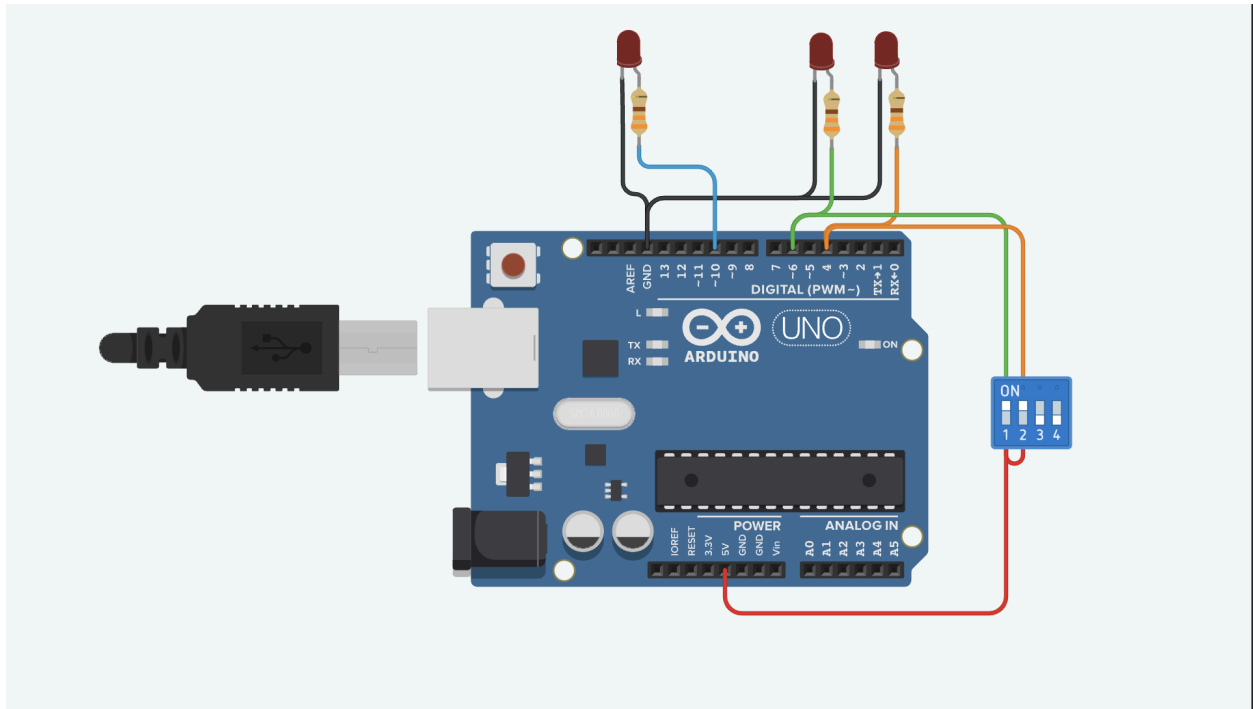


- [Basic Logic Gates](#)
- [Blinking of LED Single](#)
- [Blinking of LED Single Multiple \(1 at a time\)](#)
- [Blinking of LED all together](#)
- [Blinking of LED 2 at a time/ Alternate](#)
- [Ultrasonic Sensor](#)
- [Soil Moisture](#)
- [Temperature](#)
- [PING PONG](#)

## Basic Logic Gates



```
int InputA = 0;
int InputB = 0;
void setup(){
  pinMode(6, INPUT);
  pinMode(4, INPUT);
  pinMode(10, OUTPUT);
}
void loop(){
  InputA = digitalRead(6);
  InputB = digitalRead(4);

  //-----

  //-----AND GATE-----
  //if(InputA == HIGH && InputB == HIGH){
  //digitalWrite(10, HIGH);
  //}else{
  //digitalWrite(10, LOW);
  //}
```

```

//-----OR GATE-----
//if(InputA == HIGH || InputB == HIGH){
//digitalWrite(10, HIGH);
//}else{
//digitalWrite(10, LOW);
//}

//-----NAND GATE-----
//if(InputA == HIGH && InputB == HIGH){
//digitalWrite(10, LOW);
//}else{
//digitalWrite(10, HIGH);
//}

//-----NOR GATE-----
//if(InputA == LOW && InputB == LOW){
//digitalWrite(10, HIGH);
//}else{
//digitalWrite(10, LOW);
//}

//-----XOR GATE-----
//if((InputA == LOW && InputB == HIGH) || (InputA == HIGH && InputB == LOW)){
//digitalWrite(10, HIGH);
//}else{
//digitalWrite(10, LOW);
//}

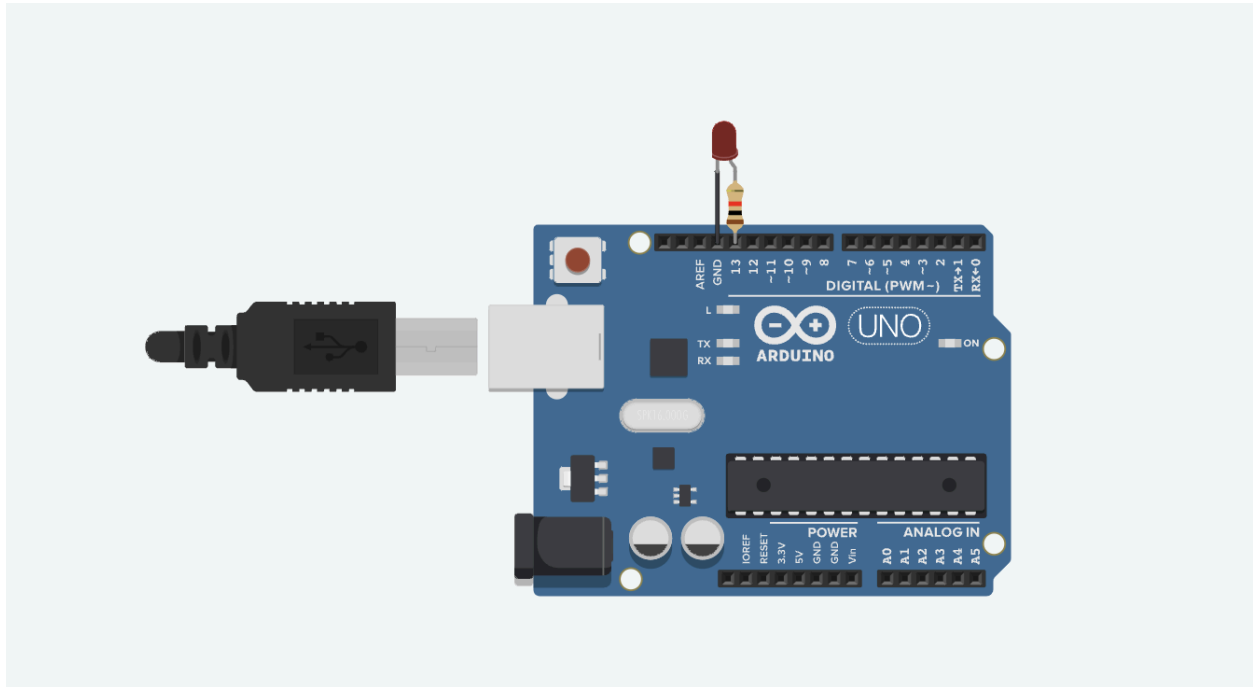
//-----XNOR GATE-----
//if((InputA == LOW && InputB == LOW) || (InputA == HIGH && InputB == HIGH)){
//digitalWrite(10, HIGH);
//}else{
//digitalWrite(10, LOW);
//}

//-----

delay(10);
}

