JOYSTICK

What is our GOAL for this CLASS?

In this class, we learned about the joystick component. We also learned how a joystick works. Using this knowledge, we created a project to animate an LED Dot Matrix display.

What did we ACHIEVE in the class TODAY?

- Learned about the joystick component.
- Understood the difference between analog signal and digital signal.
- Learned to light up a dot on the LED Dot Matrix display depending on the joystick.

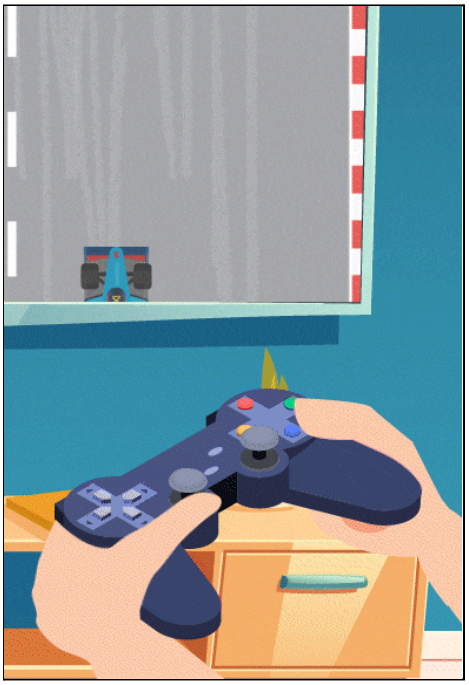
Which CONCEPTS/ CODING BLOCKS did we cover today?

- Concepts : conditional statements, functions. joysticks.
- Coding blocks : **if** statements, functions from ezButton library.

How did we DO the activities?

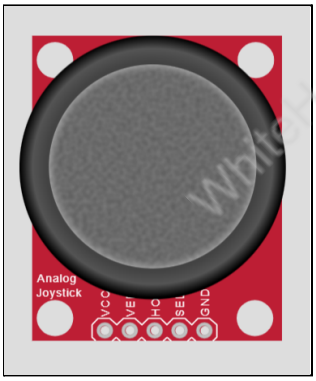
1. Learned about **joystick**:

   In the video game controllers, there is usually a joystick. Joysticks are used to control the character's / vehicle's direction of movement. Joysticks are also used in aircraft to control various aspects of the flight.



Click here to see the reference video.

In wokwi simulator, the Analog Joystick looks like the following image-



Notice that the component name is **Analog Joystick**.

Till now, we have worked with digital signals where the states are HIGH/LOW. But sometimes the values of a signal are in a range instead of just being ON/OFF. Think about light, radio waves, sound waves. It's not either in ON or OFF state. It is continuous. These are all analog signals.

Let's look at the pins of the joystick component.

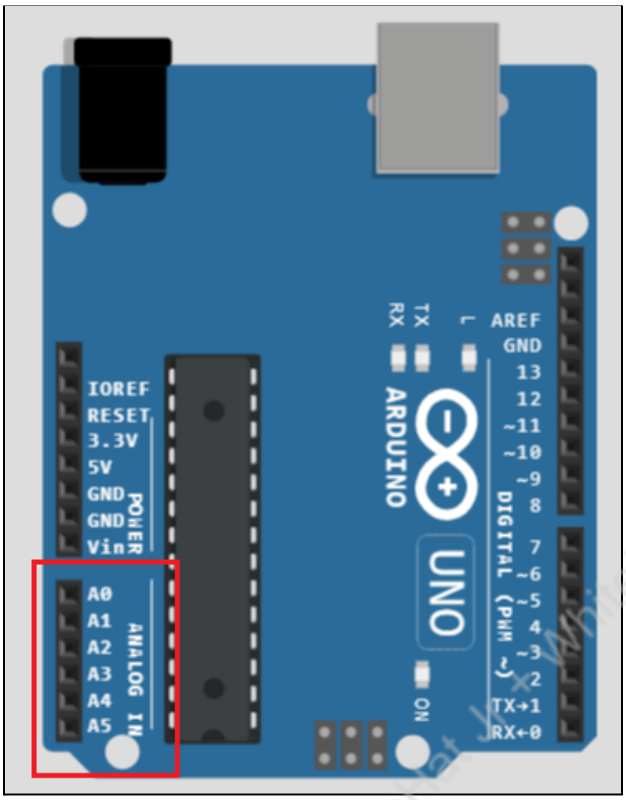
Pin Name	Description
VCC	Voltage supply
VERT	represents how much we have moved the joystick on the y-axis / vertical axis. It is an analog pin.
HORZ	represents how much we have moved the joystick on the x-axis / horizontal axis. It is an analog pin.
SEL	represents the pin to control the push button. This is a digital pin.
GND	Ground

