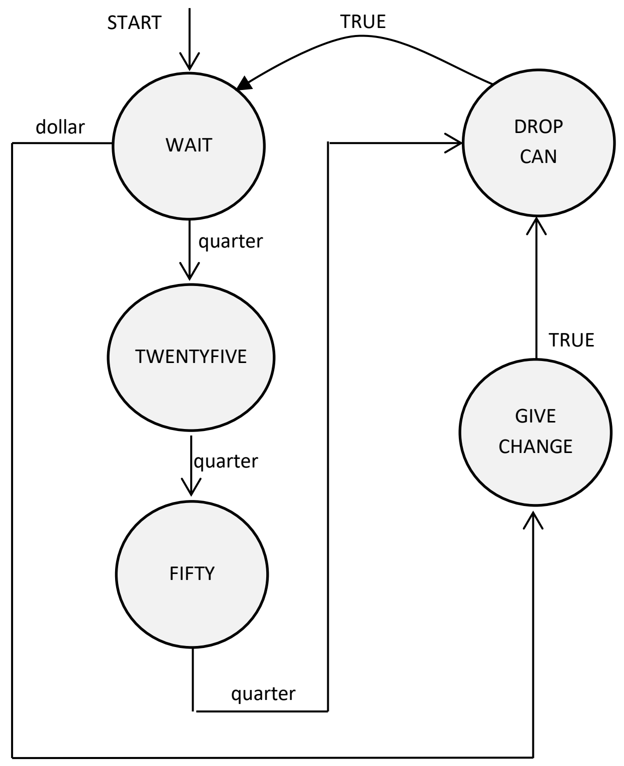
State machine with Arduino (Part 2)

The vending machine

In this part, we will simulate a simple vending machine. To make matters simple, we will agree that:

- This machine distributes bottles of water.
- Each bottle sells for 75¢.
- Only dollars and quarters are accepted.
- When a dollar is used to pay, a quarter will be returned for change.
- When a quarter has been inserted, dollars are not accepted anymore.

We will use 2 buttons: One to simulate dropping a quarter and another one to simulate dropping a dollar.
We will use two LEDs: One to simulate that the can has been dropped, and another one to simulate that the change has been given.



This machine will have 5 states:

WAIT :	Probe the two switches: If dollar: change state to GIVE CHANGE state If quarter: change state to TWENTYFIVE state
TWENTYFIVE:	Probe the quarter switch If quarter: change state to FIFTY state
FIFTY:	Probe the quarter switch If quarter: change state to DROP CAN state
GIVE CHANGE:	Flash LED Change state to DROP CAN state
DROP CAN:	Flash LED Change state to WAIT state

First, we define the state machine:

```
enum VendingMachineStates {WAIT, TWENTYFIVE, FIFTY, GIVE_CHANGE, DROP_CAN};
VendingMachineStates vmState = WAIT;

void vendingMachine() {  //We will complete it later
  switch (vmState) {
    case WAIT: { break: }
    case TWENTYFIVE { break: }
    case FIFTY: { break: }
    case GIVE_CHANGE: { break: }
    case DROP_CAN: { break: }
  }
}
```

We will use the debouncer:

```
#include <EdgeDebounceLite.h>
EdgeDebounceLite debounce;
```

We will need to read two switches. We will use the switch state machine that we created in Part 1. We will just modify it so that it can use more than one switch (changes are in blue).

```
enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState[2] = {IS_OPEN, IS_OPEN};
byte switchPin[2] = {10, 11};

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode[2] = {PULLUP, PULLUP};

void switchMachine(byte i) {
  byte pinIs = debounce.pin(switchPin[i]);
  if (switchMode[i] == PULLUP) pinIs = !pinIs;
  switch (switchState[i]) {
    case IS_OPEN:    { if(pinIs == HIGH) switchState[i] = IS_RISING;  break; }
    case IS_RISING:  {                   switchState[i] = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)  switchState[i] = IS_FALLING; break; }
    case IS_FALLING: {                   switchState[i] = IS_OPEN;    break; }
  }
}
```