```

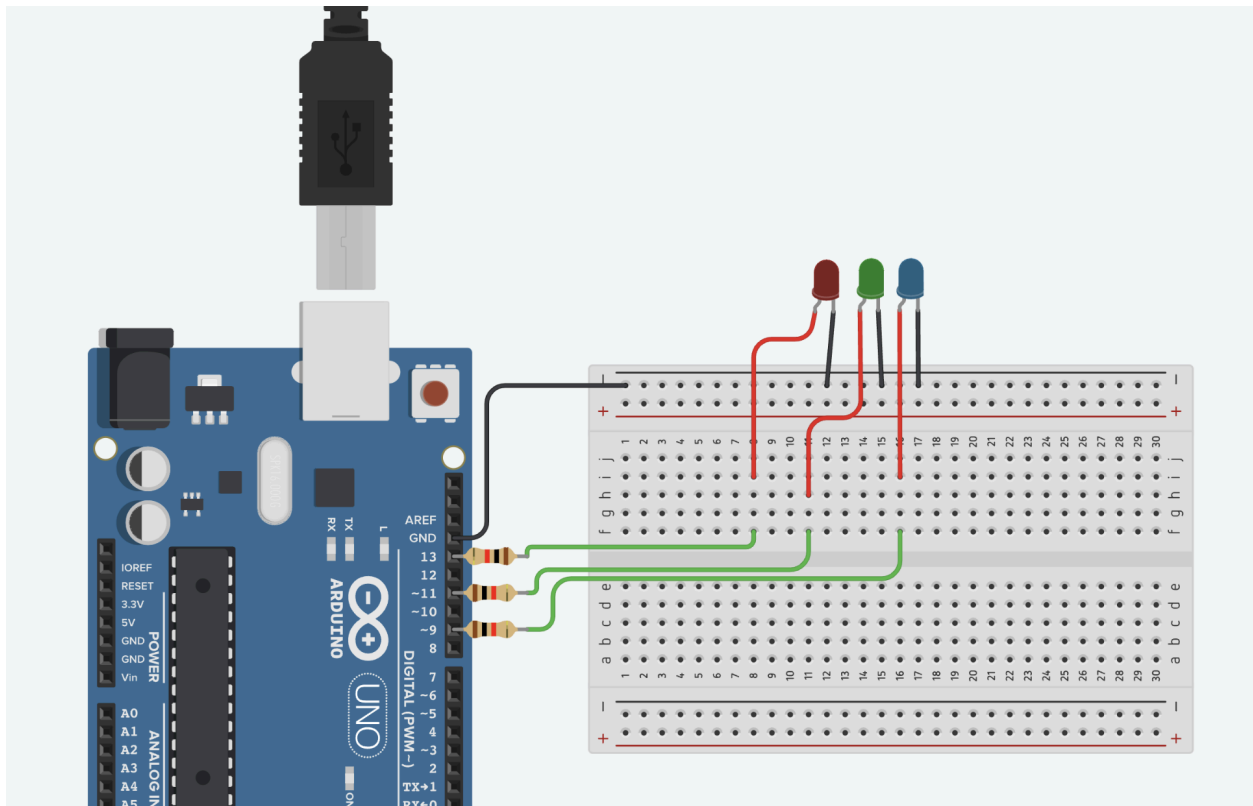
## Blinking of LED Single



```
// C++ code
//
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(500); // Wait for 1000 millisecond(s)
  digitalWrite(LED_BUILTIN, LOW);
  delay(500); // Wait for 1000 millisecond(s)
}
```

## Blinking of LED Single Multiple (1 at a time)

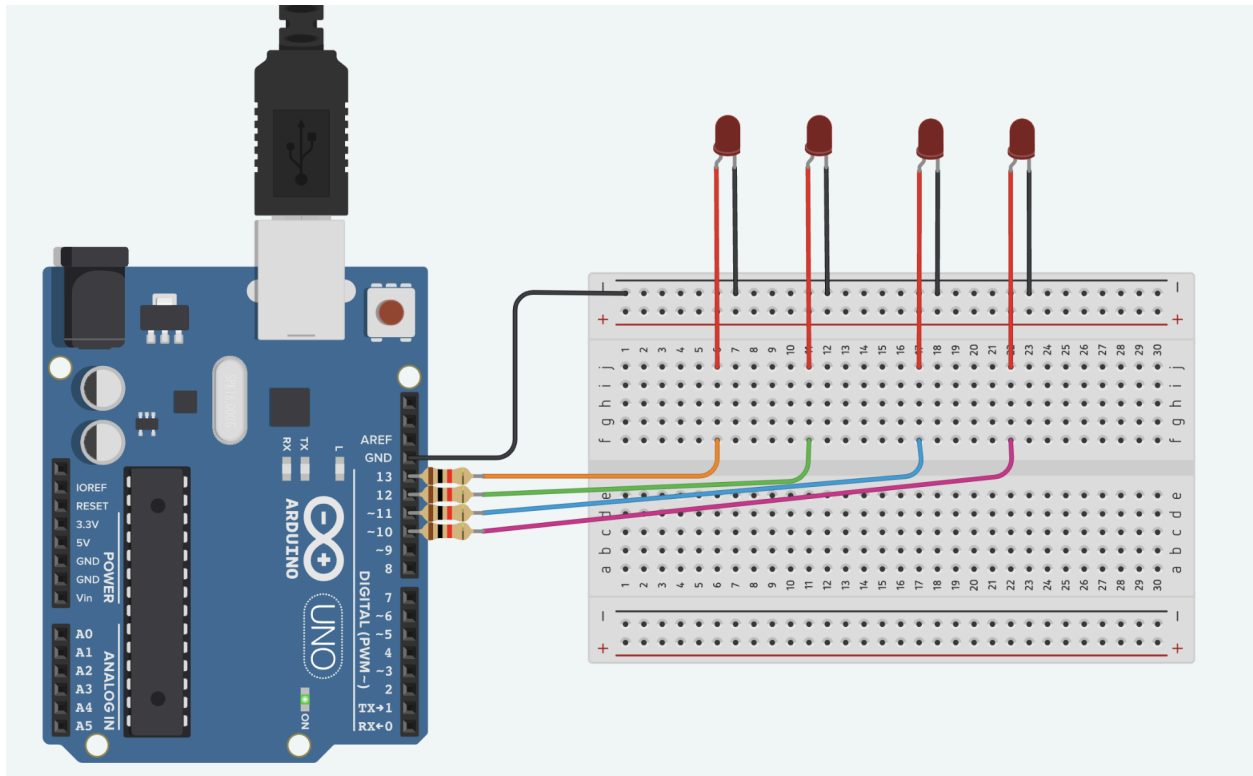


```
// C++ code
//
void setup()
{
  pinMode(13, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(9, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(200);
  digitalWrite(13, LOW);
  delay(200);
  digitalWrite(11, HIGH);
  delay(200);
  digitalWrite(11, LOW);
  delay(200);
  digitalWrite(9, HIGH);
  delay(200);
  digitalWrite(9, LOW);
}
```

```
    delay(200);  
}
```