2. Created the circuit:

    a. Download the boilerplate code from here.

    - 1 x **Arduino Uno board** (already added in the boilerplate)
    - 1 x **LED Dot Matrix with MAX7219 Controller** (already added and connected to the Arduino Uno board in the boilerplate)
    - 1 x **Analog Joystick**

    b. Let's observe the Arduino Uno board. We can see that there is a separate section for Analog pins

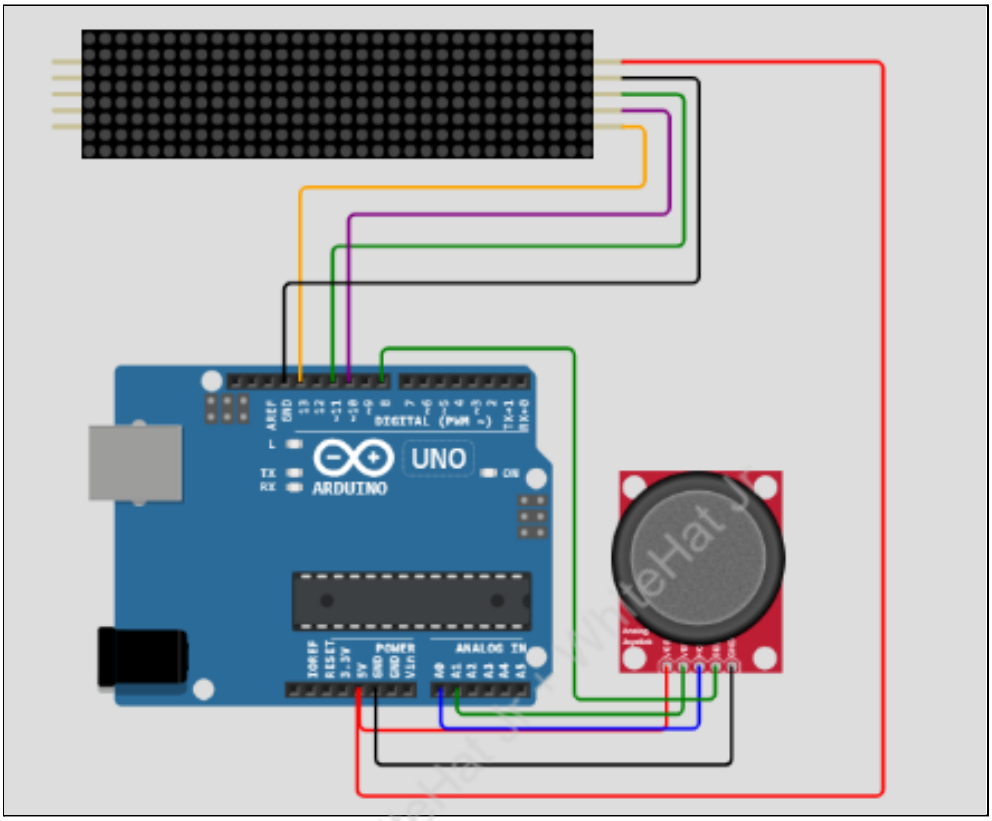


All the analog inputs need to be connected to this section. In a joystick, the **HORZ** and **VERT** pins give us analog value. So, connect those 2 pins here.

c. Let's connect:

Arduino Uno board	Analog Joystick Pin Number
5V	VCC
Ground	GND
A0	VERT
A1	HORZ
GPIO 8	SEL

