PRO-C271

SNAKE GAME-1

What is our GOAL for this CLASS?

In this class, we started building the snake game. We understood the indices of the LED Dot Matrix component and learned how to use an array to hold these indices. Using this knowledge, we implemented the movement of the snake.

What did we ACHIEVE in the class TODAY?

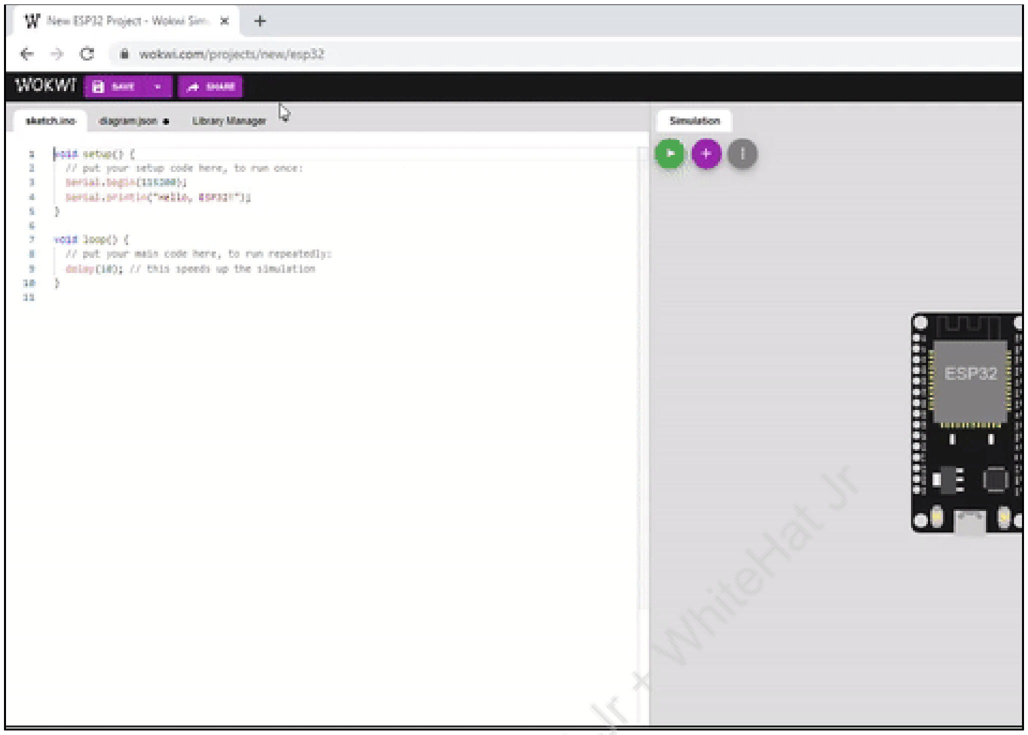
- Understood the indices of the LED Dot Matrix Display
- Learned to use an array to hold the LED position.
- Manipulate LEDs in a way that we can create a moving snake sprite.

Which CONCEPTS/ CODING BLOCKS did we cover today?

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

How did we DO the activities?

1. Open the previous class code. You can also download previous class code here. Unzip the downloaded file and upload it in wokwi project folder.



```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial.println("Hello, ESP32!");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10); // this speeds up the simulation
}
```

Click here to view the reference video.

2. Observed previous class output:

   In the last class, the output looked almost like a length increasing snake.

   Add the **matrix.clear()** in the given code.

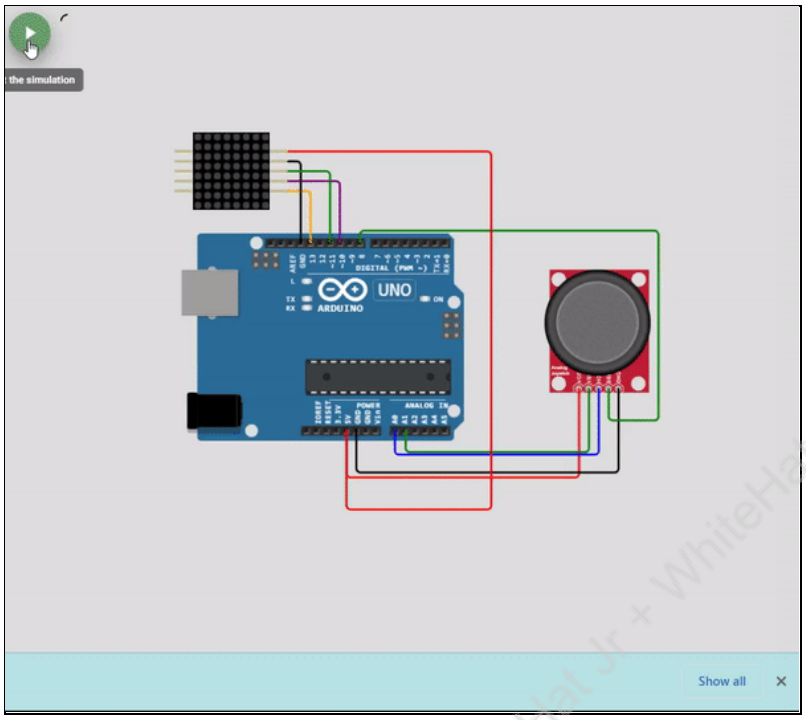
```
void loop(){

  button.loop();
  // put your main code here, to run repeatedly:
  if(flag==0){
    check_direction();
    move_sprite();
  }

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

}
```