## Blinking of LED all together



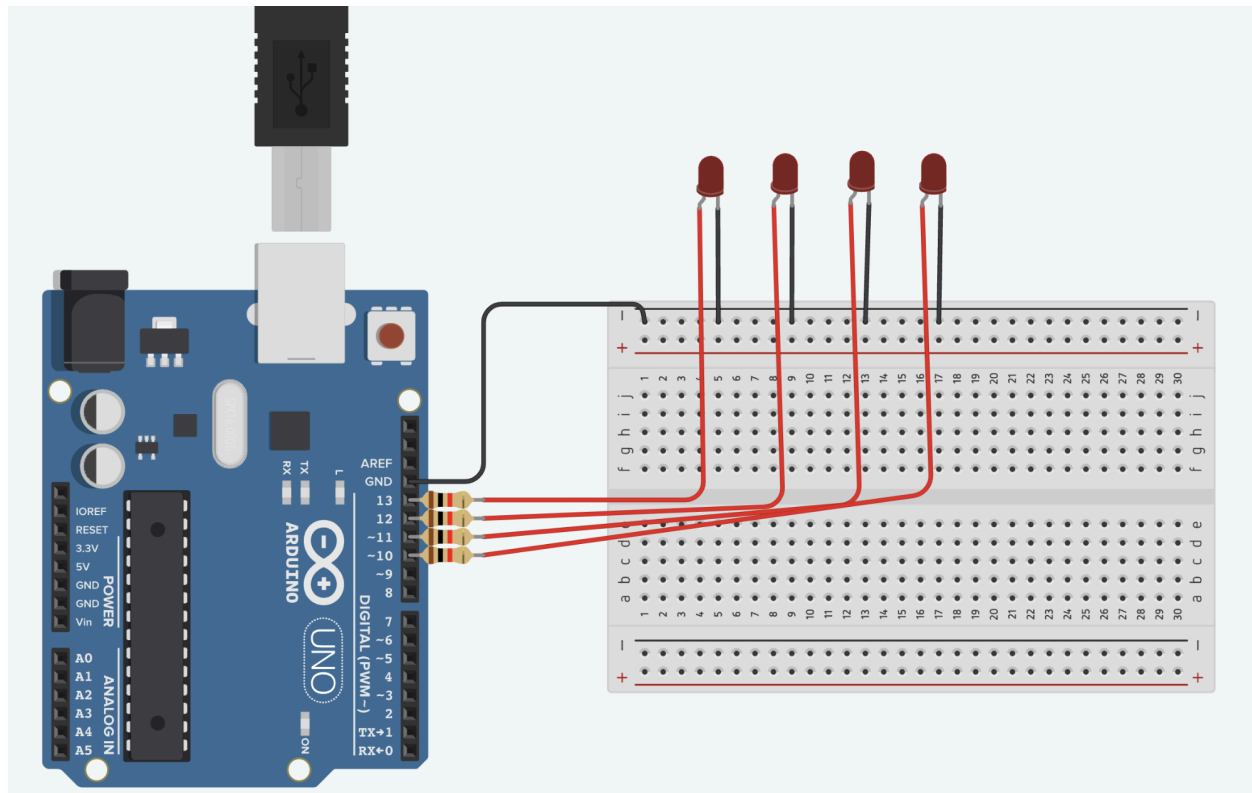
```
// C++ code  
//  
void setup()  
{  
    pinMode(13, OUTPUT);  
    pinMode(12, OUTPUT);  
    pinMode(11, OUTPUT);  
    pinMode(10, OUTPUT);  
}  
  
void loop()  
{  
    digitalWrite(13, HIGH);  
    digitalWrite(12, HIGH);  
    digitalWrite(11, HIGH);  
    digitalWrite(10, HIGH);
```

```

delay(1000);
digitalWrite(13, LOW);
digitalWrite(12, LOW);
digitalWrite(11, LOW);
digitalWrite(10, LOW);
delay(1000);
}