Reference circuit:



3. Code:

   a. Go to **sketch.ino.** Define two pins **vpin** and **hpin** and assign A0 and A1 to it-

```
const byte vpin = A0;
const byte hpin = A1;
```

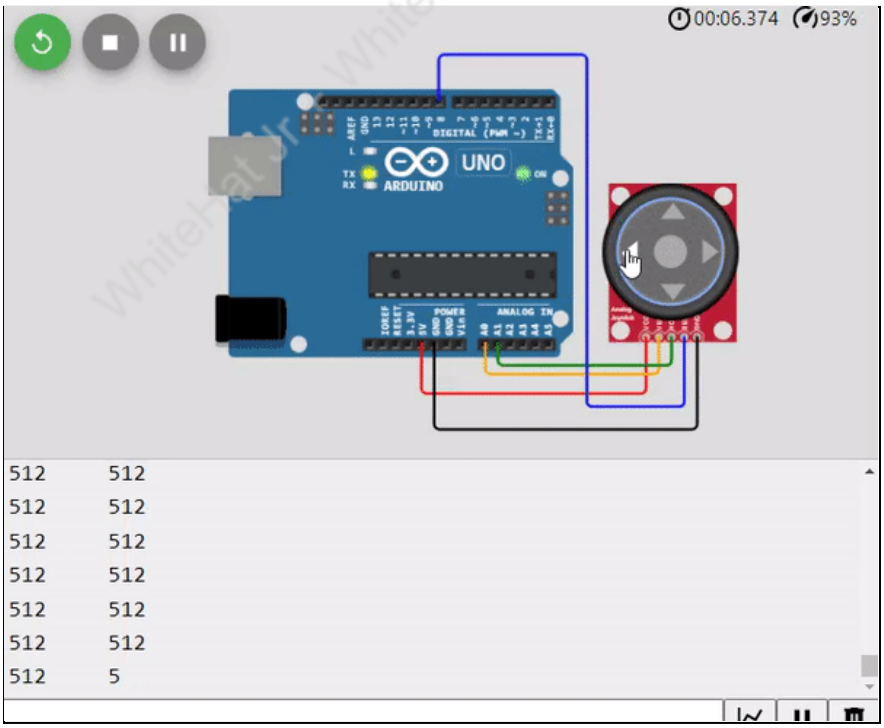
   b. We use the function **analogRead()** to read values from the analog pins. We print it in the Serial monitor to observe the output.

```
void loop() {
  // put your main code here, to run repeatedly:
  Serial.print(analogRead(hpin));
  Serial.print('\t');
  Serial.println(analogRead(vpin));
}
```

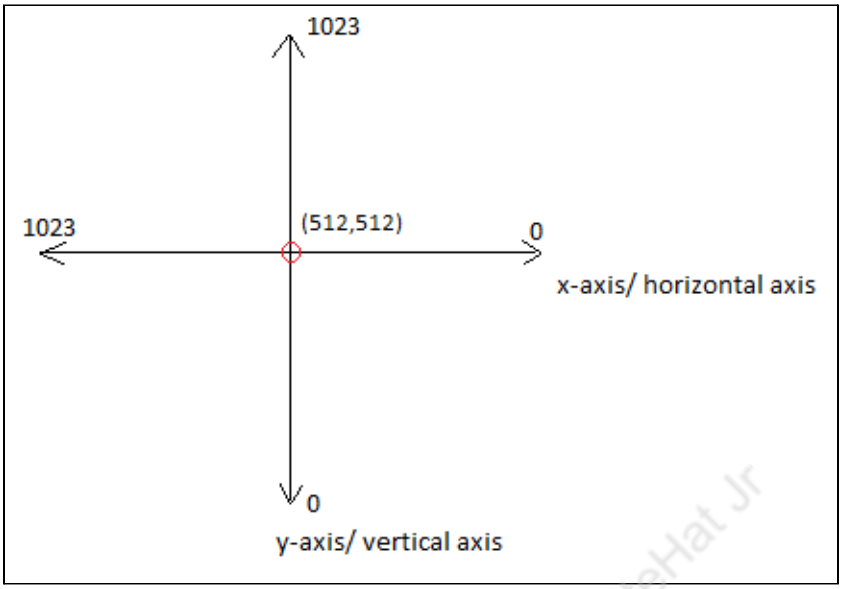
   c. Your code should look like this -

```
const byte vpin = A0;
const byte hpin = A1;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  delay(25);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.print(analogRead(hpin));
  Serial.print('\t');
  Serial.println(analogRead(vpin));
}
```

d. Let's observe the output-



Now, observe that the value ranges between 0 to 1023 i.e. total 1024 values or 2^{10} values.



