HSHL

ELE WS 25/26

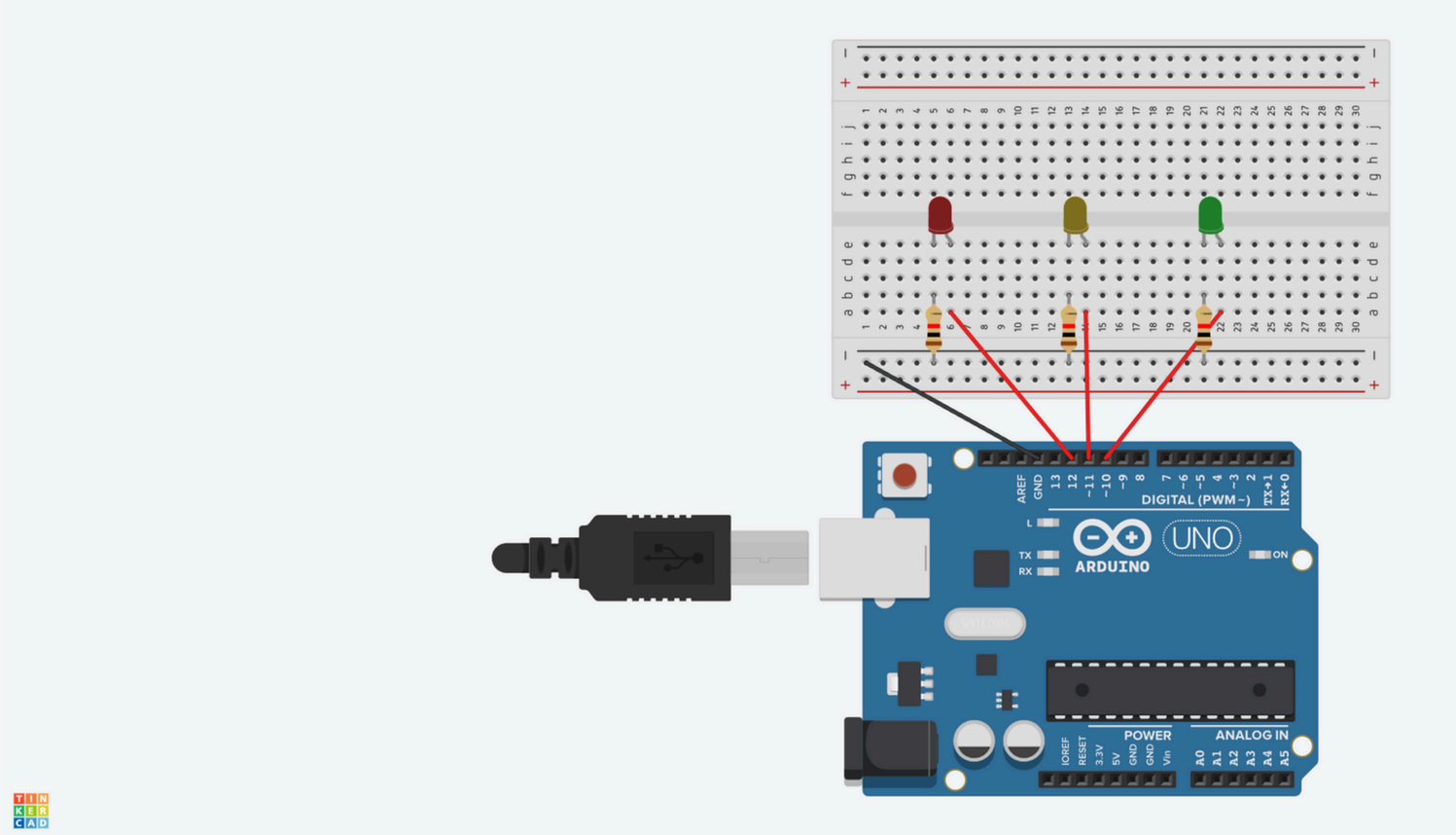
# TASK 1-5 SUMMARY REPORT

## EMBEDDED SYSTEMS

MOEEZ MUFTI
MUHAMMAD ALI
TALHA KHAN

# Task 1






TinkerCAD Simulation

# Task 1

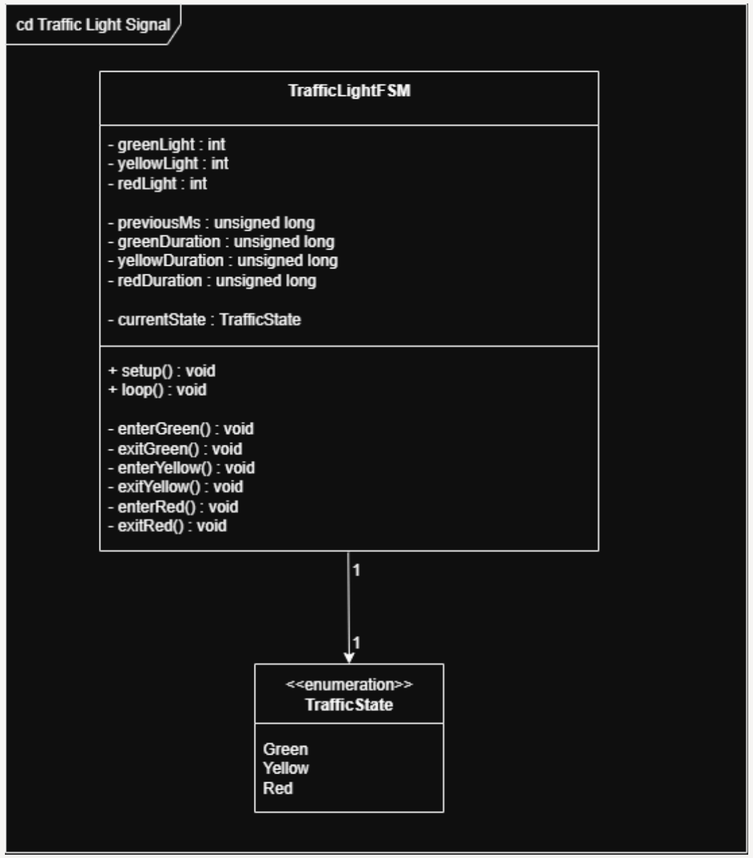


```
// Pin delegations
const int greenLight = 2;
const int yellowLight = 3;
const int redLight = 4;

// State definitions
enum TrafficState { Green, Yellow, Red };
TrafficState currentState = Green;

// Time event labels for presentation (t20 etc.)
// These are ONLY symbolic and mapped to actual durations below.
const unsigned long t20 = 10000; // Represents 10 seconds (Green)
const unsigned long t6  = 3000;  // Represents 3 seconds (Yellow)
const unsigned long t10 = 5000;  // Represents 5 seconds (Red)

// Durations bound to the symbolic time events
unsigned long previousMs = 0;
const unsigned long greenDuration  = t20;
const unsigned long yellowDuration = t6;
const unsigned long redDuration    = t10;

void setup() {
  pinMode(greenLight, OUTPUT);
  pinMode(yellowLight, OUTPUT);
  pinMode(redLight, OUTPUT);

  // Initial state: Green
  digitalWrite(greenLight, HIGH);
  previousMs = millis();
}

void loop() {
  unsigned long currentMs = millis();

  switch (currentState) {

    case Green:
      // Control the light according to timing event t20
      if (currentMs - previousMs >= greenDuration) {
        digitalWrite(greenLight, LOW);
        digitalWrite(yellowLight, HIGH);
        currentState = Yellow;
        previousMs = currentMs;
      }
      break;

    case Yellow:
      // Control the light according to timing event t6
      if (currentMs - previousMs >= yellowDuration) {
        digitalWrite(yellowLight, LOW);
        digitalWrite(redLight, HIGH);
        currentState = Red;
        previousMs = currentMs;
      }
      break;

    case Red:
      // Control the light according to timing event t10
      if (currentMs - previousMs >= redDuration) {
        digitalWrite(redLight, LOW);
        digitalWrite(greenLight, HIGH);
        currentState = Green;
        previousMs = currentMs;
      }
      break;
  }
}
```

