Brendan Castro
Mackenzie Larson
Ethan Makela
CS 454 Theory of Computation
Dr. Ravikumar

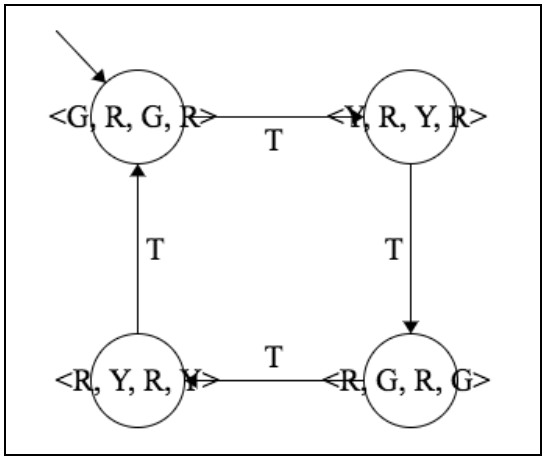
Final Project: Traffic Signal Embedded System

Problem Statement

Design an embedded system for a problem like traffic control in which the red/ green lights are turned on and off in an intersection. The idea is to implement the system in hardware (using microcontroller or FPGA). At the minimum, you are to design the finite automaton and then implement the state machine using logic design and flip-flop. You can then use an FPGA or microcontroller to implement the solution in hardware.

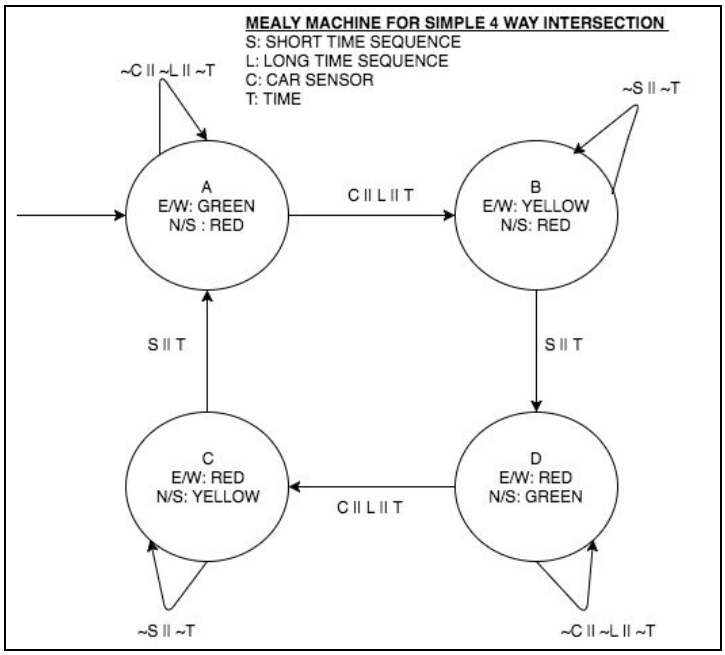
Logical Design

For the final implementation of this project we decided to create a replica of a four way intersection and transitions between states would be based on time. We came up with several designs, each increasing in complexity and efficiency. Our interval to transition between states was time (longer time while the light was green or red but shorter time when coming out of a yellow light state). We started small and built larger FSM's (finite state machine) off our basis which was a simple four way intersection that only had straight lights. By the transition of time between the states, there were only four states that the basis light could be in. This is the least optimized light as not as many vehicles could be on the road at the same time.
The following is an illustration of our most basic FSM, without vehicles, and transitions between states based on time.