Here, when the joystick is at center the values are (512,512).

If it's on the left, the values are - (1023,512).

If it's on the right, the values are - (0,512).

If it's on the up, the values are - (512,1023).

If it's on the down, the values are - (512,0).

Observe that instead of being HIGH/LOW, the values are in a range of 0-1023. That's why it is an analog signal.

e. Let's define 5 variables - **head_x**, **head_y**, **direction**, **prev_ direction**, **max_x** and **max_y**. Here, **head_x** and **head_y** will hold the joystick's position. We will assign the direction variable a value depending on the joystick's position. The **max_x** and **max_y** variable will hold the maximum column and row number respectively for the LED Dot Matrix display.

```
int head_y = 0;
int head_x = 0;

String direction="" , prev_direction = "";
int max_x = 31, max_y = 7;
```

f. Now, let's write the following code inside the **setup()** function.

- Initiate the module.

```
matrix.begin();
```

- Clear the Dot Matrix display, if there is anything by using the following function-

  matrix.clear();

- Use the **setPoint()** function to light up the x and y position on the LED.

  matrix.setPoint(head_y, head_x, true);

g. Let's define a new function named **check_direction().**

This function will read the horizontal and vertical positions of the joystick. Depending on these values, we will assign a value to the direction variable.

- Write an **if** statement. If the hpin value of the joystick is more than 512, the direction will be "**left**".
- If the hpin value of the joystick is less than 512, the direction will be "**right**".
- Similarly, if the vpin value of the joystick is more than 512, the direction will be "**up**".
- And if the vpin value of the joystick is less than 512, the direction will be "**down**".

```
void check_direction(){

  if(analogRead(hpin)>512)
    direction="left";
  else if(analogRead(hpin)<512)
    direction="right";
  else if(analogRead(vpin)>512)
    direction="up";
  else if(analogRead(vpin)<512)
    direction="down";

}
```

h. Define another function named **move_sprite().**

This function will check the **direction** variable and will change **x** and **y** variables accordingly.

- First, write an **if** statement. If the direction variable's value is "**left**", increase x by 1.
- If the direction variable's value is "**right**", decrease x by 1.
- If the direction variable's value is **"down"**, increase y by 1.
- If the direction variable's value is "**up"**, decrease y by 1.

```
void move_sprite(){

  if (direction  ==  "left")  head_x++;
  else if (direction  ==  "right")  head_x--;
  else if (direction  ==  "up") head_y--;
  else if (direction  ==  "down") head_y++;

}
```