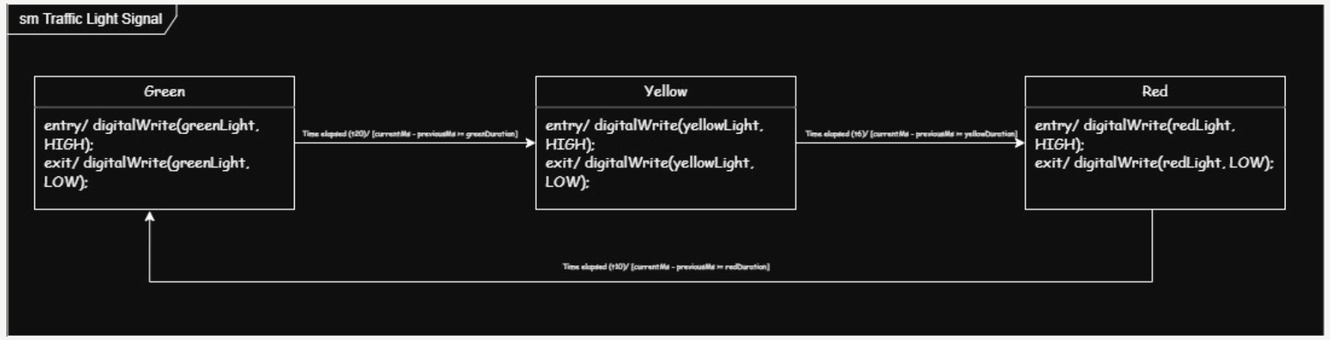
Traffic Light - Code

# Task 1





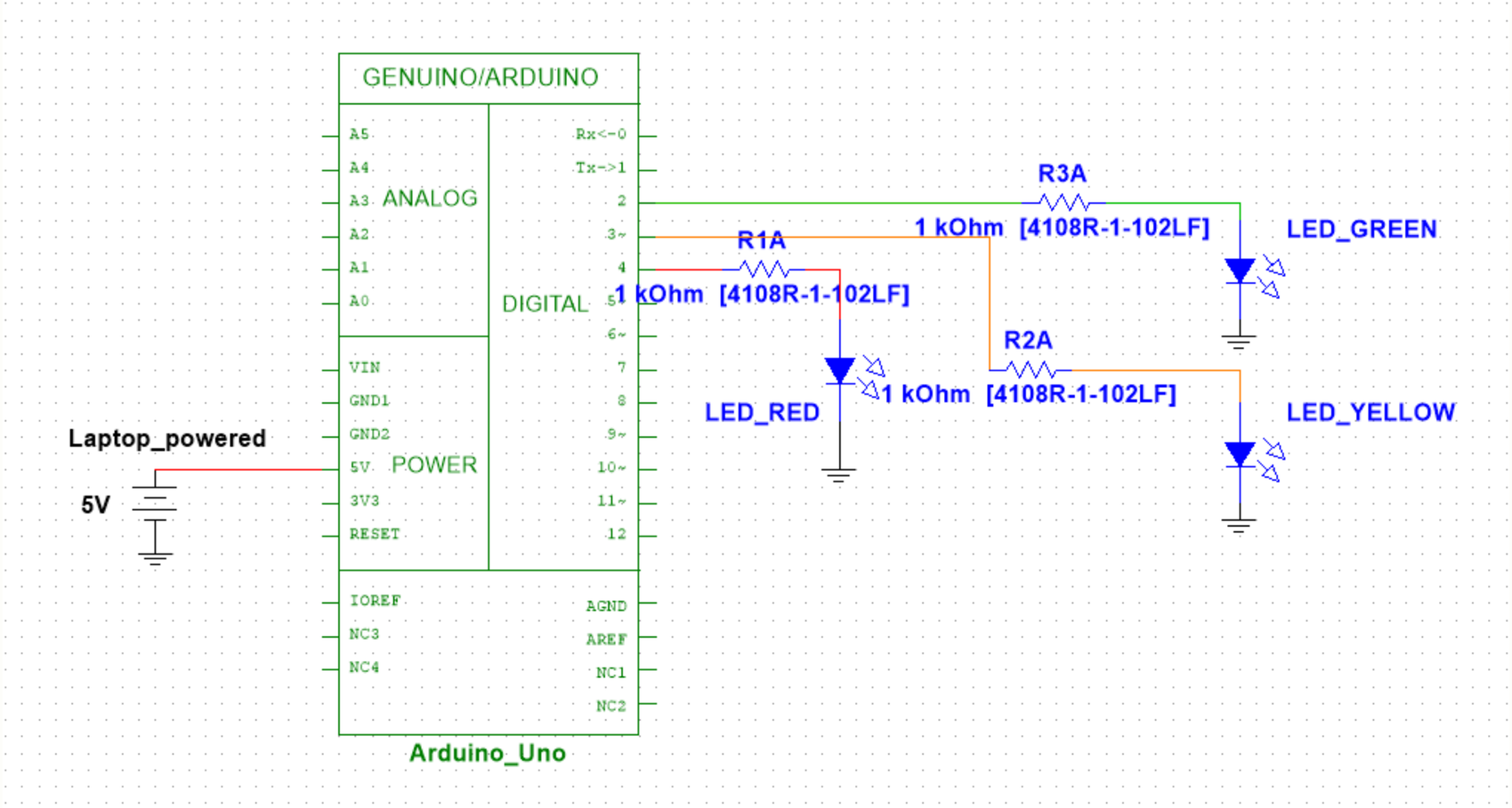

Class Diagram




State Machine Diagram

# Task 1

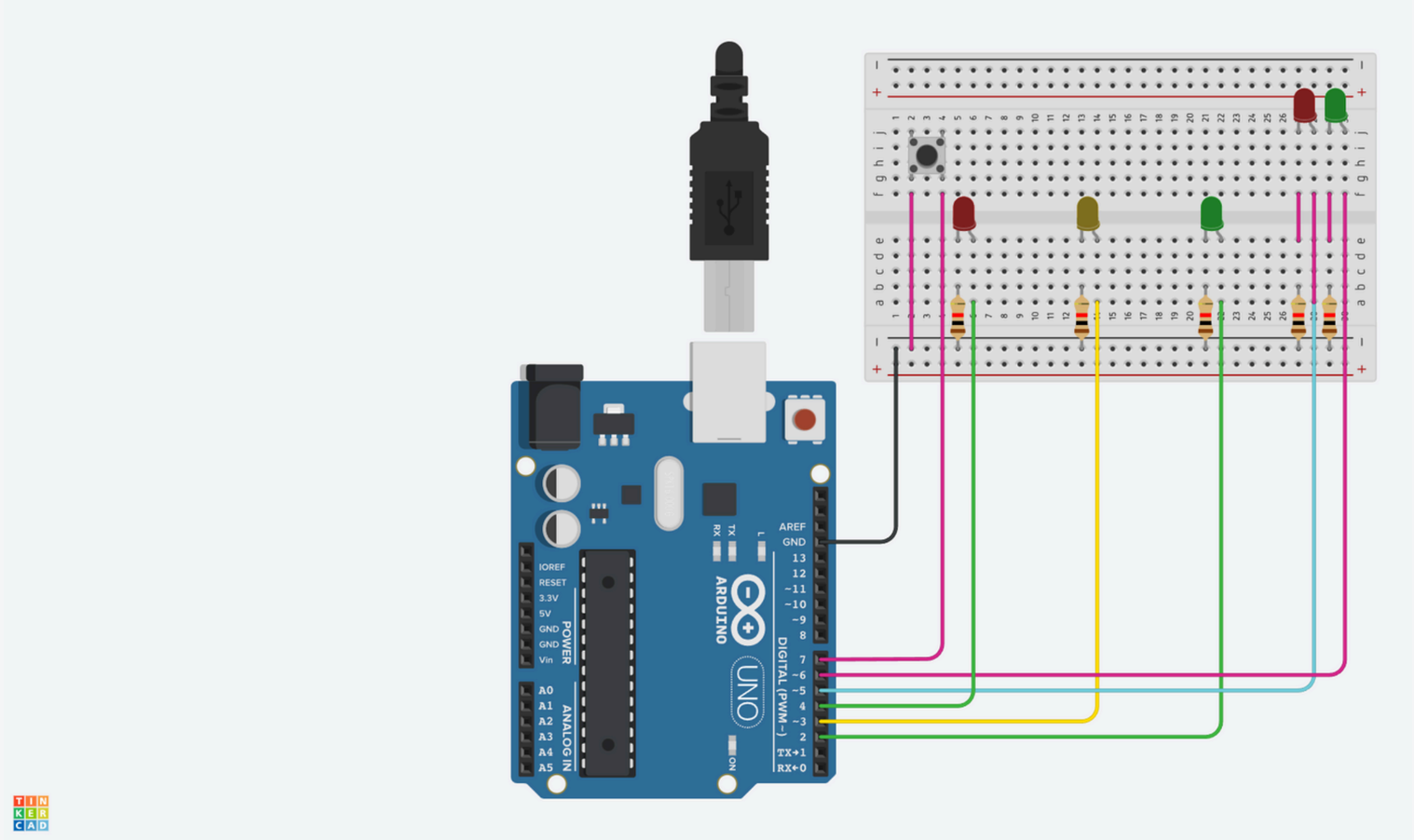





Multisim Simulation

# Task 2






TinkerCAD Simulation

# Task 2

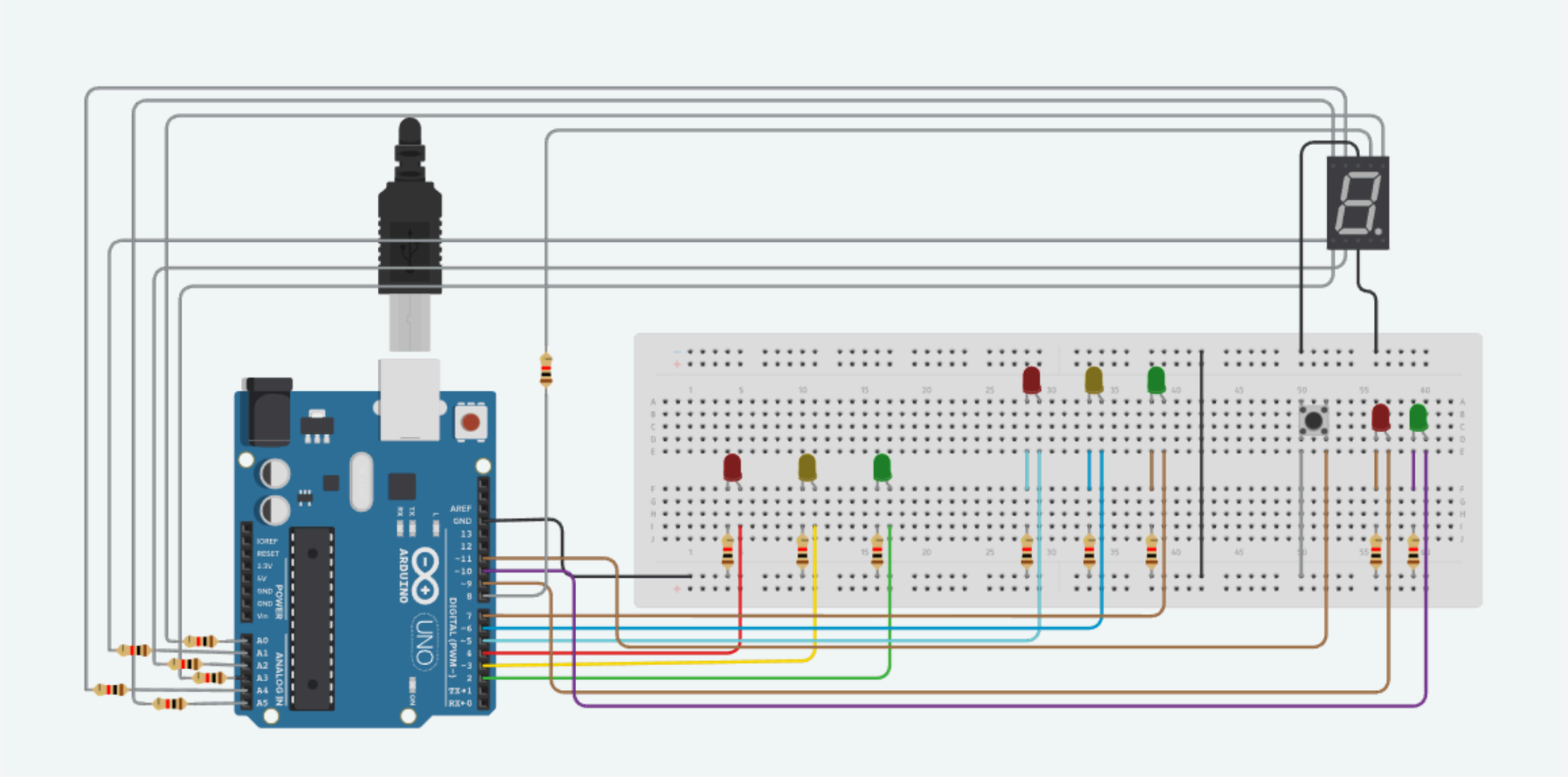


```
#include <Wire.h>

// Pin definitions
const int greenLight = 6;
const int yellowLight = 5;
const int redLight = 4;
const int pedRed = 7;
const int pedGreen = 8;
const int button = 2;  // interrupt-capable

volatile bool buttonPressed = false; // Flag set by interrupt
unsigned long previousMs = 0;
unsigned long greenDuration = 10000; // Normal green duration = 10s
unsigned long yellowDuration = 5000; // Yellow = 5s
unsigned long redDuration = 0;       // Managed dynamically

enum TrafficState { Green, Yellow, Red };
TrafficState currentState = Green;

void setup() {
  pinMode(greenLight, OUTPUT);
  pinMode(yellowLight, OUTPUT);
  pinMode(redLight, OUTPUT);
  pinMode(pedRed, OUTPUT);
  pinMode(pedGreen, OUTPUT);
  pinMode(button, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(button), buttonISR, FALLING);

  // Start with traffic green and pedestrian red
  digitalWrite(greenLight, HIGH);
  digitalWrite(pedRed, HIGH);
  previousMs = millis();
}

void loop() {
  unsigned long now = millis();

  switch (currentState) {
    case Green:
      // If button pressed, switch to yellow early (within 3 seconds)
      if (buttonPressed && (now - previousMs >= 3000)) {
        exitGreen();
        enterYellow();
      }