Mealy Finite State Machine with Car Sensors

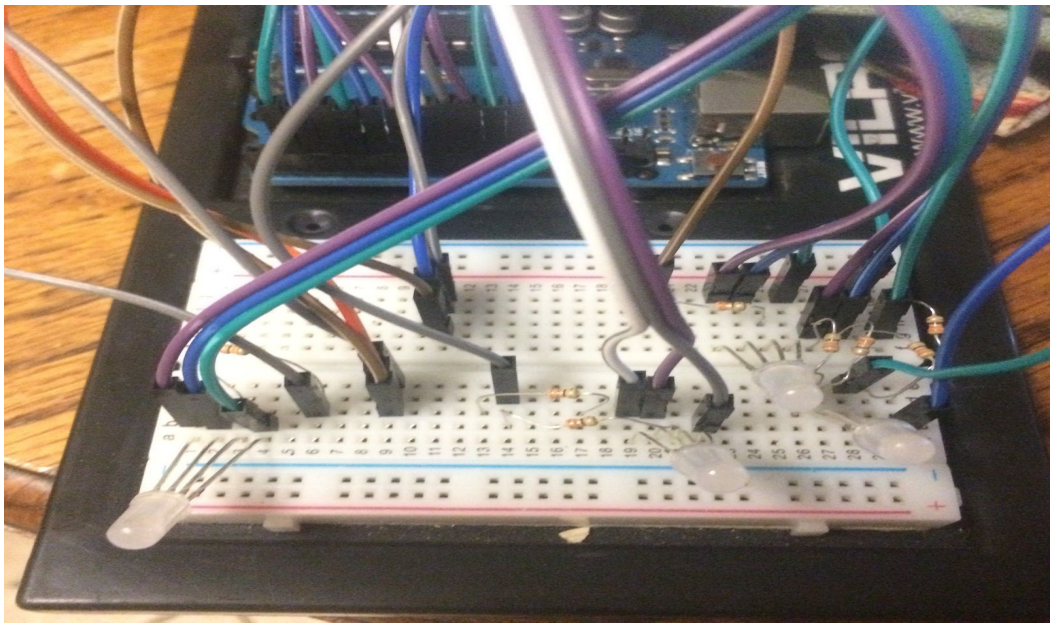
Our next finite state machine we decided to incorporate car sensors. For this we used the format of a mealy machine (a machine whose output values are determined both by its current state and the current inputs) to test how the states would change based on a car arriving. While we did not have the tools to do this in our hardware implementation, we believed it was important to none the less research and test. In the following diagram, states are changed based on either time running out (timer is reset, there are short and long timers based on yellow lights and longer state lights) or a car arriving at a different light thus beginning a trigger that would change the state of the lights to favor the car(s) that arrived else where.