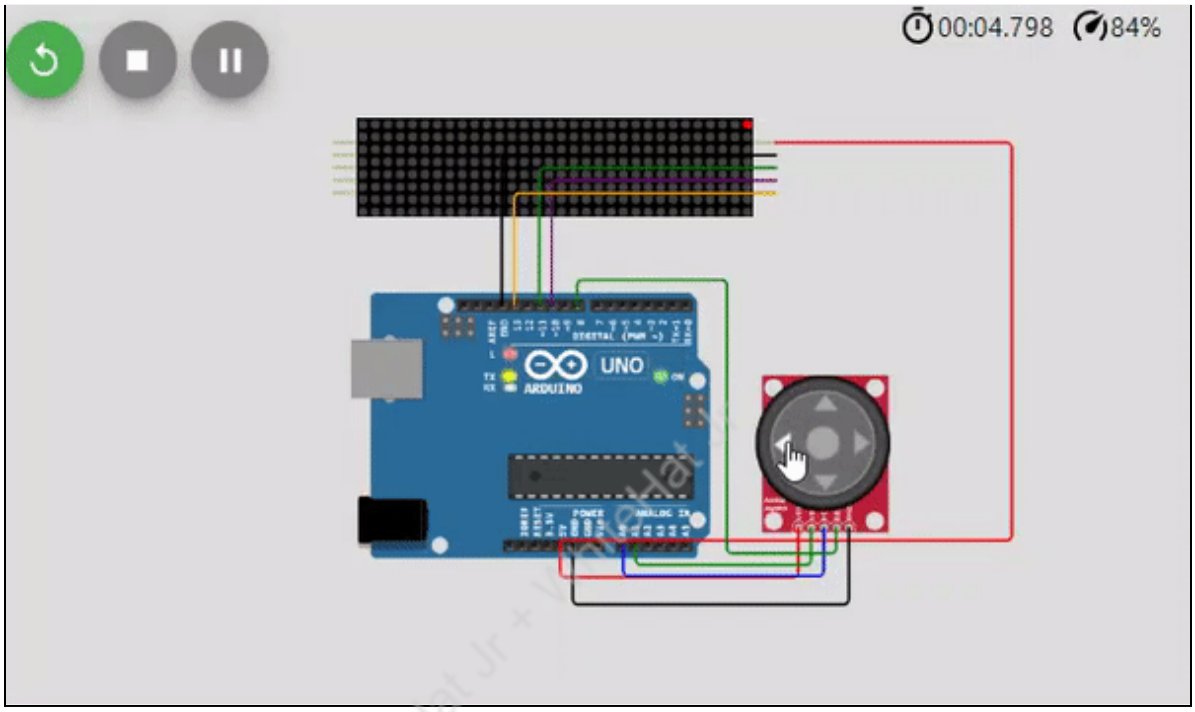
i. In the **loop()** method, call the **check_direction()** and **move_sprite()** methods.

```
void loop(){


  check_direction();
  move_sprite();

}
```

j. map the **head_x**, **head_y** position on the LED Dot Matrix display.

```
void loop(){


  check_direction();
  move_sprite();

  matrix.setPoint(head_y, head_x, true);
}
```

k. Observe the output now.



Click here to view the reference video.

There are 2 problems now. Let's resolve these problems.

Problem-1:

The next LEDs are lit up very fast. It needs to be a little slower.

Add a **delay()** of 200 milliseconds to resolve this problem.

```
void loop(){


  check_direction();
  move_sprite();

  matrix.setPoint(head_y, head_x, true);
  delay(200);
}
```

Problem-2:

The x, y increases or decreases infinitely. So, it goes out of the LED Dot Matrix.

Vertically- the y value will range between 0 to 7.

Horizontally- the x value will range between 0 to 31 as 4 LED Dot Matrix units are daisy-chained together to make a larger display.

But, make sure that when the **head_x** reaches any boundary, it should come out of the opposite side. So, if it crosses the last column from the left, it should enter from the right.

Now, write a new function named **window_check().**

```
void window_check(){

  //  window exceed check
  if (head_x > max_x)head_x = 0;
  else if (head_x < 0)head_x = max_x;
  else if (head_y > max_y)head_y = 0;
  else if (head_y < 0)head_y = max_y;

}
```

Let's call the **window_check()** function at the end of the **move_sprite()** function.

I. Let's say if the movement of the sprite is towards "**left**", the next movement should either be "**up**" or "**down**". It should not go back to the exact opposite direction, then we won't be able to observe the movement.