      // If no button pressed, normal 10s green duration
      else if (!buttonPressed && (now - previousMs >= greenDuration)) {
        exitGreen();
        enterYellow();
      }
      break;

    case Yellow:
      if (now - previousMs >= yellowDuration) {
        exitYellow();
        enterRed();
      }
      break;

    case Red:
      if (now - previousMs >= redDuration) {
        exitRed();
        enterGreen();
      }
      break;
  }
}

// ======== STATE TRANSITION FUNCTIONS ========

void enterGreen() {
  digitalWrite(greenLight, HIGH);
  digitalWrite(yellowLight, LOW);
  digitalWrite(redLight, LOW);

  digitalWrite(pedGreen, LOW);
  digitalWrite(pedRed, HIGH);

  currentState = Green;
  previousMs = millis();
}

void exitGreen() {
  digitalWrite(greenLight, LOW);
}

void enterYellow() {
  digitalWrite(yellowLight, HIGH);
  currentState = Yellow;
  previousMs = millis();
}

void exitYellow() {
  digitalWrite(yellowLight, LOW);
}

void enterRed() {
  digitalWrite(redLight, HIGH);
  currentState = Red;
  previousMs = millis();

  // Pedestrian sequence
  digitalWrite(pedRed, LOW);
  delay(2000);              // Wait before allowing pedestrians
  digitalWrite(pedGreen, HIGH);
  delay(5000);              // Pedestrian green duration

  // Reset pedestrian lights
  digitalWrite(pedGreen, LOW);
  digitalWrite(pedRed, HIGH);

  redDuration = 1000; // short pause before returning to green
}

void exitRed() {
  digitalWrite(redLight, LOW);
  buttonPressed = false; // Reset button flag
}

void buttonISR() {
  buttonPressed = true;
}
```