```

Blinking of LED 2 at a time/ Alternate



```

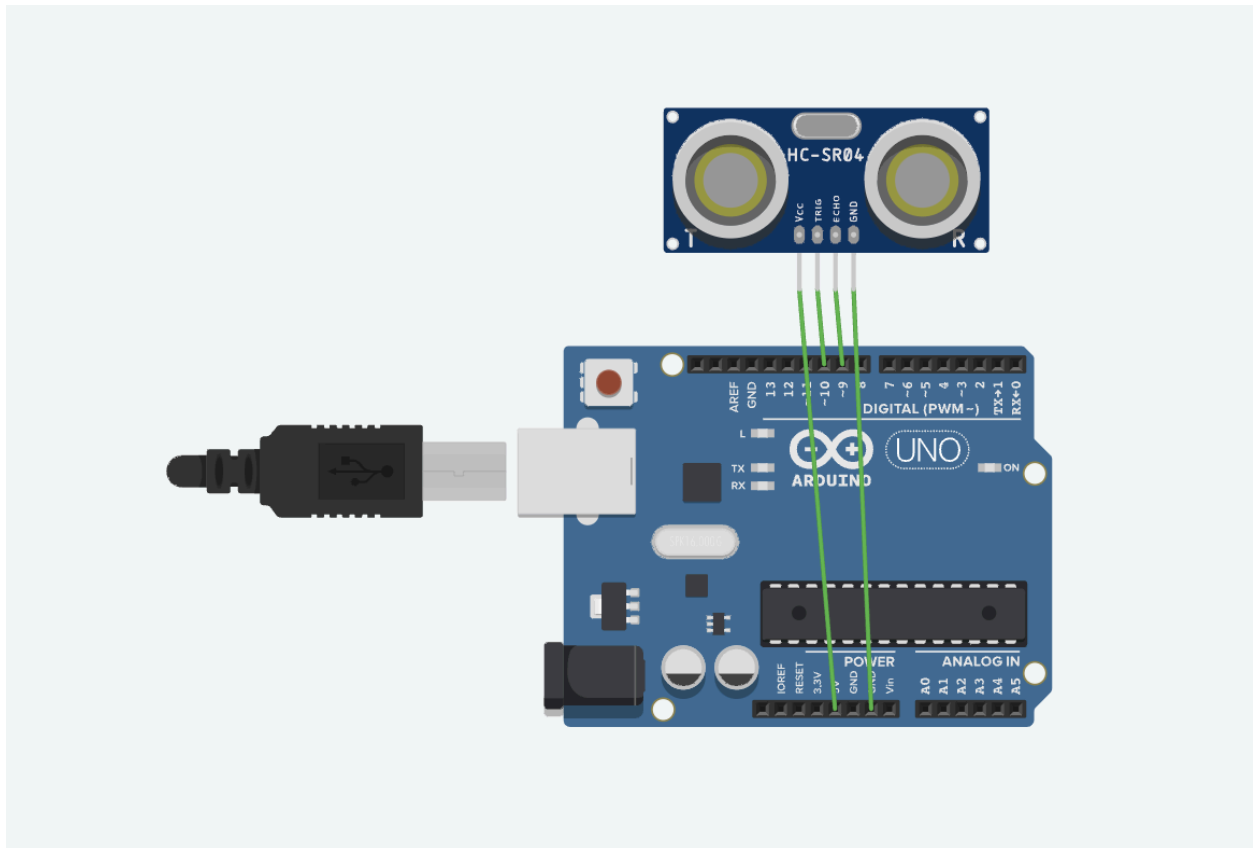
// C++ code
//
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(10, OUTPUT);
}

void loop()
{

```

```
digitalWrite(13, HIGH);  
digitalWrite(11, HIGH);  
delay(1000);  
digitalWrite(13, LOW);  
digitalWrite(11, LOW);  
delay(1000);  
digitalWrite(12, HIGH);  
digitalWrite(10, HIGH);  
delay(1000);  
digitalWrite(12, LOW);  
digitalWrite(10, LOW);  
delay(1000);  
}
```

## Ultrasonic Sensor



```
// C++ code  
//  
const int trigPin = 10;
```



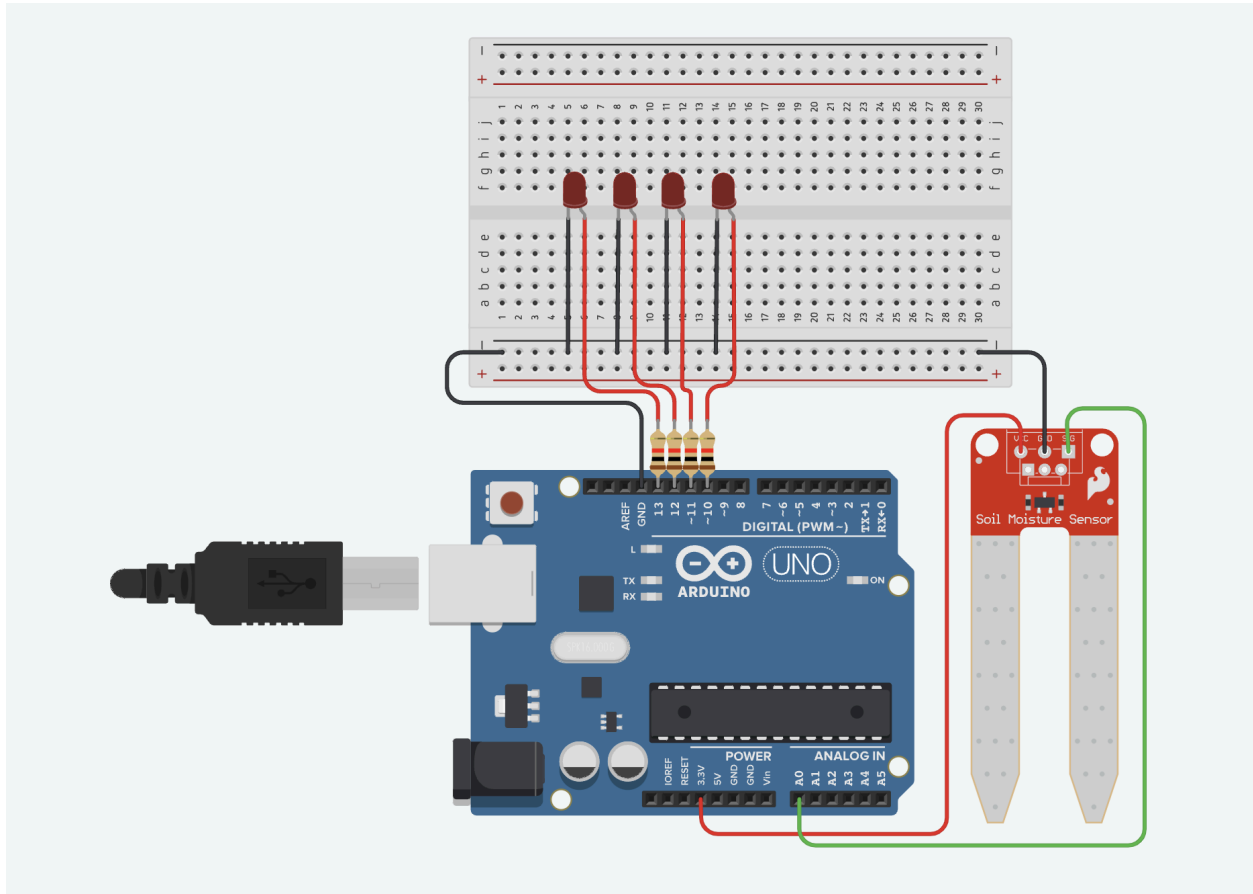
```
const int echoPin = 10;

void setup()
{
  Serial.begin(9600);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(2);
}
```