We will create two functions to read the switches:

```
bool dollarInserted() {
  switchMachine(0);
  if (switchState[0] == IS_FALLING) return true;
  else                              return false;
}

bool quarterInserted() {
  switchMachine(1);
  if (switchState[1] == IS_FALLING) return true;
  else                              return false;
}
```

We will just quickly blink the LEDs so we can afford to use delay() for this.

```
byte ledPin[2] = {2, 3};

void blinkLed(byte i) {
  digitalWrite(ledPin[i], HIGH);
  delay(10);
  digitalWrite(ledPin[i], LOW);
}
```

We will create two functions to blink the LEDs

```
void giveChange() {
  blinkLed(0);
}

void dropCan() {
  blinkLed(1);
}
```

Now we can complete the vending machine code.

```
void vendingMachine() {
  switch (vmState) {
    case WAIT:        { if (dollarInserted())  vmState = GIVE_CHANGE;
                        if (quarterInserted()) vmState = TWENTYFIVE;   break: }
    case TWENTYFIVE   { if (quarterInserted()) vmState = FIFTY;        break: }
    case FIFTY:       { if (quarterInserted()) vmState = DROP_CAN;     break: }
    case GIVE_CHANGE: { giveChange();          vmState = DROP_CAN;     break: }
    case DROP_CAN:    { dropCan();             vmState = WAIT;         break: }
  }
}
```

Putting it all together with debounced switches gives:

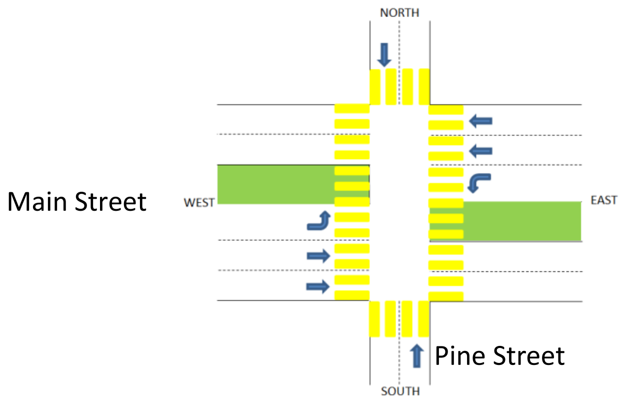
```
#include <EdgeDebounceLite.h>
EdgeDebounceLite debounce;

enum VendingMachineStates {WAIT, TWENTYFIVE, FIFTY, GIVE_CHANGE, DROP_CAN};
VendingMachineStates vmState = WAIT;

enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState[2] = {IS_OPEN, IS_OPEN};
byte switchPin[2] = {10, 11};

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode[2] = {PULLUP, PULLUP};

byte ledPin[2] = {2, 3};

void switchMachine(byte i) {
  byte pinIs = debounce.pin(switchPin[i]);
  if (switchMode[i] == PULLUP) pinIs = !pinIs;
  switch (switchState[i]) {
    case IS_OPEN:    { if(pinIs == HIGH) switchState[i] = IS_RISING;  break; }
    case IS_RISING:  {                   switchState[i] = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)  switchState[i] = IS_FALLING; break; }
    case IS_FALLING: {                   switchState[i] = IS_OPEN;    break; }
  }
}


bool dollarInserted() {
  switchMachine(0);
  if (switchState[0] == IS_FALLING) return true;
  else                              return false;
}


bool quarterInserted() {
  switchMachine(1);
  if (switchState[1] == IS_FALLING) return true;
  else                              return false;
}
void blinkLed(byte i) {
  digitalWrite(ledPin[i], HIGH);
  delay(10);
  digitalWrite(ledPin[i], LOW);
}
```