Traffic Light and Padestrian - Code

# Task 2






TinkerCAD Simulation - With 7 Segment Display

# Task 2

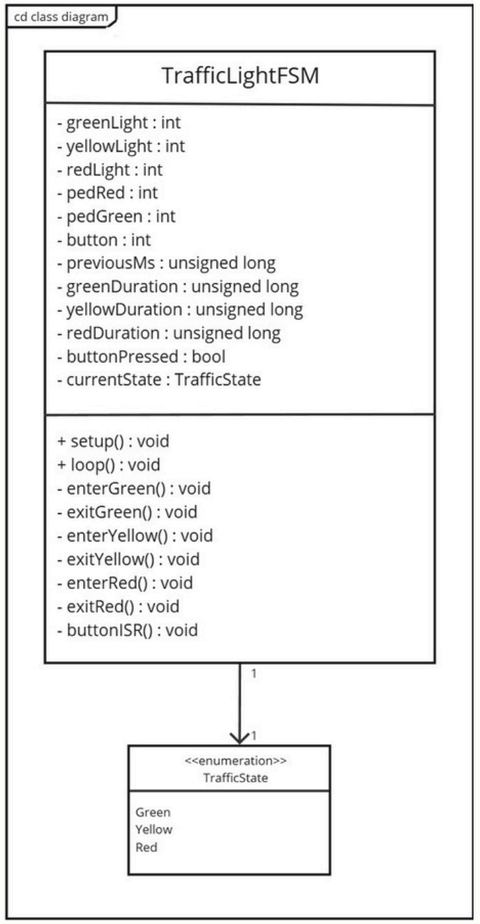


```
const int bottomGreenLight = 2;
const int bottomYellowLight = 3;
const int bottomRedLight   = 4;

const int secondaryBottomGreenLight  = 7;
const int secondaryBottomYellowLight = 6;
const int secondaryBottomRedLight    = 5;

const int pedRed   = 9;
const int pedGreen = 10;

const int button = 11;   // Pedestrian button (INPUT_PULLUP)

// Segment mapping: a, b, c, d, e, f, g
const int segA = 8;   // D8
const int segB = A0;  // digital 14
const int segC = A1;  // digital 15
const int segD = A2;  // digital 16
const int segE = A3;  // digital 17
const int segF = A4;  // digital 18
const int segG = A5;  // digital 19

const int segmentPins[7] = { segA, segB, segC, segD, segE, segF, segG };

// Digit patterns for 0–9 for common cathode (1 = segment ON)
const byte digitPatterns[10][7] = {
  // a, b, c, d, e, f, g
  {1,1,1,1,1,1,0}, // 0
  {0,1,1,0,0,0,0}, // 1
  {1,1,0,1,1,0,1}, // 2
  {1,1,1,1,0,0,1}, // 3
  {0,1,1,0,0,1,1}, // 4
  {1,0,1,1,0,1,1}, // 5
  {1,0,1,1,1,1,1}, // 6
  {1,1,1,0,0,0,0}, // 7
  {1,1,1,1,1,1,1}, // 8
  {1,1,1,1,0,1,1}  // 9
};

enum TrafficState { GREEN, YELLOW, RED };
TrafficState currentState = GREEN;

unsigned long stateTimer = 0;
bool buttonPressed = false;

void blankDisplay() {
  for (int i = 0; i < 7; i++) {
    digitalWrite(segmentPins[i], LOW);  // common cathode: LOW = off
  }
}

void displayDigit(int d) {
  if (d < 0 || d > 9) {
    blankDisplay();
    return;
  }
  for (int i = 0; i < 7; i++) {
    digitalWrite(segmentPins[i], digitPatterns[d][i] ? HIGH : LOW);
  }
}

void setup() {
  // Car lights
  pinMode(bottomGreenLight, OUTPUT);
  pinMode(bottomYellowLight, OUTPUT);
  pinMode(bottomRedLight, OUTPUT);

  pinMode(secondaryBottomGreenLight, OUTPUT);
  pinMode(secondaryBottomYellowLight, OUTPUT);
  pinMode(secondaryBottomRedLight, OUTPUT);

  // Pedestrian LEDs
  pinMode(pedRed, OUTPUT);
  pinMode(pedGreen, OUTPUT);

  // Button
  pinMode(button, INPUT_PULLUP);

  // 7-seg pins
  for (int i = 0; i < 7; i++) {
    pinMode(segmentPins[i], OUTPUT);
  }
  blankDisplay();

  // Initial state: cars GREEN, pedestrian RED
  digitalWrite(bottomGreenLight, HIGH);
  digitalWrite(secondaryBottomGreenLight, HIGH);

  digitalWrite(bottomYellowLight, LOW);