## Soil Moisture



```
// C++ code
//
int moisture_value;
float moisture_percentage;

void setup()
{
  Serial.begin(9600);
  pinMode(13,OUTPUT);
  pinMode(12,OUTPUT);
  pinMode(11,OUTPUT);
  pinMode(10,OUTPUT);
}

void loop()
{
  moisture_value = analogRead(A0);
  moisture_percentage = ((moisture_value/539.00)*100);
```

```

    if(moisture_percentage<25)
        digitalWrite(13,HIGH);
    else
    {
        digitalWrite(13,LOW);
    }
    if(moisture_percentage<50)
        digitalWrite(13,HIGH);
    else
    {
        digitalWrite(13,LOW);
        if(moisture_percentage<75)
            digitalWrite(11,HIGH);
        else
        {
            digitalWrite(11,LOW);
            if(moisture_percentage<100)
                digitalWrite(10,HIGH);
            else
                digitalWrite(10,LOW);
        }
    }
}
Serial.print("\n Moisture Value (in %): ");
Serial.print(moisture_percentage);
Serial.print("%");
delay(1000);
}

```

### **MY MODIFIED VERSION**

```

// C++ code
//
int moisture_value;
float moisture_percentage;

void setup()
{
    Serial.begin(9600);
    pinMode(13,OUTPUT);
    pinMode(12,OUTPUT);
    pinMode(11,OUTPUT);
    pinMode(10,OUTPUT);
}

void loop()

```

```

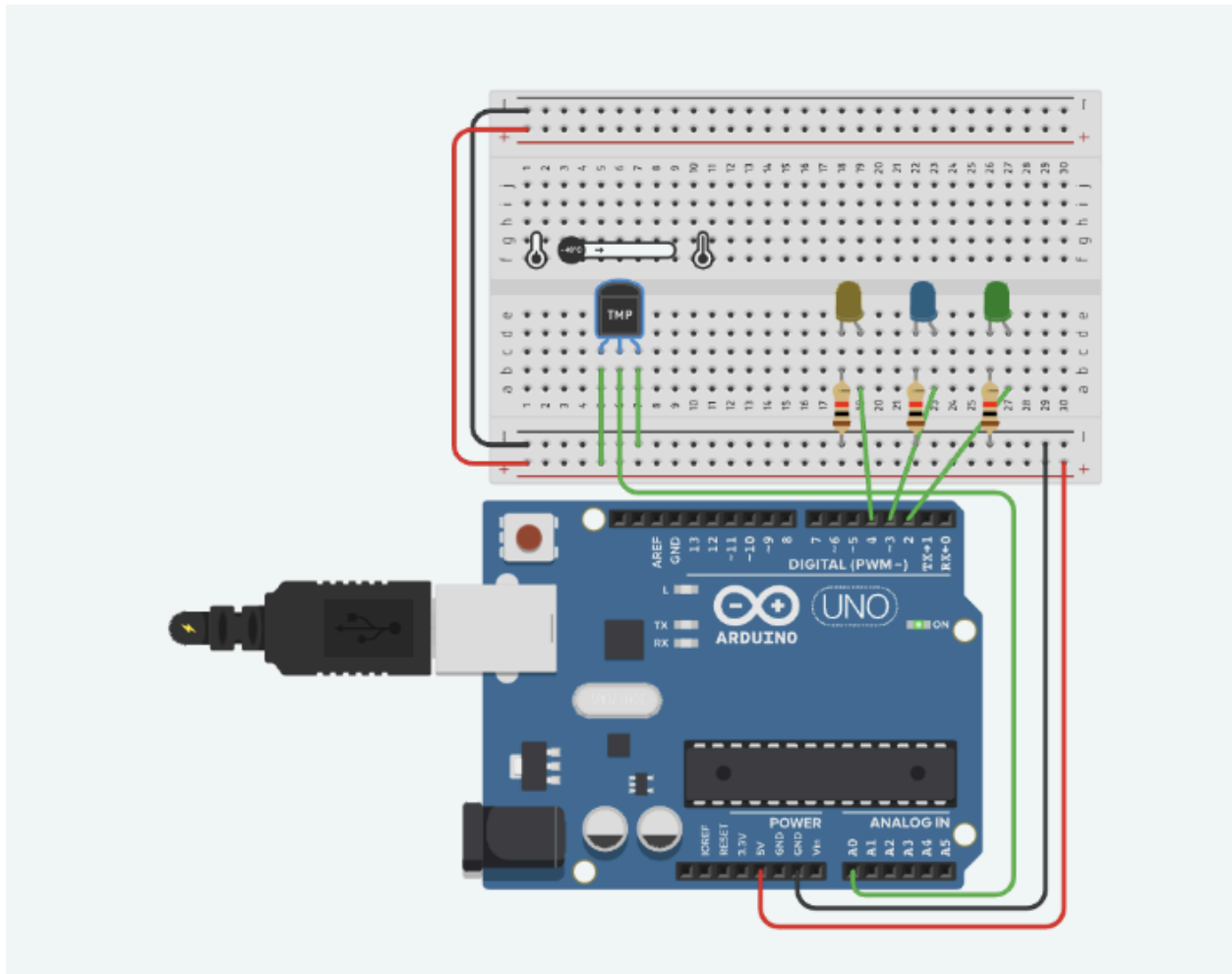
{
  moisture_value = analogRead(A0);
  moisture_percentage = ((moisture_value / 539.00) * 100);

  if (moisture_percentage < 25){
    digitalWrite(13, HIGH);
  }
  else if (moisture_percentage > 25 && moisture_percentage < 50){
    digitalWrite(13, HIGH);
    digitalWrite(12, HIGH);
  }
  else if (moisture_percentage > 50 && moisture_percentage < 75){
    digitalWrite(13, HIGH);
    digitalWrite(12, HIGH);
    digitalWrite(11, HIGH);
  }
  else if (moisture_percentage > 75){
    digitalWrite(13, HIGH);
    digitalWrite(12, HIGH);
    digitalWrite(11, HIGH);
    digitalWrite(10, HIGH);
  }
  else
  {
    digitalWrite(10, LOW);
    digitalWrite(11, LOW);
    digitalWrite(13, LOW);
  }

  Serial.print("\n Moisture Value (in %): ");
  Serial.print(moisture_percentage);
  Serial.print("%");
  delay(1000);
}