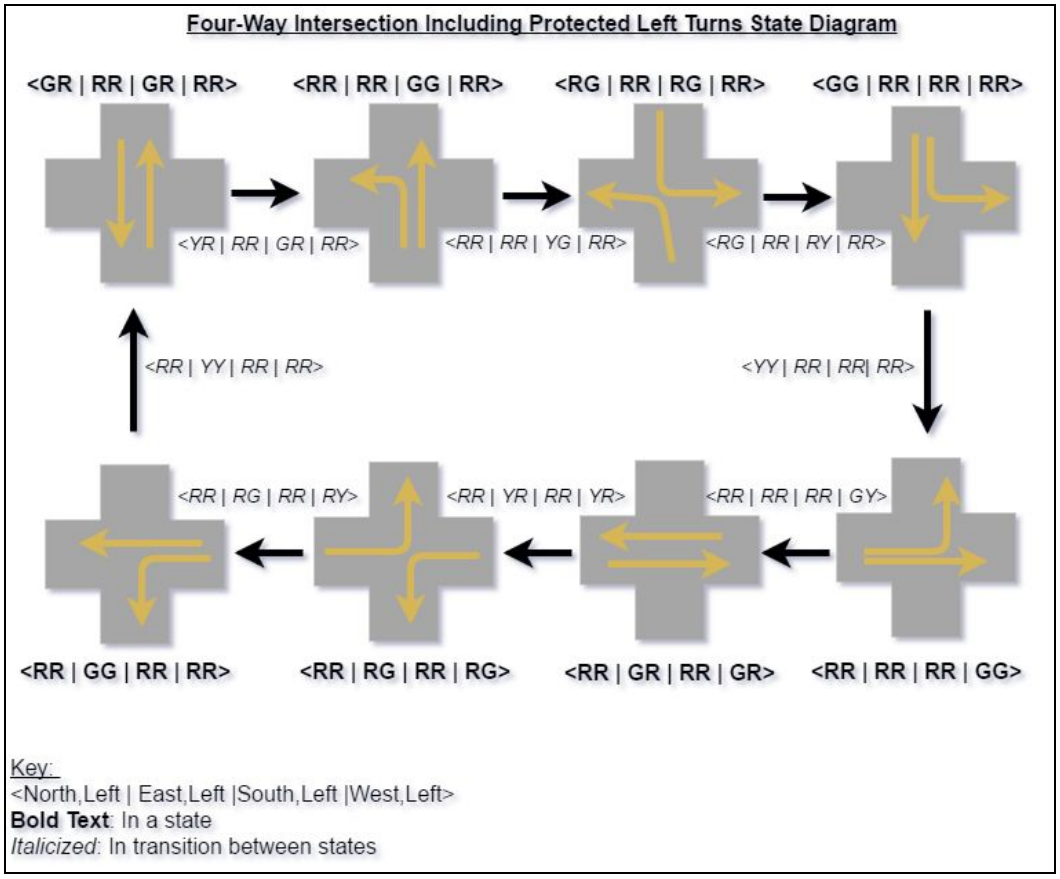
Hardware Implementation

To implement the traffic light in hardware we used an Arduino Uno hooked up to a breadboard where we used four multi-colored LED lights to represent the traffic directions. We chose to only use four lights due to the restriction on inputs the Arduino Uno has. Our Arduino only had 13 inputs; each light takes three input wires (one for red, green and blue) so therefor we could only use four lights. To be able to have traffic be able to flow in four directions with protected left, our team came up with a coloring scheme. We based our coloring scheme on having a straight light as well as a turn light ending up needing nine colors: RR- red, GG- violet, YY- orange, RG- blue, GR- green, YR- Yellow, RY- white, GY- Turquoise (The first letter is the straight light and the second is the turn lane). Once the DFA was made coding up the Arduino is simple, although our team encountered many difficulties with the multi-colored LEDs. We found that the lights were not getting equal voltages throughout the breadboard. The first light was receiving much more than each of the following lights. We tried to counter this by adding resistors but the resistors alone were not enough.

Eventually we came to the conclusion that we needed to pair our east and west lights along with our north and south and ground them separately (since the colors for those lights would be similar throughout the simulation). This trick fixed the last three lights but had no effect on the first light, which was still receiving too much power. To counter this we connected resistors in between each of the inputs of the first light which lowered the voltage by a large amount. By doing this we also lowered the voltage of the other three lights, so a trade off between too much power on the first light and not enough to run the other three began. In the end we needed a resistor per input for the first light along with two resistors in parallel for each of the pairs of lights. This did not fix the entire problem with the LEDs, although it got it to a place where we could play with lights individual RGB values to get somewhat similar light colors (other than red, green, and blue). Eventually we found individual RGB values for nine colors on each light.

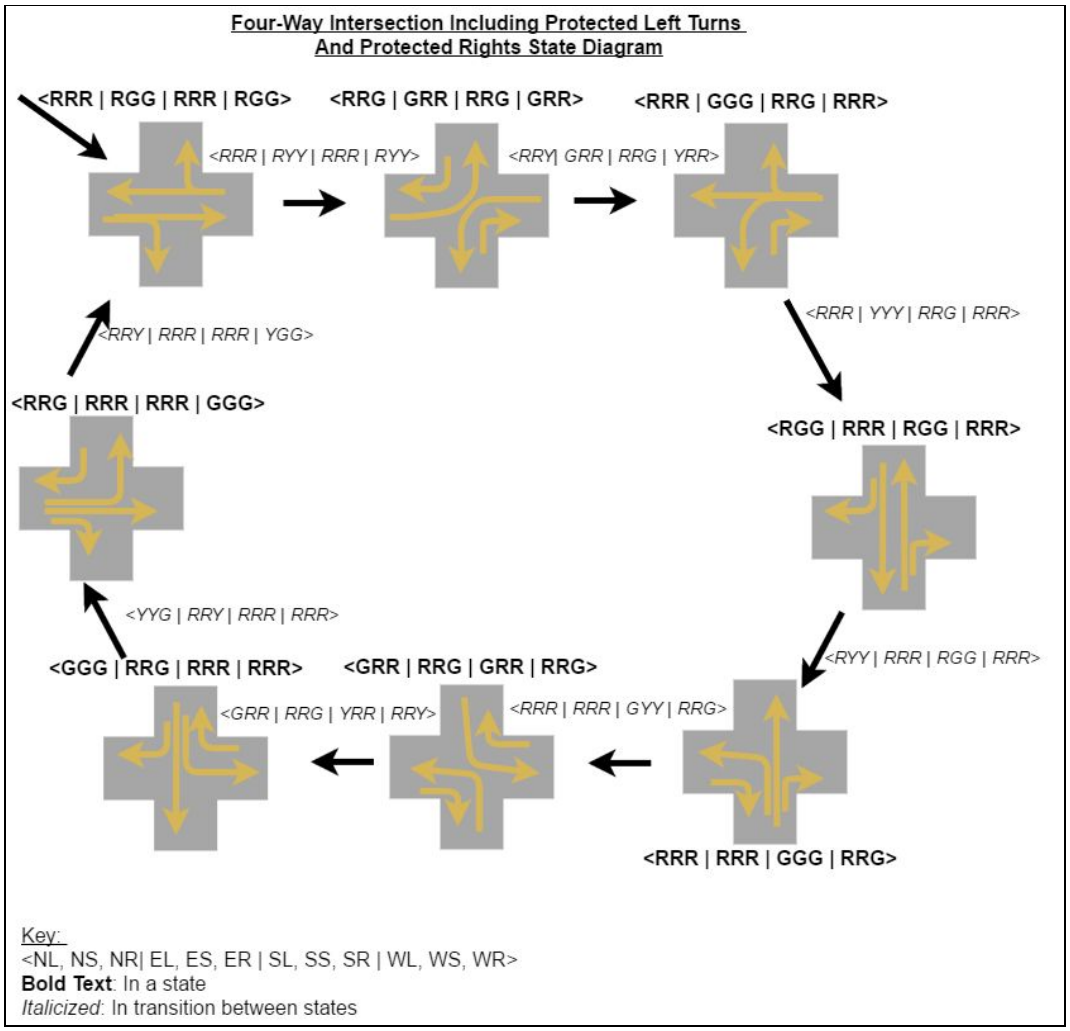


The finite state machine that our arduino implementation is based on is below. It is a continuous cycle based on time as the transition.