  digitalWrite(secondaryBottomYellowLight, LOW);

  digitalWrite(bottomRedLight, LOW);
  digitalWrite(secondaryBottomRedLight, LOW);

  digitalWrite(pedRed, HIGH);
  digitalWrite(pedGreen, LOW);

  stateTimer = millis();
}

void loop() {
  // ---- Check button during GREEN ----
  if (currentState == GREEN && !buttonPressed && digitalRead(button) == LOW) {
    buttonPressed = true;
    delay(50); // debouncing
  }

  switch (currentState) {

    case GREEN: {
      unsigned long elapsed = millis() - stateTimer;
      unsigned long requiredGreen = buttonPressed ? 5000UL : 10000UL;

      // Cars green, pedestrian red
      blankDisplay();
      digitalWrite(pedRed, HIGH);
      digitalWrite(pedGreen, LOW);

      if (elapsed >= requiredGreen) {
        // Transition to YELLOW
        digitalWrite(bottomGreenLight, LOW);
        digitalWrite(secondaryBottomGreenLight, LOW);

        digitalWrite(bottomYellowLight, HIGH);
        digitalWrite(secondaryBottomYellowLight, HIGH);

        currentState = YELLOW;
        stateTimer = millis();
      }
      break;
    }

    case YELLOW: {
      blankDisplay();
      digitalWrite(pedRed, HIGH);
      digitalWrite(pedGreen, LOW);

      if (millis() - stateTimer >= 2000UL) {
        // Transition to RED
        digitalWrite(bottomYellowLight, LOW);
        digitalWrite(secondaryBottomYellowLight, LOW);

        digitalWrite(bottomRedLight, HIGH);
        digitalWrite(secondaryBottomRedLight, HIGH);

        digitalWrite(pedGreen, HIGH);
        digitalWrite(pedRed, LOW);

        currentState = RED;
        stateTimer = millis();
      }
      break;
    }

    case RED: {
      unsigned long elapsed = millis() - stateTimer;

      int remaining = 9 - (int)(elapsed / 1000UL);
      if (remaining < 0) remaining = 0;

      displayDigit(remaining);

      if (elapsed >= 10000UL) {
        // Back to GREEN
        digitalWrite(bottomRedLight, LOW);
        digitalWrite(secondaryBottomRedLight, LOW);

        digitalWrite(bottomGreenLight, HIGH);
        digitalWrite(secondaryBottomGreenLight, HIGH);

        digitalWrite(pedGreen, LOW);
        digitalWrite(pedRed, HIGH);

        blankDisplay();

        currentState = GREEN;
        buttonPressed = false;
        stateTimer = millis();
      }
      break;
    }
  }
}
```

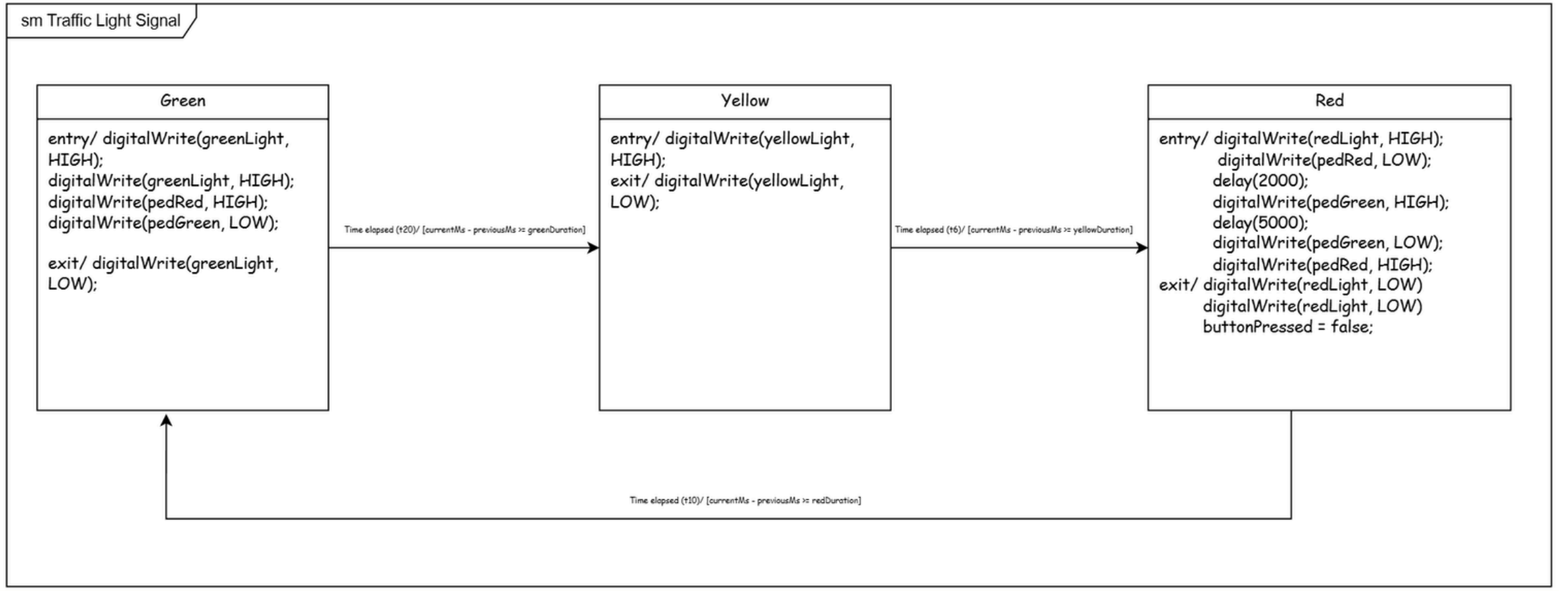
Traffic Light and Padestrian with 7 Segment Display - Code