```

## Temperature



```
int baselineTemp = 0;
int celsius = 0;
int fahrenheit = 0;

void setup()
{
  pinMode(A0, INPUT);
  Serial.begin(9600);

  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

void loop()
{
  baselineTemp = 25;
```

```

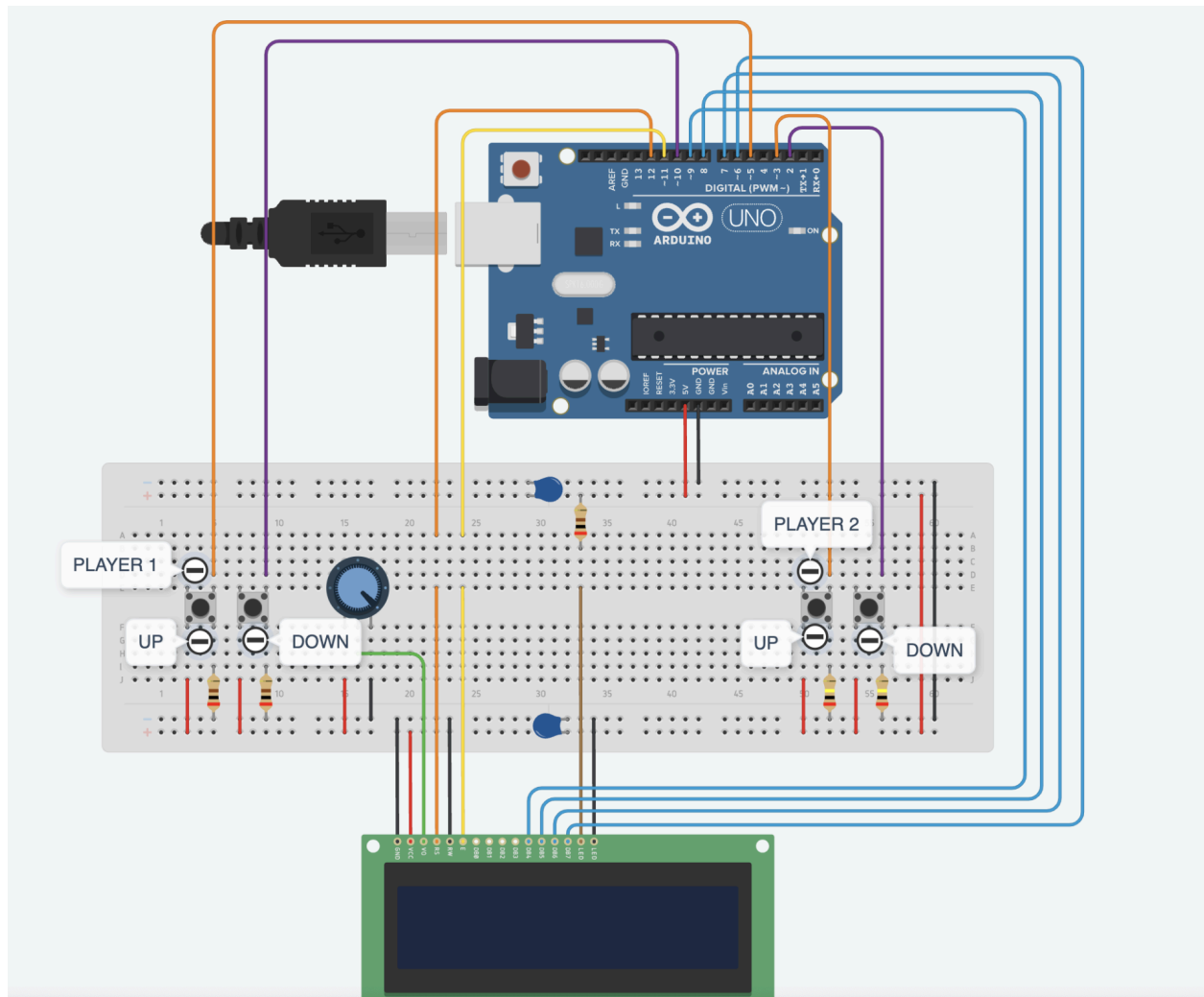
celsius = map(((analogRead(A0) - 20) * 3.04), 0, 1023, -40, 125);

fahrenheit = ((celsius * 9) / 5 + 32);
Serial.print(celsius);
Serial.print(" C, ");
Serial.print(fahrenheit);
Serial.println(" F");

if (celsius < baselineTemp) {
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
}
if (celsius >= baselineTemp && celsius < baselineTemp + 10) {
  digitalWrite(2, HIGH);
  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
}
if (celsius >= baselineTemp + 10 && celsius < baselineTemp + 20) {
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, LOW);
}
if (celsius >= baselineTemp + 20 && celsius < baselineTemp + 30) {
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
}
if (celsius >= baselineTemp + 30) {
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
}
delay(1000);
}