Reference output:

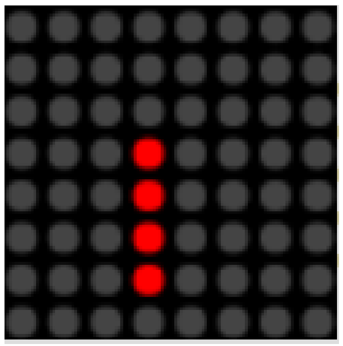


Click here to view the reference video.

Observe that it's just one LED that lights up. The snake-like output was because the LED Dot Matrix was not cleared in every loop. So, the lit up LEDs from the previous loops were still glowing when we lit up the next LED.

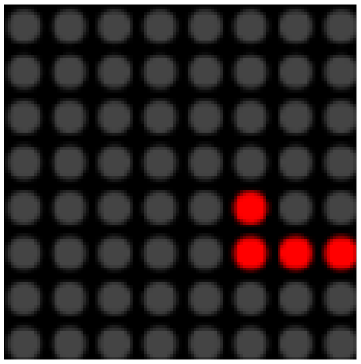
3. Made the snake object:

    a. A snake sprite would be a continuous line of LEDs lit up. e.g.



Here, 4 consecutive LEDs are lit which gives it a snake-like illusion.

Also, this snake will turn.



Assume the current position of the snake is like the image above. We need to store the position of each Dot building the snake's body.

	c7	c6	c5	c4	c3	c2	c1	c0
r0	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)	(0,0)
r1	(1,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)
r2	(2,7)	(2,6)	(2,5)	(2,4)	(2,3)	(2,2)	(2,1)	(2,0)
r3	(3,7)	(3,6)	(3,5)	(3,4)	(3,3)	(3,2)	(3,1)	(3,0)
r4	(4,7)	(4,6)	(4,5)	(4,4)	(4,3)	(4,2)	(4,1)	(4,0)
r5	(5,7)	(5,6)	(5,5)	(5,4)	(5,3)	(5,2)	(5,1)	(5,0)
r6	(6,7)	(6,6)	(6,5)	(6,4)	(6,3)	(6,2)	(6,1)	(6,0)
r7	(7,7)	(7,6)	(7,5)	(7,4)	(7,3)	(7,2)	(7,1)	(7,0)

In this case, the 4 LEDs are (4,2), (5,2), (5,1), (5,0).

Use two arrays -one array will hold all the x values and another one will hold all the y values. In C, an array stores homogeneous values i.e. values of the same data-type.

b. To make it a snake-like sprite, initiate two arrays. These 2 arrays will hold the x and y coordinates of each dot in the snake's body.

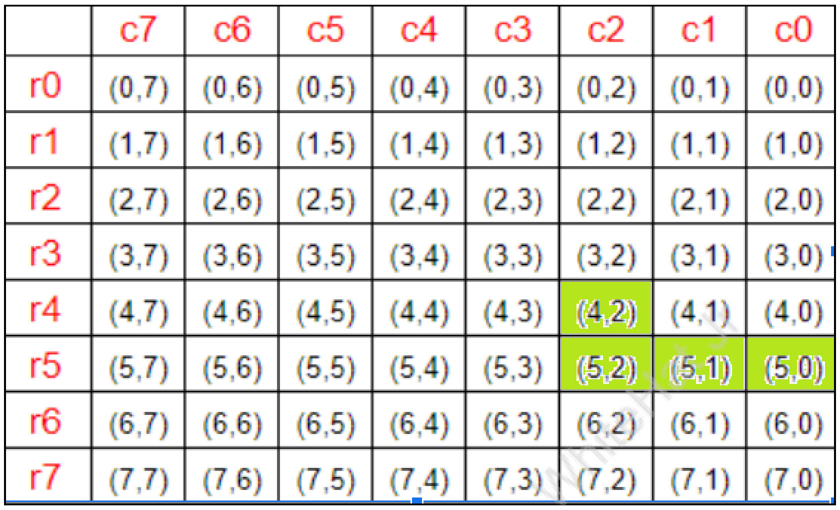
```
int snake_x[64];

int snake_y[64];
```