# Task 2



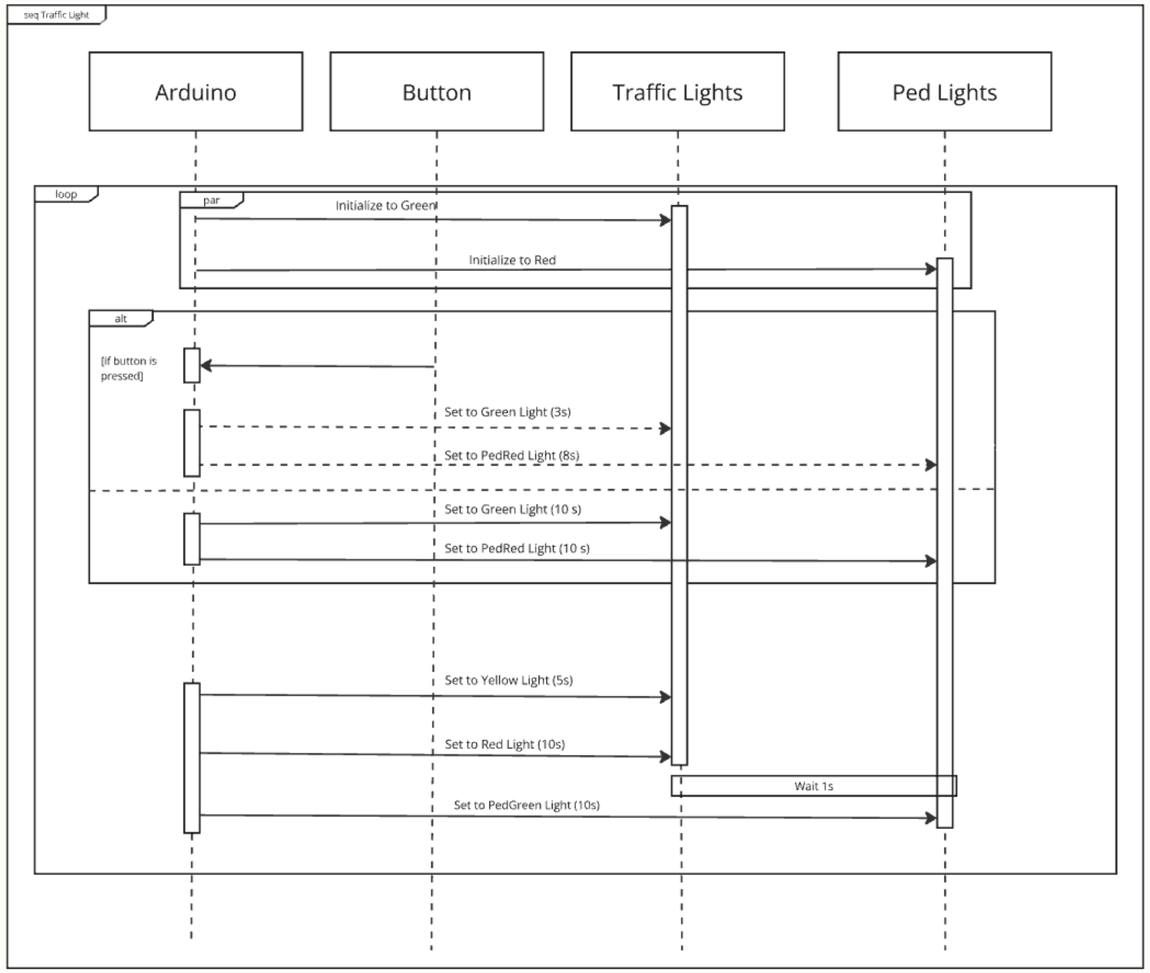



Class Diagram



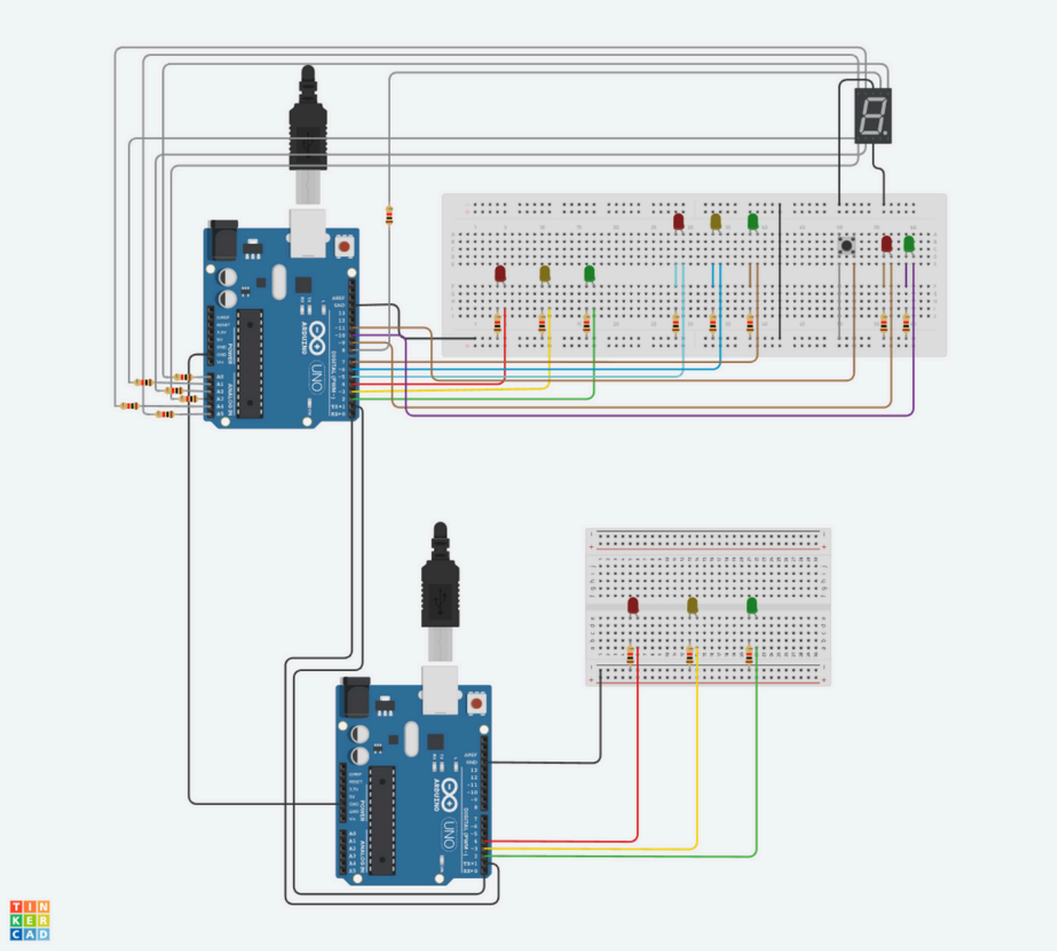

State Machine Diagram

# Task 2




Sequence Diagram

# Task 5



TinkerCAD Simulation - T-Junction with 7 Segment Display

# Task 5



```
// Master (Top Arduino)
const int bottomGreenLight = 2;
const int bottomYellowLight = 3;
const int bottomRedLight   = 4;

const int secondaryBottomGreenLight  = 7;
const int secondaryBottomYellowLight = 6;
const int secondaryBottomRedLight    = 5;

const int pedRed   = 9;
const int pedGreen = 10;

const int button = 11;   // Pedestrian button (INPUT_PULLUP)

// Segment mapping: a, b, c, d, e, f, g
const int segA = 8;   // D8
const int segB = A0;  // digital 14
const int segC = A1;  // digital 15
const int segD = A2;  // digital 16
const int segE = A3;  // digital 17
const int segF = A4;  // digital 18
const int segG = A5;  // digital 19

const int segmentPins[7] = { segA, segB, segC, segD, segE, segF, segG };

// Digit patterns for 0–9 for common cathode (1 = segment ON)
const byte digitPatterns[10][7] = {
  // a, b, c, d, e, f, g
  {1,1,1,1,1,1,0}, // 0
  {0,1,1,0,0,0,0}, // 1
  {1,1,0,1,1,0,1}, // 2
  {1,1,1,1,0,0,1}, // 3
  {0,1,1,0,0,1,1}, // 4
  {1,0,1,1,0,1,1}, // 5
  {1,0,1,1,1,1,1}, // 6
  {1,1,1,0,0,0,0}, // 7
  {1,1,1,1,1,1,1}, // 8
  {1,1,1,1,0,1,1}  // 9
};

enum TrafficState { GREEN, YELLOW, RED };
TrafficState currentState = GREEN;

unsigned long stateTimer = 0;
bool buttonPressed = false;

void sendState() {
  switch (currentState) {
    case GREEN:
      Serial.println("STATE_GREEN");
      break;
    case YELLOW:
      Serial.println("STATE_YELLOW");
      break;
    case RED:
      Serial.println("STATE_RED");
      break;
  }
}

void blankDisplay() {
  for (int i = 0; i < 7; i++) {
    digitalWrite(segmentPins[i], LOW);  // common cathode: LOW = off
  }
}

void displayDigit(int d) {
  if (d < 0 || d > 9) {
    blankDisplay();
    return;
  }
  for (int i = 0; i < 7; i++) {
    digitalWrite(segmentPins[i], digitPatterns[d][i] ? HIGH : LOW);
  }
}

void setup() {
  // Car lights
  pinMode(bottomGreenLight, OUTPUT);
  pinMode(bottomYellowLight, OUTPUT);
  pinMode(bottomRedLight, OUTPUT);

  pinMode(secondaryBottomGreenLight, OUTPUT);
  pinMode(secondaryBottomYellowLight, OUTPUT);
  pinMode(secondaryBottomRedLight, OUTPUT);

  // Pedestrian LEDs
  pinMode(pedRed, OUTPUT);
  pinMode(pedGreen, OUTPUT);

  // Button
  pinMode(button, INPUT_PULLUP);

  // 7-seg pins
  for (int i = 0; i < 7; i++) {
    pinMode(segmentPins[i], OUTPUT);
  }
  blankDisplay();

  // UART
  Serial.begin(9600);

  // Initial state: cars GREEN, pedestrian RED
  digitalWrite(bottomGreenLight, HIGH);
  digitalWrite(secondaryBottomGreenLight, HIGH);

  digitalWrite(bottomYellowLight, LOW);