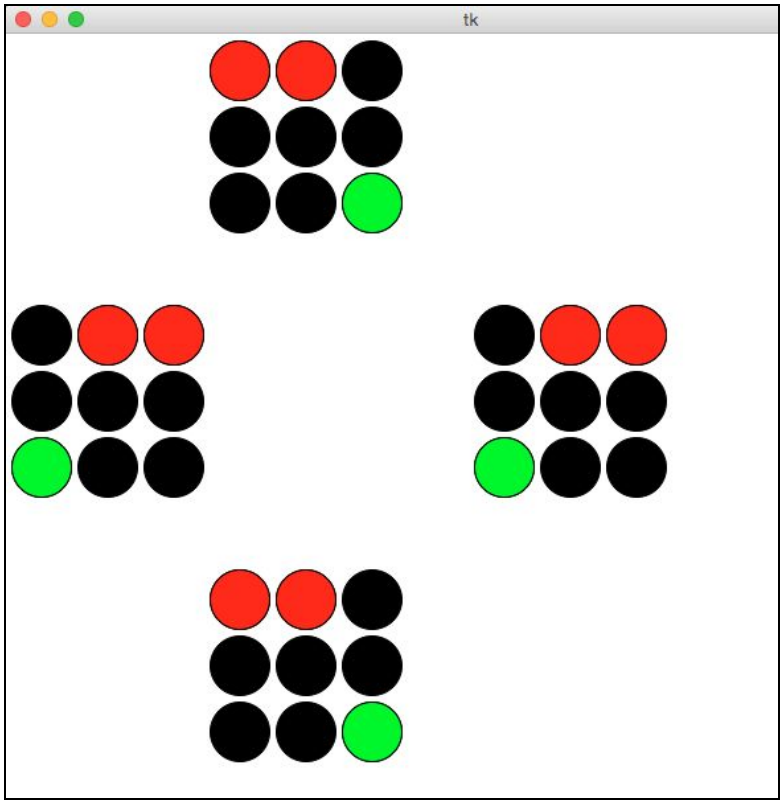
Software Implementation

In addition to the hardware implementation, we implemented a software version of the project using python & the tkinter library. We implemented a software version of the project as our breadboard had limited space for lights and we wanted to demonstrate a complete four-way intersection with three sets of lights. Our software implementation is the most optimized a traffic light can be for a four way intersection as it allows for the most traffic to be moving on the road at once. It incorporates left hand turns, straights, and protected right hand turns. The way our light system works is based on the finite state machine below (main states are documented with bolded text and an image to visualize the state, in between states are represented by italicised letters).



The following image is a screen capture of our python program. The lights are organized based on direction (North, South, East, West) and within each grouping of lights by left hand, straight and right hand lights. We wanted this display on the hardware implementation of this project but

it was difficult as we were limited by our bread board and light colors. The code is organized by states that loop based on time and a new state is generated based on its successor.



Conclusion

Overall, this project was a great learning experience to demonstrate how automata are applied to real world applications. It was interesting to see how to apply a finite state machine to a hardware application, something we have had very limited experience with. It was difficult at first to gather supplies and to really grasp what we needed to complete this project, but we now have a stronger understanding of just how powerful microcontrollers can be! It was especially interesting to see how such a seemingly trivial problem can be so challenging. This was a fitting final project to really bring together concepts learned in theory of computation.