c. Add a variable to store the length of the snake.

```
int snake_length = 0;
```

d. Store each position of the snake in the array.



	c7	c6	c5	c4	c3	c2	c1	c0
r0	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)	(0,0)
r1	(1,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)
r2	(2,7)	(2,6)	(2,5)	(2,4)	(2,3)	(2,2)	(2,1)	(2,0)
r3	(3,7)	(3,6)	(3,5)	(3,4)	(3,3)	(3,2)	(3,1)	(3,0)
r4	(4,7)	(4,6)	(4,5)	(4,4)	(4,3)	(4,2)	(4,1)	(4,0)
r5	(5,7)	(5,6)	(5,5)	(5,4)	(5,3)	(5,2)	(5,1)	(5,0)
r6	(6,7)	(6,6)	(6,5)	(6,4)	(6,3)	(6,2)	(6,1)	(6,0)
r7	(7,7)	(7,6)	(7,5)	(7,4)	(7,3)	(7,2)	(7,1)	(7,0)

To understand this, let's take the last example, snake_x and snake_y should look like this -

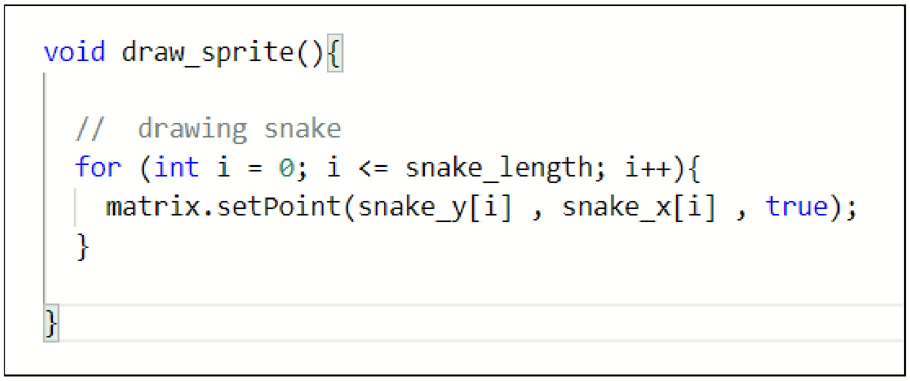
snake_x= [2, 2, 1, 0];

snake_y=[4, 5, 5, 5];

Now, these LEDs need to light up to show the snake object on the screen.

For that we write a function named **draw_sprites()**; To light up an LED, we use the **setPoint()** method.

This code can be shortened using a for loop -



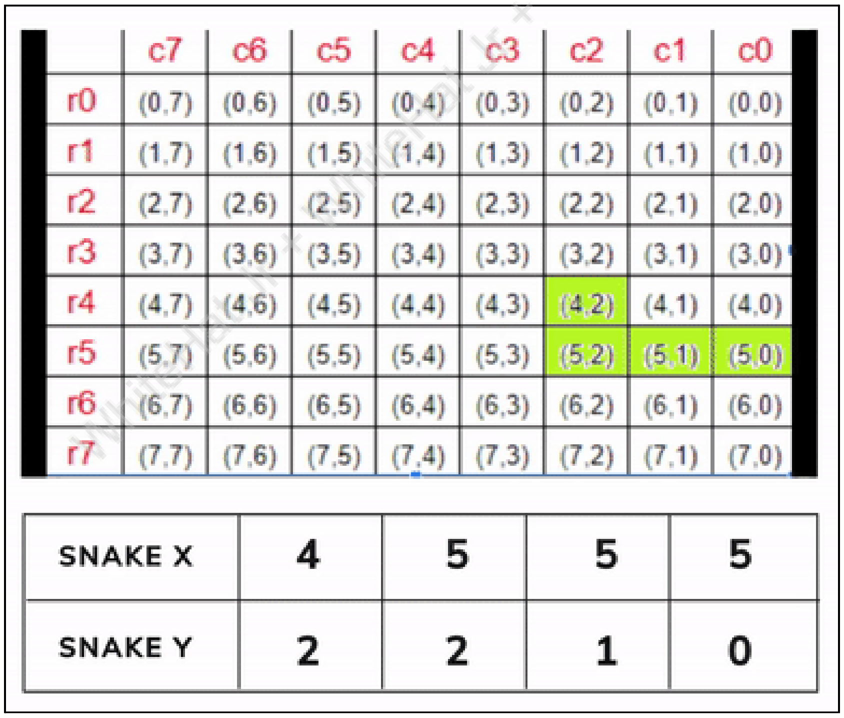
```
void draw_sprite(){

  //  drawing snake
  for (int i = 0; i <= snake_length; i++){
    matrix.setPoint(snake_y[i] , snake_x[i] , true);
  }

}
```

