. Create a string constant inside the case and set it to “2nd State”.
   
   c. Wire the string constant to the input of the shift register. This will send it to the second state in the next iteration of the loop.

![Diagram showing shift register connected to a Case selector labeled '1st State' and connected to a string constant labeled '2nd State'.]

4. Create a state called “2nd State” in the Case structure, and create the necessary code inside this case.
   
   a. Right-click the Case structure and select Add Case After.
b. Name this case “2nd State”.

c. Create a string constant inside the case and set it to “Stop”.

d. Wire the string constant to the input of the shift register. This will send it to the stop state in the next iteration of the loop.

5. Make the default case, currently labelled “False”, the Stop case. Create the necessary code inside this case.

a. Use the arrows at the top of the Case structure to navigate to the case currently labelled “False”.

b. Change this to say “Stop”.

c. Create a Boolean Constant by selecting it from the Boolean sub-palette of NXT Programming. Set this constant to True.

d. Wire the Boolean constant into the conditional terminal of the While Loop. This will stop the program when the Stop case is executed.

6. Notice that the pink box where the input of the shift register connects to the Case structure has a white center. As stated in previous lessons, this is because it has not been wired in all cases.

a. Wire an empty string constant into the shift register in the Stop case. Since the Stop case terminates the program, this constant will be sent to the shift register, but will not ever be used.
State Machine Programming Guide

7. The same problem occurs with the green box where the conditional terminal connects to the Case structure, but this will be fixed in a different manner.

a. Right-click the green box and select Use Default If Unwired. The default value here is False, this cannot be changed. This method should be used with caution, as LabVIEW™ will no longer give an error for unwired cases, which means wiring could be forgotten in which case the program would not execute as expected.