The following is the code of our software implementation:

```
import time
from time import sleep
from tkinter import *
tk=Tk()
win=Canvas(tk, width=700, height=700)
win.pack()
#functions

#north lights STRAIGHT
red=win.create_oval(205,5,250,50, fill="black")
amber=win.create_oval(205,55,250,100, fill ="black")
green=win.create_oval(205,105,250,150, fill="black")
#NORTH LIGHTS LEFT
red=win.create_oval(155,5,200,50, fill="black")
amber=win.create_oval(155,55,200,100, fill ="black")
green=win.create_oval(155,105,200,150, fill="black")
#NORTH LIGHTS RIGHT
red=win.create_oval(255,5,300,50, fill="black")
amber=win.create_oval(255,55,300,100, fill ="black")
green=win.create_oval(255,105,300,150, fill="black")


#west lights STRAIGHT
red=win.create_oval(55,205,100,250, fill="black")
yellow=win.create_oval(55,255,100,300, fill="black")
green=win.create_oval(55,305,100,350, fill="black")
#west lights left
red=win.create_oval(5,205,50,250, fill="black")
yellow=win.create_oval(5,255,50,300,fill="black")
green=win.create_oval(5,305,50,350, fill="black")
#WEST LIGHTS PROTECTED RIGHT
red=win.create_oval(105,205,150,250, fill="black")
yellow=win.create_oval(105,255,150,300, fill="black")
green=win.create_oval(105,305,150,350, fill="black")

#south lights STRAIGHT
red=win.create_oval(205,405,250,450, fill="black")
amber=win.create_oval(205,455,250,500, fill ="black")
green=win.create_oval(205,505,250,550, fill="black")
```

```
#south lights left
red=win.create_oval(155,405,200,450, fill="black")
amber=win.create_oval(155,455,200,500, fill ="black")
green=win.create_oval(155,505,200,550, fill="black")
#SOUTH LIGHTS PROTECTED RIGHT
red=win.create_oval(255,405,300,450, fill="black")
amber=win.create_oval(255,455,300,500, fill ="black")
green=win.create_oval(255,505,300,550, fill="black")

#east lights STRAIGHT
red=win.create_oval(405,205,450,250, fill="black")
yellow=win.create_oval(405,255,450,300, fill="black")
green=win.create_oval(405,305,450,350, fill="black")
#east lights left
red=win.create_oval(355,205,400,250, fill="black")
yellow=win.create_oval(355,255,400,300, fill="black")
green=win.create_oval(355,305,400,350, fill="black")
#EAST LIGHTS PROTECTED RIGHT
red=win.create_oval(455,205,500,250, fill="black")
yellow=win.create_oval(455,255,500,300, fill="black")
green=win.create_oval(455,305,500,350, fill="black")

class trafficLights:
    def state1():
        #<NLR | ELR | SLR | WLR>
        #<RRR | GRG | RRR | GRG>

        win.delete("yellowNR")
        win.delete("redER")
        win.delete("yellowWL")
        win.delete("redE")

        #north and south STRAIGHT red
        redN=win.create_oval(205,5,250,50, fill="red", tags="redN")
        redS=win.create_oval(205,405,250,450, fill="red",tags="redS")
        #north and south LEFT red
        redNL=win.create_oval(155,5,200,50, fill="red", tags="redNL")
        redSL=win.create_oval(155,405,200,450, fill="red",tags="redSL")
        #east and west STRAIGHT green
        greenE=win.create_oval(405,305,450,350, fill="green",tags="greenE")
        greenW=win.create_oval(55,305,100,350, fill="green", tags="greenW")

        #EAST AND WEST RIGHT GREEN
```

```
    greenER=win.create_oval(455,305,500,350, fill="green",tags="greenER")
    greenWR=win.create_oval(105,305,150,350, fill="green",tags="greenWR")

    #NORTH AND SOUTH PR RED
    redNR=win.create_oval(255,5,300,50,  fill="red",tags="redNR")
    redSR= win.create_oval(255,405,300,450, fill="red",tags="redSR")

    #east and west LEFT red
    redEL=win.create_oval(355,205,400,250, fill="red",tags="redEL")
    redWL=win.create_oval(5,205,50,250, fill="red",tags="redWL")

    tk.update()
    sleep(10)


def state2():
    #<NLR | ELR | SLR | WLR>
    #<RRR | YRY | RRR |YRY>

    #DELETE EAST AND WEST STRAIGHT GREENS
    win.delete("greenE")
    win.delete("greenW")
    win.delete("greenER")
    win.delete("greenWR")

    #east and west STRAIGHT yellow
    yellowE=win.create_oval(405,255,450,300, fill="orange",tags="yellowE")