e. Call the **draw_sprites()** method in the **loop()** method. Delete the **matrix.setPoint()** method from the **loop()** method.

f. Now, let's try to understand how the movement of the snake should be.



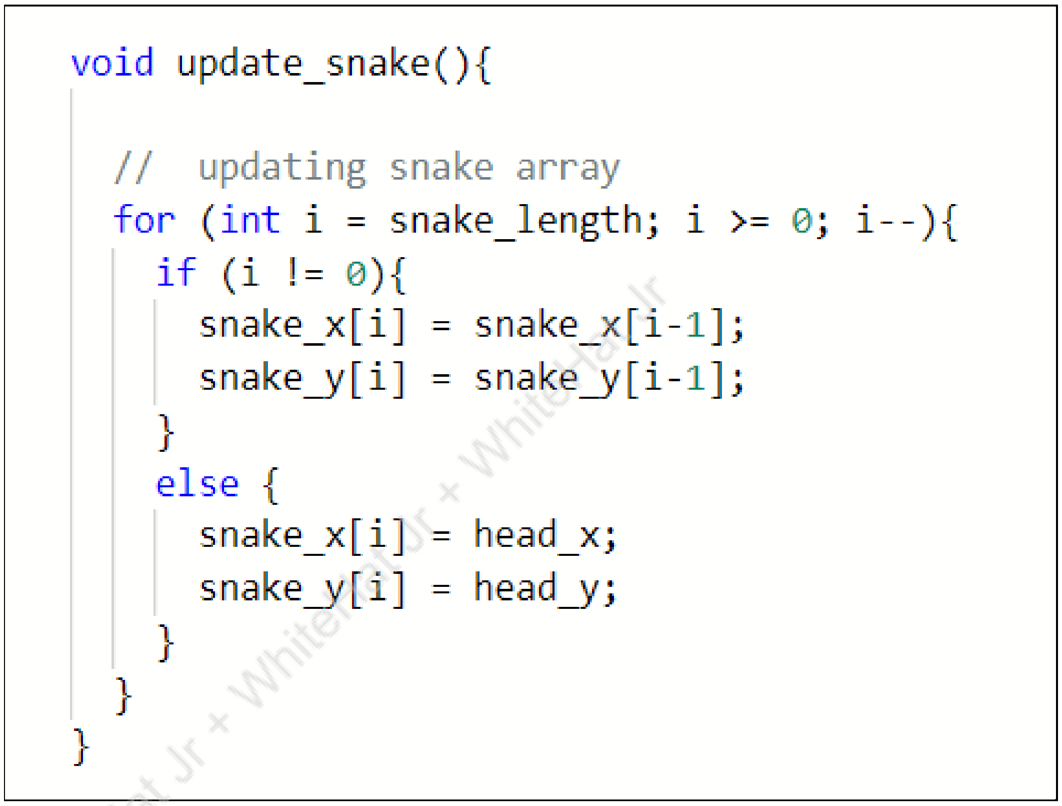
	c7	c6	c5	c4	c3	c2	c1	c0
r0	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)	(0,0)
r1	(1,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)
r2	(2,7)	(2,6)	(2,5)	(2,4)	(2,3)	(2,2)	(2,1)	(2,0)
r3	(3,7)	(3,6)	(3,5)	(3,4)	(3,3)	(3,2)	(3,1)	(3,0)
r4	(4,7)	(4,6)	(4,5)	(4,4)	(4,3)	(4,2)	(4,1)	(4,0)
r5	(5,7)	(5,6)	(5,5)	(5,4)	(5,3)	(5,2)	(5,1)	(5,0)
r6	(6,7)	(6,6)	(6,5)	(6,4)	(6,3)	(6,2)	(6,1)	(6,0)
r7	(7,7)	(7,6)	(7,5)	(7,4)	(7,3)	(7,2)	(7,1)	(7,0)

SNAKE X	4	5	5	5
SNAKE Y	2	2	1	0

Click here to view the video.