To resolve this problem, at the end of the **check_direction()** method, we will store the **direction** in the variable named **prev_direction**.

```
void check_direction(){

  if(analogRead(hpin)>512)
    direction="left";
  else if(analogRead(hpin)<512)
    direction="right";
  else if(analogRead(vpin)>512)
    direction="up";
  else if(analogRead(vpin)<512)
    direction="down";

  //  updating prev_direction
  prev_direction = direction;

}
```

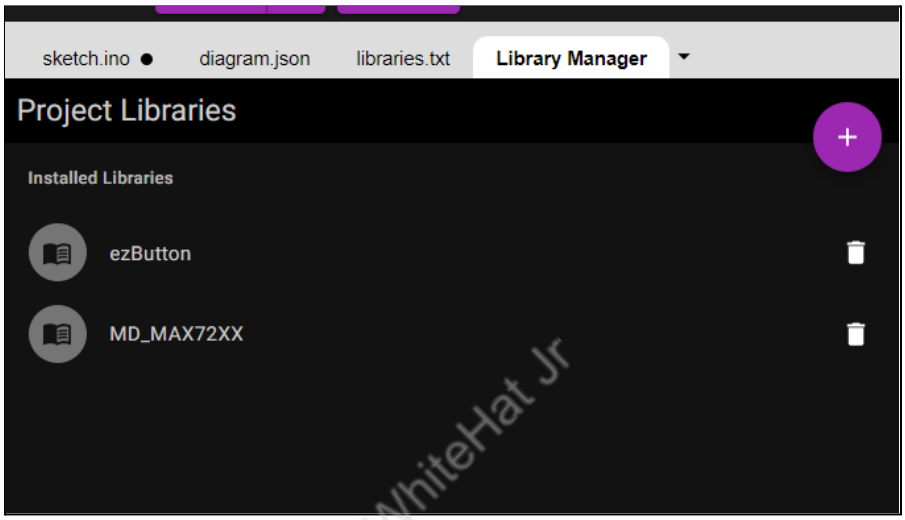
Let's add the conditions now,

```
void check_direction(){

  if(analogRead(hpin)>512 &&  prev_direction  !=  "right")
    direction="left";
  else if(analogRead(hpin)<512 &&  prev_direction  !=  "left")
    direction="right";
  else if(analogRead(vpin)>512 &&  prev_direction  !=  "down")
    direction="up";
  else if(analogRead(vpin)<512 &&  prev_direction  !=  "up")
    direction="down";

  //  updating prev_direction
  prev_direction = direction;

}
```

m. Now, stop the animation once the push button on the joystick is pressed.

To do that, first let's import a header file named "**ezButton.h**"

- Go to the **Library Manager** and import the **ezButton** library.



- Now, in the **sketch.ino** file, include the **ezButton.h** file.

```
#include <ezButton.h>
```

- Let's initiate an ezButton object that attaches to pin 8.

```
ezButton button(8);
```

- Initiate a new variable named **flag** to 0. We will change the flag value to 1, once the pushbutton is pressed.

```
#include <MD_MAX72xx.h>
#include <ezButton.h>

//  matrix controls
const byte data_pin = 11;
const byte chip_select_pin
const byte clock_pin = 13;
const byte max_devices = 4

ezButton button(8);

int y = 0;  //  initial po
int x = 0;
int flag=0;

String direction = "";
```

- Call **button.loop()** method inside the **loop()** method. Call this method to make the button work.

```
void loop(){

  button.loop();
```

- Set the debounce time to 25 milliseconds. Pushbuttons generate spurious transitions when pressed. These transitions may be read as multiple presses in a very short time fooling the program. To debounce a pushbutton is to check twice in a short period of time to make sure the pushbutton is definitely pressed.

```
void setup(){

  Serial.begin(9600);

  matrix.begin();
  matrix.clear();

  button.setDebounceTime(25);
  matrix.setPoint(y, x, true);
```