    yellowW=win.create_oval(55,255,100,300, fill="orange",tags="yellowW")

    yellowER=win.create_oval(455,255,500,300, fill="orange",tags="yellowER")
    yellowWR=win.create_oval(105,255,150,300, fill="orange",tags="yellowWR")

    tk.update()
    sleep(5)


def state3():
    #<NLR | ELR | SLR | WLR>
    #<RRG | RGR | RRG | RGR>

    #delete east and west straight yellows & EAST AND WEST REDS
    win.delete("yellowE")
    win.delete("yellowW")
```

```
    win.delete("yellowER")
    win.delete("yellowWR")
    win.delete("redNR")
    win.delete("redSR")
    win.delete("redEL")
    win.delete("redWL")

    #MAKE EAST AND WEST LEFTS GREEN
    greenEL=win.create_oval(355,305,400,350, fill="green",tags="greenEL")
    greenWL=win.create_oval(5,305,50,350, fill="green", tags="greenWL")

    #make east and west straights red
    redE=win.create_oval(405,205,450,250, fill="red", tags="redE")
    redW= win.create_oval(55,205,100,250, fill="red", tags="redW")



    redER= win.create_oval(455,205,500,250, fill="red",tags="redER")
    redWR=win.create_oval( 105,205,150,250, fill="red",tags="redWR")

    greenNR=win.create_oval(255,105,300,150, fill="green",tags="greenNR")
    greenSR=win.create_oval(255,505,300,550, fill="green",tags="greenSR")



    tk.update()
    sleep(10)

def state4():
    #<NLR | ELR | SLR | WLR>
    #<RRY | RGR | RRG | RYR>

    #delete west green LEFT
    win.delete("greenWL")
    win.delete("greenNR")

    #make west left yellow
    yellowWL=win.create_oval(5,255,50,300, fill="orange",tags="yellowWL")
    yellowNR=win.create_oval( 255,55,300,100,fill="orange",tags="yellowNR")

    tk.update()
    sleep(5)

def state5():
    #<NLR | ELR | SLR | WLR>
```

```
    #<RRR | GGG | RRG | RRR>
    #delete west left yellow & EAST RED
    win.delete("yellowWL")
    win.delete("yellowNR")
    win.delete("redE")
    win.delete("redER")

    #make east straight green & west left red
    greenER= win.create_oval(455,305,500,350, fill="green",tags="greenER")
    greenE=win.create_oval(405,305,450,350, fill="green",tags="greenE")
    redWL=win.create_oval(5,205,50,250, fill="red",tags="redWL")

    redNR=win.create_oval(255,5,300,50,  fill="red",tags="redNR")

    tk.update()
    sleep(10)

def state6():
    #<NLR | ELR | SLR | WLR>
    #<RRR | YYY | RRG | RRR>

    #remove east straight and east left greens
    win.delete("greenE")
    win.delete("greenEL")
    win.delete("greenER")

    #make east straight and east left yellow
    yellowE=win.create_oval(405,255,450,300, fill="orange",tags="yellowE")
    yellowEL=win.create_oval(355,255,400,300, fill="orange",tags="yellowEL")
    yellowER=win.create_oval( 455,255,500,300, fill="orange",tags="yellowER")

    tk.update()
    sleep(5)

def state7():
    #<GRG | RRR | GRG | RRR>

    win.delete("yellowE")
    win.delete("yellowEL")
    win.delete("yellowER")
    win.delete("redNR")
    win.delete("redN")
    win.delete("redS")
```

```
    #MAKE BOTH EAST STRAIGHT AND EAST LEFT RED
    redE=win.create_oval(405,205,450,250, fill="red", tags="redE")
    redEL=win.create_oval(355,205,400,250, fill="red", tags="redEL")
    redER= win.create_oval(455,205,500,250, fill="red",tags="redER")

    #MAKE NORTH AND SOUTH STRAIGHT GREEN
    greenN=win.create_oval(205,105,250,150,fill="green",tags="greenN")
    greenS= win.create_oval( 205,505,250,550, fill="green",tags="greenS")
    greenNR=win.create_oval(255,105,300,150, fill="green",tags="greenNR")

    tk.update()
    sleep(10)

def state8():
    #<NLR | ELR | SLR | WLR>
    #<YRY | RRR | GRG | RRR>

    win.delete("greenN")
    win.delete("greenNR")
    yellowN= win.create_oval(205,55,250,100,fill="yellow",tags="yellowN")
    yellowNR= win.create_oval( 255,55,300,100,fill="orange",tags="yellowNR")
    tk.update()
    sleep(5)

def state9():
    #<NLR | ELR | SLR | WLR>
    #<RRR | RRR | GGG | RRG>

    win.delete("yellowN")
    win.delete("redSL")
    win.delete("yellowNR")
    win.delete("redWR")

    greenWR= win.create_oval(105,305,150,350, fill="green",tags="greenWR")