- When the snake goes forward in any direction, shift each value in the **snake_x** and **snake_y** to the next index.

- Write a **for** loop, in which we shift each value in the **snake_x** and **snake_y** to the next index. And we assign the 0th index value to **head_x** and **head_y**.



```
void update_snake(){

  //  updating snake array
  for (int i = snake_length; i >= 0; i--){
    if (i != 0){
      snake_x[i] = snake_x[i-1];
      snake_y[i] = snake_y[i-1];
    }
    else {
      snake_x[i] = head_x;
      snake_y[i] = head_y;
    }
  }
}
```

g. Now, observe that there is a problem with the output. Sometimes when the joystick is clicked, the snake doesn't move towards the given direction. it needs to be clicked multiple times to make the snake move towards that direction.

   This happens due to the **delay()** method that is used inside the **loop()** method.

   The **delay()** method makes the **loop()** method wait for a certain time until it runs again. So the **check_direction()** function will not run till the wait time is over and it will not detect the joystick input for that period of time.

   To resolve this, use the **millis()** method.

   millis() method returns the number of milliseconds passed since the

Arduino board began running the current program.

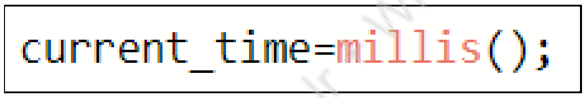
- Define 3 variables named **current_time, prev_time** and **threshold**.

  The datatype for **threshold** can be **int**.

  But **current_time, prev_time** must be **long int** so that these variables can hold larger values.

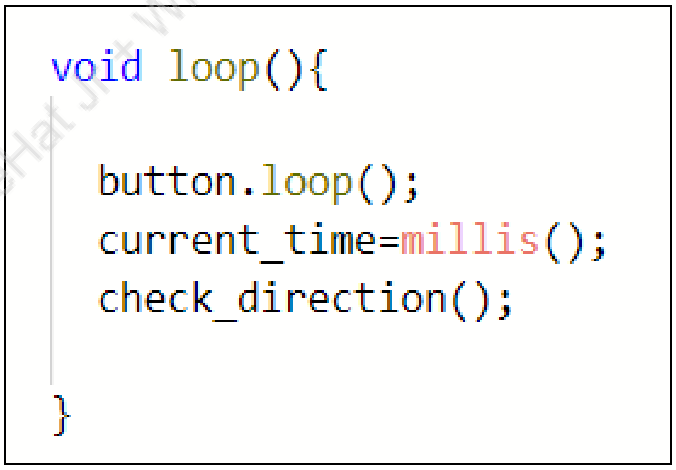
  ```
  long int current_time , prev_time = 0;
  int threshold = 800;
  ```

- Delete the **delay()** method and rewrite the **loop()** method.
- Add the following line of code in **loop()** method.



```
current_time=millis();
```

- After that, call the **check_direction()** method. Now, it will be called every time the **loop()** runs.



```
void loop(){

  button.loop();
  current_time=millis();
  check_direction();

}
```

- Initially the **prev_time** is 0 and the **current_time** holds the number of milliseconds passed since the beginning of the program.

  Deduct **prev_time** from **current_time** and check if the value is greater than or equal to the **threshold** value.

  If it is, we will assign **current_time** as **prev_time.**

This conditional will run every 800 milliseconds now.

```
void loop(){

  button.loop();
  current_time=millis();
  check_direction();

 if (current_time - prev_time  >=  threshold){
    prev_time = current_time;


  }

}
```

- Now, call the **move_sprite()**, **update_snake()** and **draw_sprites()** methods from the conditional block.

h. Generating random food objects:

Initiate 2 variables named **food_x**, **food_y**

```
int food_x = random(0,8);
int food_y = random(0,8);
```

Now, let's show this random food food object by lighting up the LED.

We will write the code in the **draw_sprites()** method.

```
void draw_sprites() {

  matrix.setPoint(food_y , food_x , true);

  for (int i = 0; i <= snake_length; i++) {
    matrix.setPoint(snake_y[i], snake_x[i], true);
  }
}
```

i. Increase the length of the snake:

The next task is when the snake eats the food, its length should increase. Also, the next food object should appear in another random position.