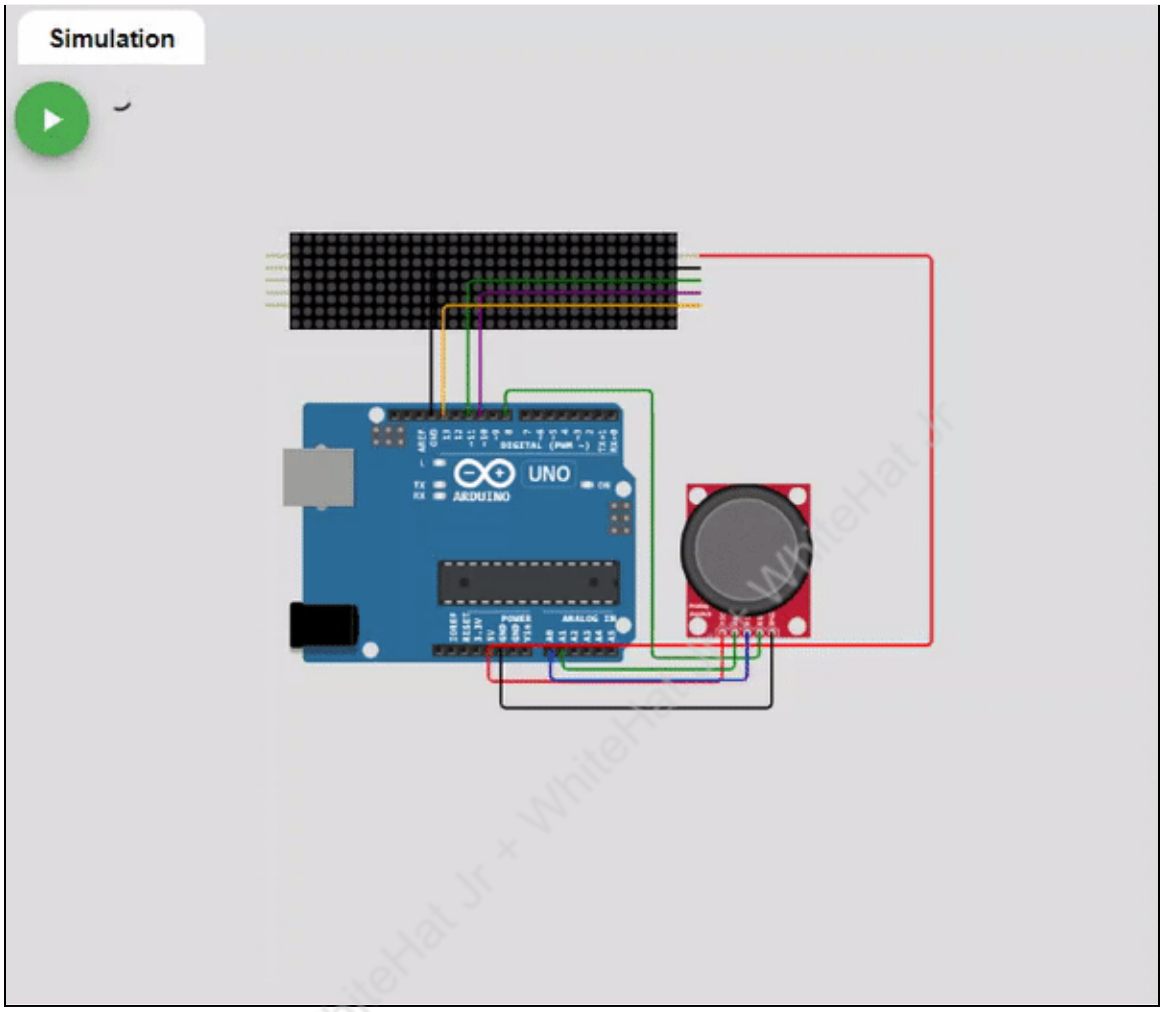
- change the flag input to 1, once the button is pressed.

```
void check_direction(){

  if (direction  ==  "left")x++;
  else if (direction  ==  "right")x--;
  else if (direction  ==  "up")y--;
  else if (direction  ==  "down")y++;

  x = constrain(x , 0 , 31);
  y = constrain(y , 0 , 7);

  if(button.isPressed()){
    Serial.println("The button is pressed");
    flag=1;
  }
}
```

- Now, we want to run the **check_direction()** and **move_sprite()** methods only when the flag is 0. So, let's add an **if** statement.

```
void loop(){
  button.loop();

  if (flag==0)
  {
    check_direction();
    move_sprite();
  }

  //matrix.clear();
  matrix.setPoint(head_y, head_x, true);
  delay(200);
}
```

Reference Output:



Click here to view the reference video.

What's NEXT?

In the **next class**, we will learn to make a game using a joystick.

Expand Your Knowledge

To know more about **joysticks** on arduino**,** click here.