State Machines

State Machines are another programming concept with an origin in mathematics. Ant although there entire text books and many college classes are devoted to them, the the basic principles are pretty darn simple.

In fact, basic state machines are so simple that many programmers will eventually start using them inadvertently. But knowing a bit about them will allow you to research as much about them as you desire.

It's often the case that your program needs to make a decision based not just on a single set of current inputs, but based on a series of inputs that have occurred in the past. Here's a simple real-world example. Imagine that you have a fan that can operate at four speeds: Off, Low, Medium, & High. To control the fan there are just two buttons: Up and Down. When a button is pressed, how can you determine what speed to run the fan?

You could store all the inputs from both buttons in an array. Then every time a button is pushed, sum up the number of each button, subtract the 'low' count from the 'high' count and try to set the speed from that. In addition to taking more time to calculate the more times a button is pressed. And what will happen if up is pressed 10 times in a row, then a single down press?

Using a state machine, requires remembering only one item, the current speed of the fan, and building a few rules. For example, If you are on Medium and receive an Up input, you need to transition to High. You can think of the speeds of the fan as modes, or 'states', and the buttons as inputs. When a button is pressed, a state transition is needed, and a state machine describes the rules for those transitions. The arrows in the graphic describe the states.

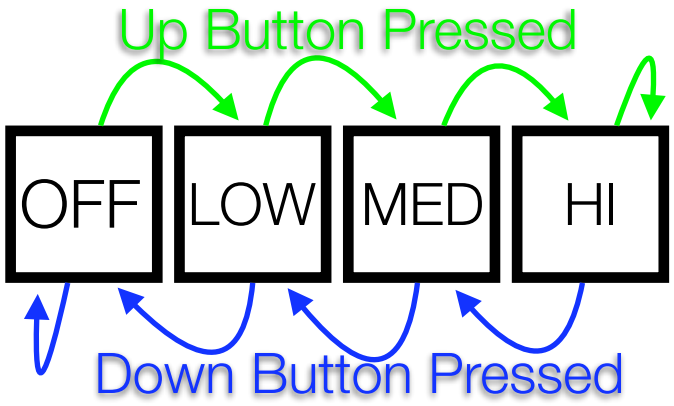
Note that sometimes the state transition actually returns to the original state, such as in the case of a Down button press when already Off.

While something as trivial as our fan state demo can be handled with a small number of simple comparisons. One can imagine that as the number of states, inputs and potential transitions increases, the idea of formalizing these details will simplify and streamline the programming.

Some other common real-world state machine examples are would be stop lights, a vending machine, or an elevator. Both can start out simple but become quite complex as you add details and think about all of the possible states and transitions. For example the elevator can be pretty simple if there are only two floors and you consider only the up and down button in the car. But if you add floors, consider all the call buttons, door open and close buttons, floor number buttons and cancels, the fireman/super over-ride switch, etc. things start getting out of hand.

State machines are also used extensively in video game programming. State machines may handle simple items such as drawing animations, or more complicated things like determining how a monster reacts to a player or if a player has completed all the appropriate tasks for a particular level.

Most state machines you will write or work with will be considered Finite State Machines (FSMs), primarily because the number of potential states is just that, finite. However, there are another class of state machines, called transition systems, where that is not always the case.



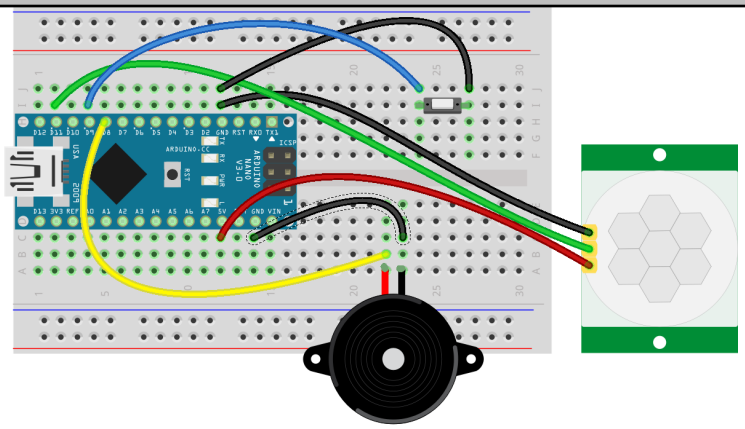
In this exercise, we are going to use your motion detector from the previous session. You can keep whatever sound or light alarms you had, but we'll be adding a button to disarm the system, and using the built-in LED to show if the system is armed or not.

When armed, the LED should be lit, and any motion will trigger your audible (and, optionally, visual) alarm.

When disarmed, the LED will be off, and the alarm will not sound.

© 2016-17 Dataseam

Experiment: Red State, Blue State, One State, Two State

Components	Wiring Diagram
✓Battery Pack ✓Breadboard ✓HC-SR501 motion ✓Button ✓speaker ✓optional LEDs	

Connection Instructions

Use the same motion + speaker setup you had from the previous section. Use a push button as you did in the bleeps and blorks session, connecting one side to ground and the other to whatever pin you prefer.

Sketch(es)	stately.ino

Analysis Questions

Do you think the software on the BIS001 microcontroller in the HC-SR501 is implemented using a state machine?
Can you think of any other real-world things that would be good state machine examples.
Do you need to hold the button in or push it a few times to go from ARMED to DISARMED after an alarm was triggered? Why?
Could that be corrected?

Programming Tasks

Modify the program such that the motion detection system operates in two modes: Armed and Disarmed.
When Armed the internal LED on pin 13 should be lit and any motion will trigger an audible alert.

When disarmed, the internal LED should not be lit, and no amount of motion will cause an alert.
Pushing the button toggles between Armed and Disarmed.

Optional, if time permits. Add a third mode.
In this mode 2 motions events with less than (approximately) 1/2 second between them need to be detected in order to trigger an alert.
This mode is indicated by a periodically flashing internal LED.

© 2016-17 Dataseam