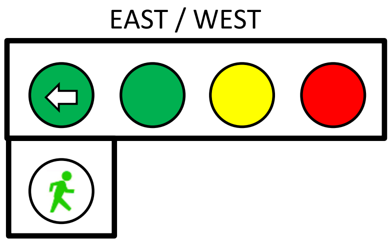
```
void giveChange() {
  blinkLed(0);
}

void dropCan() {
  blinkLed(1);
}
void vendingMachine() {
  switch (vmState) {
    case WAIT:        { if (dollarInserted())  vmState = GIVE_CHANGE;
                        if (quarterInserted()) vmState = TWENTYFIVE;   break; }
    case TWENTYFIVE:  { if (quarterInserted()) vmState = FIFTY;        break; }
    case FIFTY:       { if (quarterInserted()) vmState = DROP_CAN;     break; }
    case GIVE_CHANGE: { giveChange();          vmState = DROP_CAN;     break; }
    case DROP_CAN:    { dropCan();             vmState = WAIT;         break; }
  }
}

void setup() {
  for (int i = 0 ; i < 2 ; i++) pinMode(switchPin[i], INPUT_PULLUP);
  for (int i = 0 ; i < 2 ; i++) pinMode(ledPin[i], OUTPUT);
}
void loop() {
  vendingMachine();
}
```

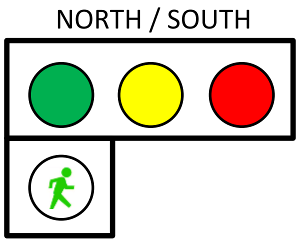
NEXT PAGE: THE TRAFFIC LIGHTS STATE MACHINE

The Traffic Lights

We will now take a look at an example with traffic lights. Sometown wants to add traffic lights on the corner of Main Street and Pine Street.







Description of the street lights

- Main Street is a two lines street and has a third line at intersections to allow cars to turn left. Its green light will be on for 20 seconds. The yellow light will be on for 3 seconds and then light will turn red.
- Pine Street is a secondary street. Its green light will be on for 10 seconds. The yellow light will be on for 3 seconds and then the light will turn red.
- After that Pine Street's light turns red, the cars on the Main Street will have their light on to turn left for 10 seconds. It will then blink for 3 seconds, and finally, the light will turn off.
- If a button is pressed by a pedestrian, he/she will get his/her walk light after the turn left light turns off. The walk light is on for 5 seconds, it will blink for 10 seconds and then it will turn off. Both walk lights will be activated.
- When a red light comes on, all red lights stay on for 2 seconds.

Hardware requirements

The LEDs

We need 8 LEDs

```
#define MAIN_G_PIN 2    //Main Street green LED pin
#define MAIN_Y_PIN 3    //Main Street yellow LED pin
#define MAIN_R_PIN 4    //Main Street red LED pin
#define PINE_G_PIN 5    //Pine Street green LED pin
#define PINE_Y_PIN 6    //Pine Street yellow LED pin
#define PINE_R_PIN 7    //Pine Street red LED pin
#define TURN_PIN   8    //Turn left LED pin
#define WALK_PIN   9    //Pedestrian LED pin
```

Two of those LEDs will be blinking at some point. We will blink at a 250 milliseconds rate (4 times per second).

```
#define BLINK_SPEED 250
```

We will use the blink without delay technique, so we need a chronometer.

```
unsigned long blinkChrono;
```

This is the same blink function that has been seen in the **millis** Tutorial, except for the fourth line that actually turns the pin on and off. It **writes** to the pin: NOT what it **reads** from the pin. Thanks to UKHeliBob from the Arduino Forum for this idea.

```
void blink(byte ID) {
  if (millis() - blinkChrono >= BLINK_SPEED) {
    blinkChrono = millis();
    digitalWrite(ID, !digitalRead(ID));
  }
}
```

The switch

The pedestrians will have four switches to call for a walk light. All four switches are connected in parallel, so we need only one pin to read them. We will use our Switch state machine from part 1.