```

## PING PONG



```
#include <LiquidCrystal.h>
#include <avr/interrupt.h>
```

```
//-----MACROS-----
```

```
// Assigning LCD PINS
```

```
#define RS 12
```

```
#define EN 11
```

```
#define D4 9
```

```
#define D5 8
```

```
#define D6 7
```

```
#define D7 6
```

```
// Buttons that move the paddle
```

```
#define Player_1_moveDownButton 10 // Player 1's Down Button is connected to digital pin 4
```

```
#define Player_1_moveUpButton 5 // Player 1's Down Button is connected to digital pin 4
#define Player_2_moveDownButton 2 // Player 2's Down Button is connected to digital pin 4
#define Player_2_moveUpButton 3 // Player 2's Down Button is connected to digital pin 4
```

```
// Assigning Player Numbers
```

```
#define Player_1 1
```

```
#define Player_2 2
```

```
// Delay for updating the ball
```

```
#define DiagonalballUpdateTime 21
```

```
#define HorizontalballUpdateTime 15
```

```
//Starting Position of the Ball
```

```
#define Start_X 35
```

```
#define Start_Y 7
```

```
#define Button_Pressed (p1_UpButState | p1_DownButState | p2_UpButState |
p2_DownButState)
```

```
void(* resetFunc) (void) = 0; //declaring reset function at address 0
```

```
// Global Variables for Pin Change Interrupt Service Routine
```

```
volatile boolean x_Up = true;
```

```
volatile boolean x_Down = true;
```

```
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7); // Creating Object of type Liquid Crystal
```

```
//-----PADDLE CLASS-----
```

```
class Paddle
```

```
{
```

```
public:
```

```
// Storing Value of each row of the Character Column for the Player's Paddle
```

```
uint8_t PaddleColArray[16] = {0,0,0,0,0,4,4,4,0,0,0,0,0,0,0};
```

```
uint8_t PaddlePos = 6; // Recording Paddles's middle position as reference to move it
```

```
uint8_t Score = 0; // Score of each Player
```

```
// When the Player presses the "UP" Button
```

```
void MovePaddleUp()
```

```
{
```

```
// Make sure the paddle doesn't go off the board
```

```
if(PaddlePos != 1)
```

```
{
```



```

    PaddlePos--;
    PaddleColArray[PaddlePos+2]=0;
    PaddleColArray[PaddlePos-1]=4;
}
}

// When the Player presses the "DOWN" Button
void MovePaddleDown()
{
    if(PaddlePos != 14)
    {
        PaddlePos++;
        PaddleColArray[PaddlePos-2]=0;
        PaddleColArray[PaddlePos+1]=4;
    }
}

// Printing Paddles using each Player's Number
void PrintPaddles(uint8_t Player_Num)
{
    if(Player_Num == 2)
    {
        // Each character must have a unique character ID and array to print
        lcd.createChar(0, PaddleColArray);
        lcd.createChar(1, PaddleColArray+8);

        // Move cursor to 15th character on the 1st row
        lcd.setCursor(14, 0);
        lcd.write(byte(0));

        // Move cursor to 15th character on the 1st row
        lcd.setCursor(14, 1);
        lcd.write(byte(1));
    }

    //if(Player_Num == 1)
    else
    {
        lcd.createChar(2, PaddleColArray);
        lcd.createChar(3, PaddleColArray+8);

        lcd.setCursor(1, 0);
        lcd.write(byte(2));
    }
}

```

```

        lcd.setCursor(1, 1);
        lcd.write(byte(3));
    }
}
};

// Creating Objects of Class "Paddle"
Paddle p1, p2;

//-----PRINT_GAME CLASS-----

class Print_Game
{
public:

    // Printing texts when the game starts
    void Start_Game()
    {
        lcd.print(F(" PING PONG GAME "));

        //Set the Cursor at the starting of 2nd row
        lcd.setCursor(0, 1);
        lcd.print(F("PRESS ANY BUTTON"));

        // Variables to record the state of Push-Buttons
        uint8_t p1_UpButState = 0;
        uint8_t p1_DownButState = 0;
        uint8_t p2_UpButState = 0;
        uint8_t p2_DownButState = 0;

        // Waiting till any one of the button is pressed
        while(!(Button_Pressed))
        {
            // Low Level Code for digitalread() function for each input pin
            p1_UpButState = ((PIND & (1 << Player_1_moveUpButton)) );
            p1_DownButState = ((PINB & (1 << (Player_1_moveDownButton-8))) );
            p2_UpButState = ((PIND & (1 << Player_2_moveUpButton)) );
            p2_DownButState = ((PIND & (1 << Player_2_moveDownButton)) );
        }
        lcd.clear();    //Clearing LCD to start the game
    }

    // Printing score each time a player scores
    void Print_Score()

```

```

{
    // Clearing the LCD to print scores
    lcd.clear();
    lcd.print(F("PLAYER1  PLAYER2"));
    lcd.setCursor(3,1);
    lcd.print(p1.Score);
    lcd.setCursor(12,1);
    lcd.print(p2.Score);

    // Scores remain on the display for 2 seconds
    delay(2000);

    // Clearing the display to continue the game
    lcd.clear();
}

// Printing the Winner on Display
void Print_Winner(int Player_Num)
{
    lcd.setCursor(0,0);
    lcd.print(F("  GAME  OVER  "));
    lcd.setCursor(1, 1);
    lcd.print(F("PLAYER "));
    lcd.print(Player_Num);
    lcd.setCursor(11,1);
    lcd.print(F("WINS"));

    // Text remains on screen for 5 seconds
    delay(5000);

    // Resetting the Game
    resetFunc();
}
};

// Creating an Object of Print_Game Class
Print_Game g;

//-----BALL CLASS-----

class Ball
{
    private:

```

```

// Flag to reset the ball and paddles when a point is scored
uint8_t Point_Scored = 0;

// X and Y Direction Components of the Ball
uint8_t ballYDir = 0;    // Ball starts off going horizontal
uint8_t ballXDir = -1;   // Ball starts off going left

// Location of the Ball
uint8_t ballY = Start_Y;
uint8_t ballX = Start_X;

// Row values of Character of the LCD in which the Ball is present
uint8_t ballCharArray[8] = {0, 0, 0, 0, 0, 0, 0, 16};

public:

// Declaring member functions defined outside of class
void GenerateBallArray();

void PrintBall();

void UpdateBall(uint8_t , uint8_t);

void AwardAPoint();
};

//-----Member Functions of Class BALL-----

// Generate the 8 values that make up the character to draw
void Ball :: GenerateBallArray()
{
    for(uint8_t i=0; i<8; i++)
    {
        if(i == (ballY % 8))
        {
            ballCharArray[i] = 2 << (4 - (ballX % 5));
        }
        else
        {
            ballCharArray[i] = 0;
        }
    }
}

```

```

void Ball :: PrintBall()
{
    // Calculate the column we will draw in
    uint8_t LCDCol = ballX / 5;

    // Either the top or bottom row
    uint8_t LCDRow = (ballY <= 7) ? 0 : 1;

    // Assign array to the charNum
    lcd.createChar(4, ballCharArray);

    // Move the cursor into position
    lcd.setCursor(LCDCol,LCDRow);

    // Draw the character
    lcd.write(byte(4));
}

// Updating the Ball's Position
void Ball :: UpdateBall(uint8_t P1_PaddlePos, uint8_t P2_PaddlePos)
{
    // Short wait before update
    // Handling different delays so that
    // ball has same speed when it's going diagonally
    if(ballYDir)      // When it's going Diagonally
    {
        delay(DiagonalballUpdateTime);
    }
    else              // When it's going Horizontally
    {
        delay(HorizontalballUpdateTime);
    }

    //-----CALCULATING BALL'S NEXT POSITION-----

    //If the Ball goes off-board
    if((ballX <= 6) || (ballX >= 73))
    {
        AwardAPoint();
    }

    //If Ball is at the Player 2 edge
    else if(ballX == 72)
    {