- Define a new method named **food_eat_check().**
- Write an **if** statement to check if the food object's position is the same as the position of the snake's head.

```
if (head_x == food_x && head_y == food_y){

}
```

- Inside this if statement, mention what we want to do once this condition is fulfilled.
  - ➢ Generate a random position for the food again.

```
food_x = random(0,8);
food_y = random(0,8);
```

  - ➢ Also, increase the length of the snake by 1;

```
snake_length++;
```

- Call the **food_eat_check()** method in the **loop()** method.

j. Make the snake faster:

We control the snake's speed by the threshold variable.
Every time the snake eats a food object, we will reduce the threshold by 50.

```
threshold = threshold - 50;
```

Also, we don't want the threshold to be a negative value.

So, we will add a conditional statement which reset the threshold to 80 once it reaches a value less than 100.

```
if (threshold < 100)
    threshold = 80;
```

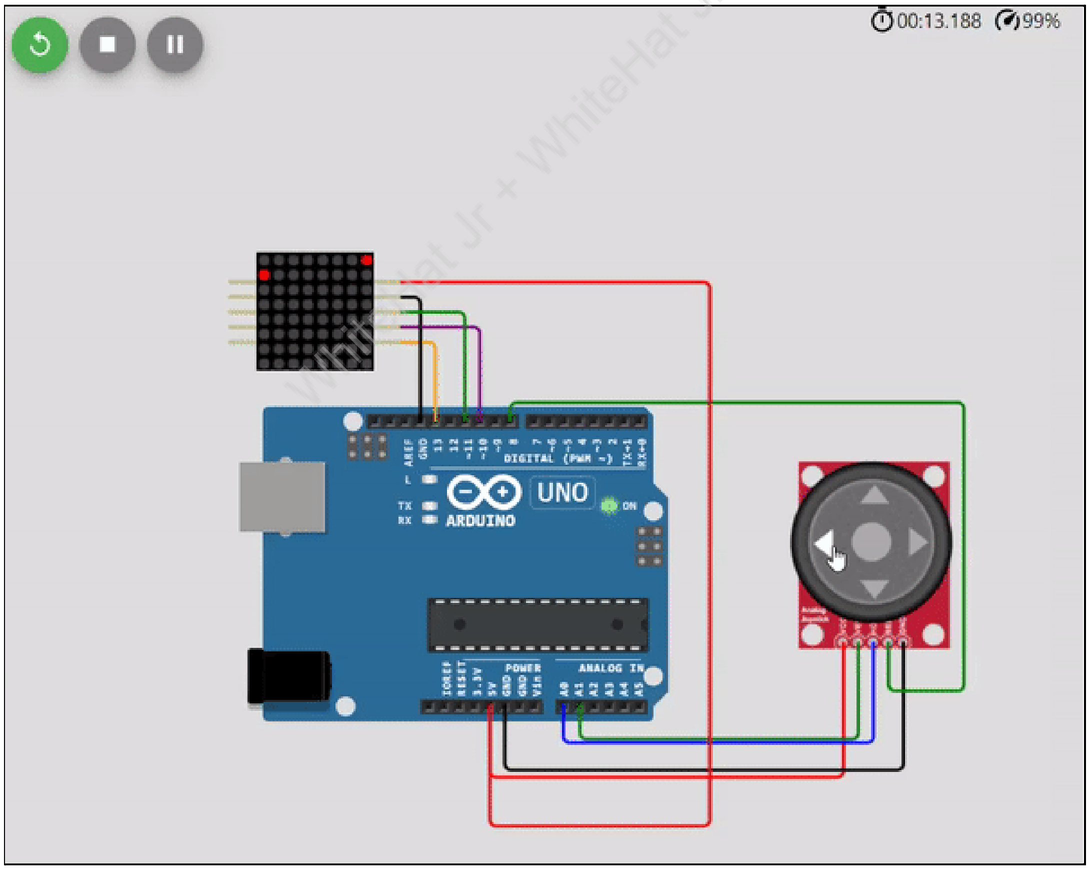
The code should look like this-

```
void food_eat_check() {
  if (head_x  ==  food_x  &&  head_y  ==  food_y) {
    food_x = random(0, 8);
    food_y = random(0, 8);

    snake_length++;

    threshold = threshold - 50;
    if (threshold < 100)
          threshold = 80;
  }
}
```

Reference Output:



Click here to view the reference output.

What's NEXT?

In the **next class**, we will complete the snake game.

Expand Your Knowledge

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