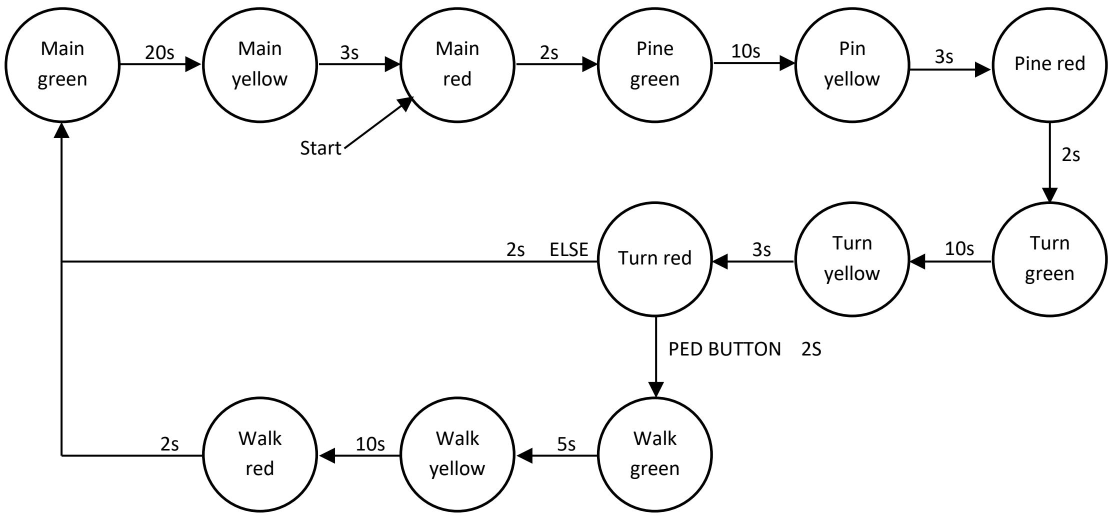
Of course, we will debounce the switches.

```
#include <EdgeDebounceLite.h>
EdgeDebounceLite debounce;
```

The global variable *pedestrians* is a flag that is set to true when the pin IS_FALLING (click). The Traffic Light State Machine will read this flag to decide if it will run the WALK sequence.

```
bool pedestrians = false;
byte switchPin = 10;

enum SwitchStates {IS_OPEN, IS_RISING, IS_CLOSED, IS_FALLING};
SwitchStates switchState = IS_OPEN;

enum SwitchModes {PULLUP, PULLDOWN};
SwitchModes switchMode = PULLUP;

void readSwitch() {
  byte pinIs = debounce.pin(switchPin);
  if (switchMode == PULLUP) pinIs = !pinIs;
  switch (switchState) {
    case IS_OPEN:    { if(pinIs == HIGH)   switchState = IS_RISING;  break; }
    case IS_RISING:  {                     switchState = IS_CLOSED;  break; }
    case IS_CLOSED:  { if(pinIs == LOW)    switchState = IS_FALLING; break; }
    case IS_FALLING: { pedestrians = true; switchState = IS_OPEN;    break; }
  }
}
```

The Traffic Light State Machine



```
enum TraficLights { MAIN_G, MAIN_Y, MAIN_R, PINE_G, PINE_Y, PINE_R,
                    TURN_G, TURN_Y, TURN_R, WALK_G, WALK_Y, WALK_R};
TraficLights statusTL = MAIN_R;
```

For the Main and Pine states, the algorithm is:

- We check if the time is up to jump to the next state. If so:
  - We reset the chronometer
  - We turn off the light from our state
  - We turn on the one for the other state
  - We change the machine’s state
- We jump out of the switch construct

For the Turn left and Walk states, we have to manage the On, Blink and Off cycles. When they are in a *green* state, we will write the pin HIGH and when they are in a *red* state, we will write the pin LOW. When they are in a *yellow* state, we call *blink* each time that the state is probed (before the *if* statement).