  digitalWrite(secondaryBottomYellowLight, LOW);
  digitalWrite(bottomRedLight, LOW);
  digitalWrite(secondaryBottomRedLight, LOW);

  digitalWrite(pedRed, HIGH);
  digitalWrite(pedGreen, LOW);

  stateTimer = millis();
  sendState();
}

void loop() {
  // Check button during GREEN
  if (currentState == GREEN && !buttonPressed && digitalRead(button) == LOW) {
    buttonPressed = true;
    delay(50); // debouncing
  }

  switch (currentState) {

    case GREEN: {
      // Current timing: 10s if no button; 5s if button pressed
      unsigned long elapsed = millis() - stateTimer;
      unsigned long requiredGreen = buttonPressed ? 5000UL : 10000UL;

      // While cars are green, pedestrian is red, so blank display
      blankDisplay();
      digitalWrite(pedRed, HIGH);
      digitalWrite(pedGreen, LOW);

      if (elapsed >= requiredGreen) {
        // Transition to YELLOW
        digitalWrite(bottomGreenLight, LOW);
        digitalWrite(secondaryBottomGreenLight, LOW);

        digitalWrite(bottomYellowLight, HIGH);
        digitalWrite(secondaryBottomYellowLight, HIGH);

        currentState = YELLOW;
        stateTimer = millis();
        sendState();
      }
      break;
    }

    case YELLOW: {
      // Cars yellow, ped still red, display stays blank
      blankDisplay();
      digitalWrite(pedRed, HIGH);
      digitalWrite(pedGreen, LOW);

      if (millis() - stateTimer >= 2000UL) { // 2 seconds yellow
        // Transition to RED (cars stop, ped go)
        digitalWrite(bottomYellowLight, LOW);
        digitalWrite(secondaryBottomYellowLight, LOW);

        digitalWrite(bottomRedLight, HIGH);
        digitalWrite(secondaryBottomRedLight, HIGH);

        digitalWrite(pedGreen, HIGH);  // pedestrian can cross
        digitalWrite(pedRed, LOW);

        currentState = RED;
        stateTimer = millis();
        sendState();
      }
      break;
    }

    case RED: {
      // Cars red, pedestrian green: show countdown 9 till 0
      unsigned long elapsed = millis() - stateTimer;

      // Each second, decrease the displayed number
      int remaining = 9 - (int)(elapsed / 1000UL);
      if (remaining < 0) remaining = 0;

      displayDigit(remaining);

      if (elapsed >= 10000UL) { // 10 seconds of red/ped green
        // Back to GREEN
        digitalWrite(bottomRedLight, LOW);
        digitalWrite(secondaryBottomRedLight, LOW);

        digitalWrite(bottomGreenLight, HIGH);
        digitalWrite(secondaryBottomGreenLight, HIGH);

        digitalWrite(pedGreen, LOW);  // stop pedestrian crossing
        digitalWrite(pedRed, HIGH);

        blankDisplay();  // turn off countdown

        currentState = GREEN;
        buttonPressed = false;
        stateTimer = millis();
        sendState();
      }
      break;
    }
  }
}
```