```

```

// IF THE BALL STRIKES THE MIDDLE POSITION OF THE PADDLE
if(ballY == P2_PaddlePos)
{
    ballXDir = -1;
}

// IF THE BALL STRIKES THE BOTTOM POSITION OF THE PADDLE
else if(ballY == (P2_PaddlePos + 1))
{
    ballXDir = -1;
    if(ballY == 15) // If the ball strikes the paddle at the bottom corner of the display
    {
        ballYDir = -1;
    }
    else
    {
        ballYDir = 1;
    }
}

// IF THE BALL STRIKES THE TOP POSITION OF THE PADDLE
else if(ballY == (P2_PaddlePos - 1)){
    ballXDir = -1;
    if(ballY == 0) // If the ball strikes the paddle at the upper corner of the display
    {
        ballYDir = 1;
    }
    else
    {
        ballYDir = -1;
    }
}

//If Ball is at the Player 1 edge
else if(ballX == 7)
{
    // IF THE BALL STRIKES THE MIDDLE POSITION OF THE PADDLE
    if(ballY == P1_PaddlePos)
    {
        ballXDir = 1;
    }

    // IF THE BALL STRIKES THE BOTTOM POSITION OF THE PADDLE

```

```

else if(ballY == (P1_PaddlePos + 1)){
    ballXDir = 1;
    if(ballY == 15) // If the ball strikes the paddle at the bottom corner of the
    {
        ballYDir = -1;
    }
    else
    {
        ballYDir = 1;
    }
}

// IF THE BALL STRIKES THE TOP POSITION OF THE PADDLE
else if(ballY == (P1_PaddlePos - 1))
{
    ballXDir = 1;
    if(ballY == 0) // If the ball strikes the paddle at the upper corner of the display
    {
        ballYDir = 1;
    }
    else
    {
        ballYDir = -1;
    }
}

// If the Ball Hit top or bottom of display then change Y Direction
else if((ballY == 0) || (ballY == 15))
{
    ballYDir *= -1;
}

//Resetting the Ball and Paddles IF a point is scored
if(Point_Scored == 1)
{
    // Resetting the Ball
    ballX = Start_X;
    ballY = Start_Y;
    ballXDir *= -1;
    ballYDir = 0;

    // Resetting the Paddles
    p1.PaddlePos = 6;

```

```

p2.PaddlePos = 6;

for(uint8_t i=0; i<16; i++)
{
    if((i==5) || (i==6) || (i==7))
    {
        p1.PaddleColArray[i] = 4;
        p2.PaddleColArray[i] = 4;
    }
    else
    {
        p1.PaddleColArray[i] = 0;
        p2.PaddleColArray[i] = 0;
    }
}

Point_Scored = 0;    //Resetting the Point_Scored Flag
}

// Delete Ball from it's Current Location
uint8_t LCDCol = ballX / 5;
uint8_t LCDRow = (ballY <= 7) ? 0 : 1;
lcd.setCursor(LCDCol, LCDRow);
lcd.print(" ");

// Change ball's position based on direction set on X & Y
ballX += ballXDir;
ballY += ballYDir;

// Create the array for the ball character
GenerateBallArray();

// Printing ball after calculating the new location
PrintBall();
}

// Increase player's score each time one scores
void Ball :: AwardAPoint()
{
    if(ballX <= 8)    // When Player 2 Scores
    {
        p2.Score++;
    }
}

```



```

else          // When Player 2 Scores
{
    p1.Score++;
}

// When one of the player reaches the winning score of 10

if(p1.Score == 10)      // If Player 1 reaches 10 first
{
    g.Print_Winner(Player_1); // Printing Player 1 as Winner
}

else if(p2.Score == 10)  // If Player 2 reaches 10 first
{
    g.Print_Winner(Player_2); // Printing Player 2 as Winner
}

// Printing the Score of both players
g.Print_Score();

Point_Scored = 1;      // Setting the Flag to reset the locations of Paddles and Ball
}

// Creating an Object of class BALL
Ball b;

//-----SETUP-----

void setup()
{
    // The display has 2 rows with 16 characters per row
    lcd.begin(16, 2);

    // Setup buttons so we can receive input
    DDRD &= ~(1<<Player_1_moveUpButton);
    // Low level version of the statement: pinMode( Player_1_moveUpButton, INPUT);
    // Similarly, setting all other pins to input
    DDRD &= ~(1<<Player_1_moveDownButton);
    DDRD &= ~(1<<Player_2_moveUpButton);
    DDRD &= ~(1<<Player_2_moveDownButton);

    // Printing the first statement
    g.Start_Game();
}

```

```

// Print paddles on LCD
p1.PrintPaddles(Player_1);
p2.PrintPaddles(Player_2);

// Print Ball on LCD
b.PrintBall();

// Enabling Pin Change Interrup for Player 1's "UP" Button
PCMSK2 |= (1 << PCINT21);
PCICR |= (1 << PCIE2);

// Enabling Pin Change Interrup for Player 1's "DOWN" Button
PCMSK0 |= (1 << PCINT2);
PCICR |= (1 << PCIE0);

// Enabling Global interrupts
sei();

//Player 2 will input through Interrupts
attachInterrupt(digitalPinToInterrupt(Player_2_moveDownButton), P2_Move_Down, RISING);
attachInterrupt(digitalPinToInterrupt(Player_2_moveUpButton), P2_Move_Up, RISING);
}

//-----LOOP-----

void loop()
{
    //Printing Both Paddles
    p1.PrintPaddles(Player_1);
    p2.PrintPaddles(Player_2);

    //Updating Ball's Position
    b.UpdateBall(p1.PaddlePos, p2.PaddlePos);
}

//----INTERRUPT SERVICE ROUTINES----FOR PLAYER 2-----

void P2_Move_Down()
{
    p2.MovePaddleDown();
}

void P2_Move_Up()
{

```

```
    p2.MovePaddleUp();  
}
```

```
//----PIN CHANGE INTERRUPT SERVICE ROUTINES----FOR PLAYER 1----
```

```
// Interrupt Routine for Player 1 "UP" Button
```

```
ISR(PCINT2_vect)
```

```
{  
    // The if statements make sure that interrupts does something  
    // only on rising edge of the clock  
    if(x_Up)  
    {  
        p1.MovePaddleUp();  
        x_Up = false;  
    }  
    else  
    {  
        x_Up = true;  
    }  
}
```

```
// Interrupt Routine for Player 1 "DOWN" Button
```

```
ISR(PCINT0_vect)
```

```
{  
    // The if statements make sure that interrupts does something  
    // only on rising edge of the clock  
    if(x_Down)  
    {  
        p1.MovePaddleDown();  
        x_Down = false;  
    }  
    else  
    {  
        x_Down = true;  
    }  
}
```