    redNR=win.create_oval(255,5,300,50,  fill="red",tags="redNR")
    greenSL=win.create_oval(155,505,200,550, fill="green", tags="greenSL")
    redN= win.create_oval(205,5,250,50, fill="red", tags="redN")


    tk.update()
    sleep(10)
```

```
def state10():
    #<NLR | ELR | SLR | WLR>
    #<RRR | RRR | YGY | RRG>

    win.delete("greenSR")
    win.delete("greenS")
    yellowSR= win.create_oval(255,455,300,500, fill="orange",tags="yellowSR")
    yellowS= win.create_oval(205,455,250,500,fill="yellow",tags="yellowS")

    tk.update()
    sleep(5)

def state11():
    #<NLR | ELR | SLR | WLR>
    #<RGR | RRG | RGR | RRG>


    win.delete("yellowS")
    win.delete("yellowSR")
    win.delete("redNL")
    win.delete("redER")

    greenER= win.create_oval(455,305,500,350, fill="green",tags="greenER")
    #greenNR= win.create_oval(255,105,300,150, fill="green",tags="greenNR")
    greenNL= win.create_oval(155,105,200,150,fill="green",tags="greenNL")
    redS= win.create_oval(205,405,250,450, fill="red",tags="redS")
    redSR = win.create_oval(255,405,300,450, fill="red",tags="redSR")

    tk.update()
    sleep(10)

def state12():
    #<NLR | ELR | SLR | WLR>
    #<RGG | RRG | RYY | RRY>

    win.delete("greenSL")
    win.delete("greenWR")

    yellowWR= win.create_oval(105,255,150,300, fill="orange",tags="yellowWR")
    yellowSL= win.create_oval(155,455,200,500, fill="yellow",tags="yellowSL")

    tk.update()
    sleep(5)
```

```
def state13():
    #<NLR | ELR | SLR | WLR>
    #<GGG | RRG | RRR | RRR>

    win.delete("redN")
    win.delete("redNR")
    win.delete("yellowWR")
    win.delete("yellowSL")
    win.delete("yellowSR")

    redWR= win.create_oval( 105,205,150,250, fill="red",tags="redWR")
    redSR= win.create_oval(255,405,300,450, fill="red",tags="redSR")
    redSL=win.create_oval(155,405,200,450, fill="red",tags="redSL")
    greenN=win.create_oval(205,105,250,150,fill="green",tags="greenN")
    greenNR= win.create_oval(255,105,300,150, fill="green",tags="greenNR")

    tk.update()
    sleep(10)

def state14():
    #<NLR | ELR | SLR | WLR>
    #<YYG | RRY | RRR| RRR>

    win.delete("greenER")
    win.delete("greenN")
    win.delete("greenNL")

    yellowER= win.create_oval(455,255,500,300, fill="orange",tags="yellowER")
    yellowN= win.create_oval(205,55,250,100,fill="yellow",tags="yellowN")
    yellowNL= win.create_oval(155,55,200,100,fill="yellow",tags="yellowNL")

    tk.update()
    sleep(5)

def state15():
    #<NLR | ELR | SLR | WLR>
    #<RRG | RRR | RRR| GGG>

    win.delete("yellowER")
    win.delete("yellowN")
    win.delete("yellowNL")
    win.delete("redW")
```

```
        win.delete("redWL")
        win.delete("redWR")
        win.delete("redNR")

        greenWR= win.create_oval(105,305,150,350, fill="green",tags="greenWR")
        redER= win.create_oval(455,205,500,250, fill="red",tags="redER")
        redN=win.create_oval(205,5,250,50, fill="red", tags="redN")
        redNL=win.create_oval(155,5,200,50, fill="red", tags="redNL")
        greenW= win.create_oval(55,305,100,350, fill="green", tags="greenW")
        greenWL= win.create_oval(5,305,50,350, fill="green", tags="greenWL")

        tk.update()
        sleep(10)

    def state16():
        #<NLR | ELR | SLR | WLR>
        #<RRY | RRR | RRR | GYG>

        win.delete("greenWL")
        win.delete("greenNR")

        yellowNR= win.create_oval( 255,55,300,100,fill="orange",tags="yellowNR")
        yellowWL= win.create_oval(5,255,50,300, fill="yellow", tags="yellowWL")
        tk.update()
        sleep(5)

lights = trafficLights

for x in range(200):
    lights.state1()
    lights.state2()
    lights.state3()
    lights.state4()
    lights.state5()
    lights.state6()
    lights.state7()
    lights.state8()
    lights.state9()
    lights.state10()
    lights.state11()
    lights.state12()
    lights.state13()
    lights.state14()
```

```
    lights.state15()
    lights.state16()

#root.mainloop()

tk.mainloop()
```