The last state that is special is when it is time to leave the TURN_R state. We have to check if the pedestrian button has been pressed in this cycle to decide if we are going to the MAIN_G state or the WALK_G state.

```
void trafficLights() {
  switch(statusTL) {
    case MAIN_G: { if (millis() - chrono >= 20000) {
                     chrono = millis();
                     digitalWrite(MAIN_G_PIN,  LOW);
                     digitalWrite(MAIN_Y_PIN, HIGH);
                     statusTL = MAIN_Y; }
                   break; }
    case MAIN_Y: { if (millis() - chrono >=  3000) {
                     chrono = millis();
                     digitalWrite(MAIN_Y_PIN, LOW);
                     digitalWrite(MAIN_R_PIN, HIGH);
                     statusTL = MAIN_R; }
                   break; }
    case MAIN_R: { if (millis() - chrono >=  2000) {
                     chrono = millis();
                     digitalWrite(PINE_R_PIN,  LOW);
                     digitalWrite(PINE_G_PIN, HIGH);
                     statusTL = PINE_G; }
                   break; }
    case PINE_G: { if (millis() - chrono >= 10000) {
                     chrono = millis();
                     digitalWrite(PINE_G_PIN,  LOW);
                     digitalWrite(PINE_Y_PIN, HIGH);
                     statusTL = PINE_Y; }
                   break; }
    case PINE_Y: { if (millis() - chrono >=  3000) {
                      chrono = millis();
                      digitalWrite(PINE_Y_PIN, LOW);
                      digitalWrite(PINE_R_PIN, HIGH);
                      statusTL = PINE_R; }
                   break; }
    case PINE_R: { if (millis() - chrono >=  2000) {
                     chrono = millis();
                     digitalWrite(TURN_PIN, HIGH);
                     statusTL = TURN_G; }
                   break; }
    case TURN_G: { if (millis() - chrono >= 10000) {
                     chrono = millis();
                     statusTL = TURN_Y; }
                   break; }
    case TURN_Y: { blink(TURN_PIN);
                   if (millis() - chrono >=  3000) {
                     chrono = millis();
                     digitalWrite(TURN_PIN, LOW);
                     statusTL = TURN_R; }
                   break; }
    case TURN_R: { if (millis() - chrono >=  2000) {
                     chrono = millis();
                     if (pedestrians) {
                        pedestrians = false;
                        digitalWrite(WALK_PIN, HIGH);
                        statusTL = WALK_G; }
                     else {
                       digitalWrite(MAIN_R_PIN,  LOW);
                       digitalWrite(MAIN_G_PIN, HIGH);
                       statusTL = MAIN_G; } }
                   break; }
    case WALK_G: { if (millis() - chrono >=  5000) {
                     chrono = millis();
                     statusTL = WALK_Y; }
                   break; }
    case WALK_Y:  { blink(WALK_PIN);
                    if (millis() - chrono >= 10000) {
                      chrono = millis();
                      digitalWrite(WALK_PIN, LOW);
                      statusTL = WALK_R; }
                     break; }
    case WALK_R:  { if (millis() - chrono >=  2000) {
                      chrono = millis();
                      digitalWrite(MAIN_R_PIN,  LOW);
                      digitalWrite(MAIN_G_PIN, HIGH);
                      statusTL = MAIN_G; }
                  break; }
  }
}
```

The setup and the loop

We will set the pin modes for all the LEDs and the switch. We will also make sure that the LEDs are appropriate for the start state: All the LEDs are off except for the red Main Street LED and the red Pine Street LED.

```
void setup() {
  for (byte i =2 ; i <= 9 ; i++) { pinMode(i, OUTPUT); digitalWrite(i, LOW); }
  pinMode(switchPin, INPUT_PULLUP);
  digitalWrite(MAIN_R_PIN, HIGH);
  digitalWrite(PINE_R_PIN, HIGH);
}
```

The loop only has to read the switch and call the Traffic Light State Machine.

```
void loop() {
  readSwitch();
  trafficLights();
}
```