State machines with the Arduino

Objectives

- Understand the concept of what state machines are and their purpose
- Understand how to implement a state machine in Arduino

State Machines

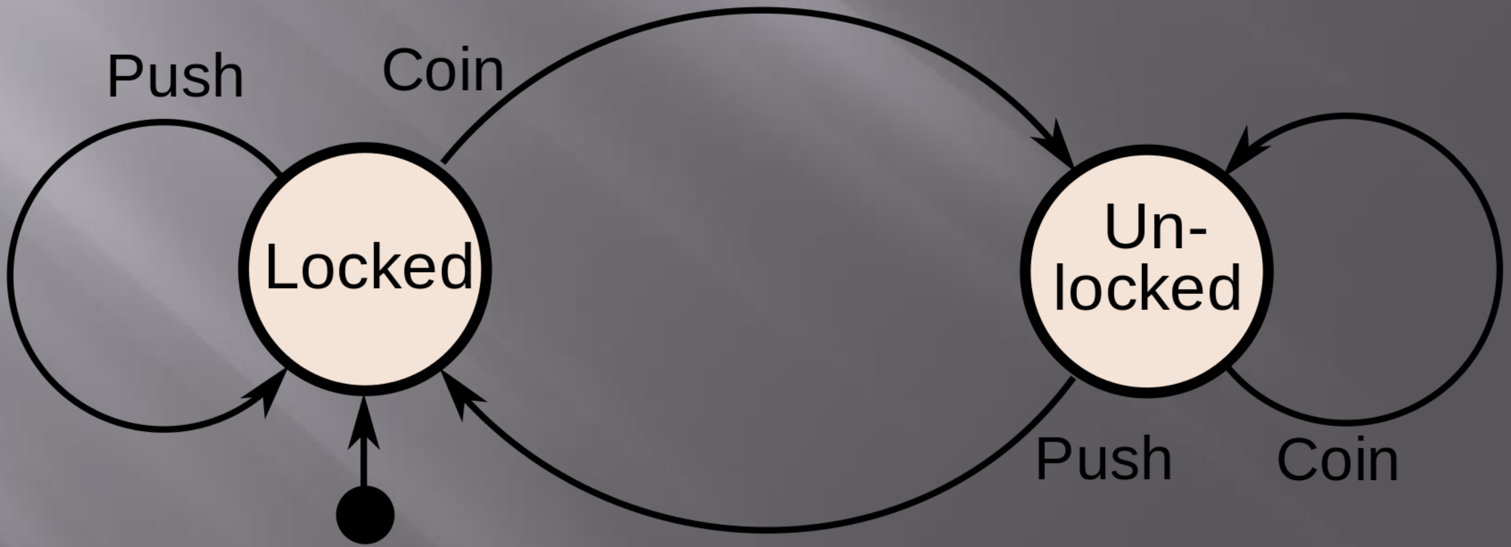
- In coding, a *State Machine*, or *Finite State Machine* is a program that has the following characteristics:
  - There are a “finite” number of *states* the program can be in.
  - The program can be in only <u>one</u> state at any time.
  - Inputs determine the next state the program will go to. Inputs can be switches, sensors, or timers for example.

Advantages of State Machines

- State machines eliminate the possibility of "jumping" from one state to another out of sequence or causing "glitches."
- They are structured in a way that makes code easier to implement and more robust.
- They are scalable and adaptable; in other words you can easily add more tasks or change the sequence without re-writing lots of code.
- They act immediately on inputs without any delays.

State machine diagrams

- A common way of visualizing state machines is using a *state diagram*. The following example is for a turnstile with two inputs and one output:



State machine examples

- State machines are used in systems that require high reliability and operate in a logical sequence

- Elevators
- Traffic lights
- Factory automation (i.e. assembly lines)

State machine coding

- Let's start with a simple example where we cycle three LED's (green, yellow and red) sequentially
- We start by defining the states with names:

```
// Define states

#define GREEN  1
#define YELLOW  2
#define RED   3
```

Note: In this case we use the #define directive rather than a variable. This basically substitutes the number on the right in the code for the names when it is compiled. It could also be done using variables that are defined as constants.

State machine coding

In this basic example, an *if statement* is used to turn on a green LED if STATE = GREEN. Notice that the end of the code will force the program into the YELLOW state after this state is completed.

```
if (STATE == GREEN)
 {
    digitalWrite (greenLED,HIGH);
    digitalWrite (yellowLED,LOW);
    digitalWrite (redLED,LOW);
    while (digitalRead (button) == 1);  // Wait for button to be pressed
    debounce();                         // short debounce
    STATE = YELLOW;                     // Advance to next state
}
```

State machine coding

The next state turns only the yellow LED on. What will the next state be?

```
if (STATE == YELLOW)
 {
    digitalWrite (greenLED,LOW);
    digitalWrite (yellowLED,HIGH);
    digitalWrite (redLED,LOW);
    while (digitalRead(button) == 1);  // Wait for button to be pressed
    debounce();                         // short debounce
    STATE = RED;                        // Advance to next state
}
```

State machine coding

In this state the red LED is turned on. The next state will return to state number 1, which is called GREEN.

```
if (STATE == RED)
 {

    digitalWrite (greenLED,LOW);
    digitalWrite (yellowLED,LOW);
    digitalWrite (redLED,HIGH);
    while (digitalRead(button) == 1);  // Wait for button to be pressed
    debounce();                        // short debounce
    STATE = GREEN;                     // Advance to next state
}
```

State machine coding

- This example may seem like a lot of work to just run LED's in sequence (as we have coded before) using a for loop.
- However, it would be very easy to change the sequence, or add more LED's.
- Also, we could add other functions inside each state (for example, look at another switch input to move to another state)

State machine coding

- Counters (often using a variable called "tick") can be used to cause delays during states, or automatically move from state to state.
- The main method used here is to implement a variable that increments quickly, for example every 10 milliseconds.
- This allows a switch or other input to be read frequently without using delays, so the system can respond quickly to external events.
- The following screen shows an example of how this is implemented.

Using a counter

```
delayTime = 5;                        // Set delay to 5, which will create a 5 second delay
void loop()
{
  tick++;                             // increment tick every 10 milliseconds
  if (tick == delayTime * 100)  // if tick is equal to delay time, go to next state
  {
    changeLIGHTS (STATE);  // Cycle through states
    tick = 0;                         // Reset tick
  }
delay(10);                            // Short delay for cycle time
}
```

In this case the states are changed in a function called changeLIGHTS()
We will walk through a specific example to see how this works.

References

Great reference: Look at turnstile example

https://en.wikipedia.org/wiki/Finite-state_machine