HardWare Code

```
 //Cs 454
 //Final Project
 //Ethan Makela, Brendan Castro, Mackenzie Larson


  int northRed = 2;
  int northGreen = 3;
  int northBlue = 4;
  int eastRed = 5;
  int eastGreen = 6;
  int eastBlue = 7;
  int southBlue = 8;
  int southRed = 9;
  int southGreen = 10;
  int westRed = 11;
  int westGreen = 12;
  int westBlue = 13;
  int red = 255;
  int green = 255;
  int blue = 255;

void setup() {
  // put your setup code here, to run once:
  pinMode(northGreen, OUTPUT);
  pinMode(northBlue, OUTPUT);
  pinMode(northRed, OUTPUT);

  pinMode(eastGreen, OUTPUT);
  pinMode(eastBlue, OUTPUT);
  pinMode(eastRed, OUTPUT);
  pinMode(southGreen, OUTPUT);
  pinMode(southBlue, OUTPUT);
  pinMode(southRed, OUTPUT);
  pinMode(westGreen, OUTPUT);
```

```
  pinMode(westBlue, OUTPUT);
  pinMode(westRed, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  state1();
  delay(5000);
  state2();
  delay(3000);
  state3();
  delay(5000);
  state4();
  delay(3000);
  state5();
  delay(5000);
  state6();
  delay(3000);
  state7();
  delay(5000);
  state8();
  delay(3000);
  state9();
  delay(5000);
  state10();
  delay(3000);
  state11();
  delay(5000);
  state12();
  delay(3000);
  state13();
  delay(5000);
  state14();
  delay(3000);
  state15();
  delay(5000);
  state16();
  delay(3000);

}

void clearAll(){
  analogWrite(northRed, 0);
```

```
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  analogWrite(eastRed, 0);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  analogWrite(southRed, 0);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  analogWrite(westRed, 0);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);
}


void state1(){
 clearAll();
 //North green
  analogWrite(northRed, 0);
  analogWrite(northGreen, 255);
  analogWrite(northBlue, 0);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south green
  analogWrite(southRed, 0);
  analogWrite(southGreen, 255);
  analogWrite(southBlue, 0);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state2(){
  clearAll();
 //North yellow
  analogWrite(northRed, 160);
  analogWrite(northGreen, 255);
  analogWrite(northBlue, 0);
  //east red
```

```
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south green
  analogWrite(southRed, 0);
  analogWrite(southGreen, 255);
  analogWrite(southBlue, 0);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state3(){
  clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south violete
  analogWrite(southRed, 160);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 190);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state4(){
clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
```

```
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south pink
  analogWrite(southRed, 228);
  analogWrite(southGreen, 102);
  analogWrite(southBlue, 204);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);

}

void state5(){
  clearAll();
 //North blue
  analogWrite(northRed, 0);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 152);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south blue
  analogWrite(southRed, 0);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 152);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state6(){
  clearAll();
 //North blue
  analogWrite(northRed, 0);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 152);
```

```
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south white
  analogWrite(southRed, 150);
  analogWrite(southGreen, 255);
  analogWrite(southBlue, 255);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state7(){
  clearAll();
 //North violete
  analogWrite(northRed, 160);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 190);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state8(){
  clearAll();
 //North orange
  analogWrite(northRed, 185);
  analogWrite(northGreen, 65);
```

```
  analogWrite(northBlue, 120);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

 void state9(){
  clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west violet
  analogWrite(westRed, 160);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 190);


}

 void state10(){
  clearAll();
 //Northred
  analogWrite(northRed, 255);
```

```
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east red
  analogWrite(eastRed, 255);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 0);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west turquoise
  analogWrite(westRed, 72);
  analogWrite(westGreen, 209);
  analogWrite(westBlue, 204);


}

void state11(){
  clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east green
  analogWrite(eastRed, 0);
  analogWrite(eastGreen, 255);
  analogWrite(eastBlue, 0);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west green
  analogWrite(westRed, 0);
  analogWrite(westGreen, 255);
  analogWrite(westBlue, 0);


}

void state12(){
  clearAll();
 //North red
```

```
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east yellow
  analogWrite(eastRed, 160);
  analogWrite(eastGreen, 255);
  analogWrite(eastBlue, 0);
  //southred
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west
  analogWrite(westRed, 160);
  analogWrite(westGreen, 255);
  analogWrite(westBlue, 0);


}

void state13(){
  clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east blue
  analogWrite(eastRed, 0);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 152);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west blue
  analogWrite(westRed, 0);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 152);


}

void state14(){
  clearAll();
```

```
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east blue
  analogWrite(eastRed, 0);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 152);
  //south white
  analogWrite(southRed, 150);
  analogWrite(southGreen, 255);
  analogWrite(southBlue, 255);
  //west white
  analogWrite(westRed, 150);
  analogWrite(westGreen, 255);
  analogWrite(westBlue, 255);


}

void state15(){
  clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east violet
  analogWrite(eastRed, 160);
  analogWrite(eastGreen, 0);
  analogWrite(eastBlue, 190);
  //southred
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}

void state16(){
```

```
  clearAll();
 //North red
  analogWrite(northRed, 255);
  analogWrite(northGreen, 0);
  analogWrite(northBlue, 0);
  //east orange
  analogWrite(eastRed, 115);
  analogWrite(eastGreen, 145);
  analogWrite(eastBlue, 120);
  //south red
  analogWrite(southRed, 255);
  analogWrite(southGreen, 0);
  analogWrite(southBlue, 0);
  //west red
  analogWrite(westRed, 255);
  analogWrite(westGreen, 0);
  analogWrite(westBlue, 0);


}
```