T-Junction with 7 Segment Display - Master Code

# Task 5

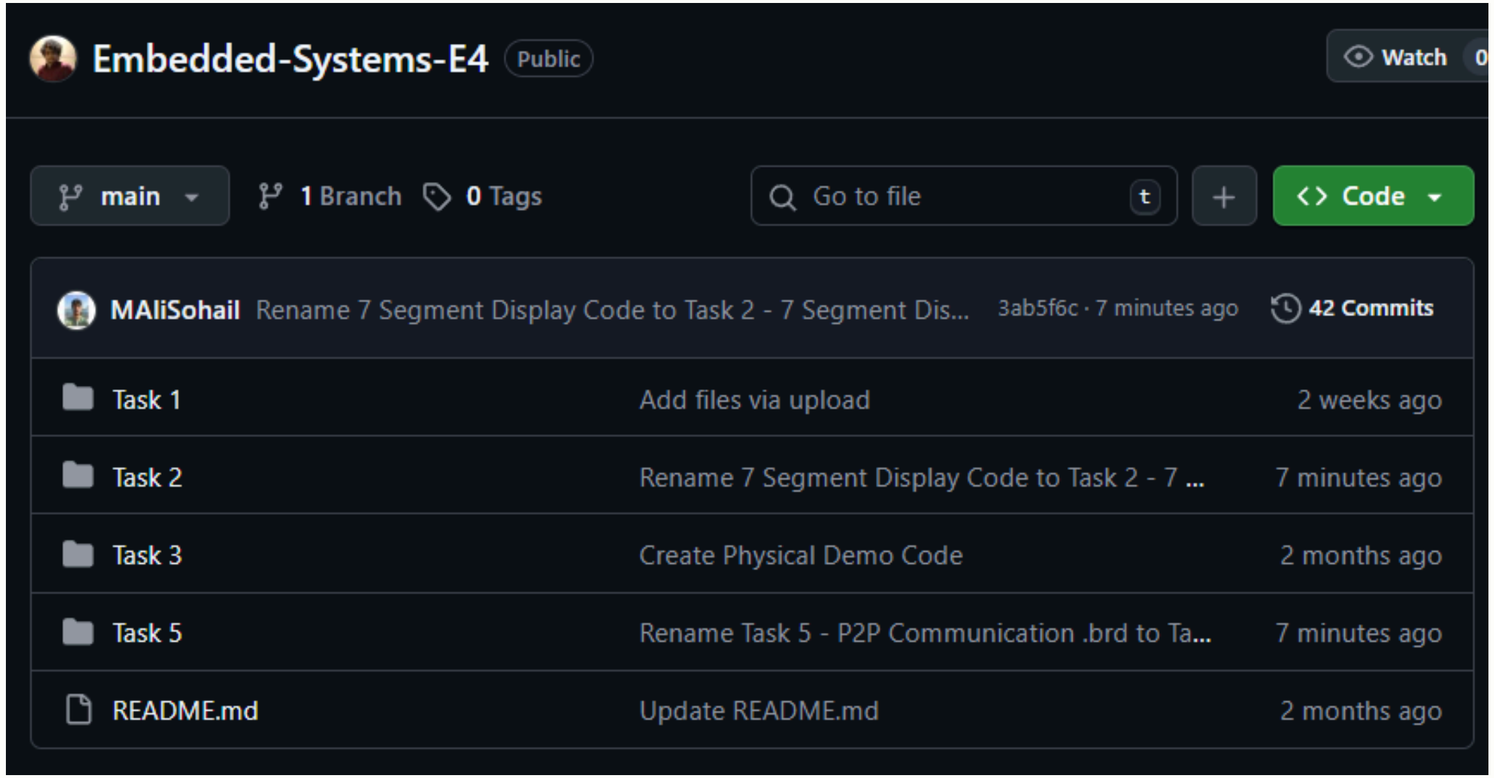


```
// SLAVE (Bottom Arduino)

const int slaveGreen  = 2;
const int slaveYellow = 3;  // unused in this simple mapping
const int slaveRed    = 4;

void setSlaveForState(const String& state) {
  if (state == "STATE_GREEN" || state == "STATE_YELLOW") {
    // Master cars moving then pedestrian red then slave red
    digitalWrite(slaveRed, HIGH);
    digitalWrite(slaveGreen, LOW);
    digitalWrite(slaveYellow, LOW);
  }
  else if (state == "STATE_RED") {
    // Master cars stopped then pedestrian green then slave green
    digitalWrite(slaveRed, LOW);
    digitalWrite(slaveGreen, HIGH);
    digitalWrite(slaveYellow, LOW);
  }
}

void setup() {
  pinMode(slaveGreen, OUTPUT);
  pinMode(slaveYellow, OUTPUT);
  pinMode(slaveRed, OUTPUT);

  Serial.begin(9600);

  // Default: assume master starts in GREEN then slave must be RED
  digitalWrite(slaveRed, HIGH);
  digitalWrite(slaveGreen, LOW);
  digitalWrite(slaveYellow, LOW);
}

void loop() {
  if (Serial.available() > 0) {
    String msg = Serial.readStringUntil('\n');
    msg.trim(); // remove \r, spaces

    if (msg == "STATE_GREEN" || msg == "STATE_YELLOW" || msg == "STATE_RED") {
      setSlaveForState(msg);
    }
  }
}
```

T-Junction with 7 Segment Display - Slave Code

# Git Usage

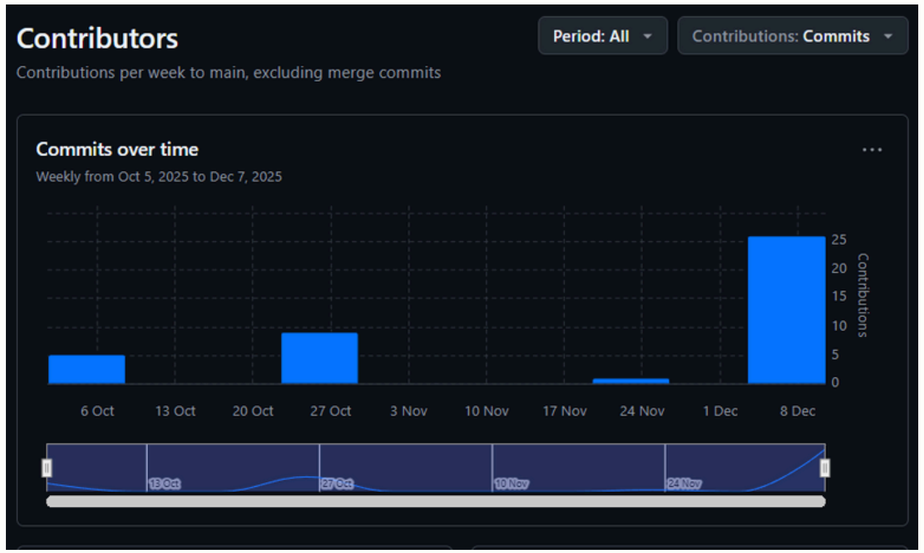





Folders on Github

# Git Usage







Github Commits