Topic	SNAKE GAME - 1	
Class Description	**Students will start with the snake game. They will understand the indices of the LED Dot Matrix component and learn how to use an array to hold these indices. Using this knowledge, they will implement the movement of the snake.**	
Class	**PRO C271**	
Class time	**45 mins**	
Goal	• Understanding the indices of the LED Dot Matrix Display • Learning to use an array to hold the LED position. • Manipulate LED in a way that we can create a moving snake sprite.	

Resources Required	• Teacher Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen
	  ○ Smartphone
	• Student Resources:
	  ○ Laptop with internet connectivity
	  ○ Earphones with mic
	  ○ Notebook and pen	

Class structure	**Warm-Up** **Teacher-Led Activity** **Student-Led Activity** **Wrap-Up**	**10 mins** **15 mins** **15 mins** **05 mins**

WARM-UP SESSION - 10 mins	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?	**ESR**: Hi, thanks! Yes, I am excited about it!

Following are the WARM-UP session deliverables: ● Greet the student. ● Revision of previous class activities. ● Quizzes.	Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Activity Details **Following are the session deliverables:** ● Appreciate the student. ● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
TEACHER-LED ACTIVITY - 15mins	
Teacher Initiates Screen Share	
● **Understanding how to create the snake.** ● **Writing code to make the snake move.**	
Teacher Action	**Student Action**
Do you remember what we did in the previous class? Do you have any doubts? *If the student has any doubts, clarify the doubts.*	**ESR**: We had learnt about joysticks. We had built a project which helped us create patterns on an LED Dot Matrix unit using a joystick. **ESR:** Varied.

In the class, we will start with a new game. We will call it the **Snake Game.** Let's discuss how the snake game should work. What are the objects that we will need to make in the game?	**ESR:** We will need to make the snake first and we will need to code to make it move.
Great! How should we control the movement of the snake?	**ESR:** We can use the joystick to control it.
Yes. We learned how to control an LED in the Dot Matrix Display in the last class. We will also add food objects which will be positioned randomly. What should happen when the snake eats a food object?	**ESR:** It should grow.
Yes. We will write code to increase the length of the snake too. We have discussed almost all the features of the game. What's left to discuss?	**ESR:** How the game should end.

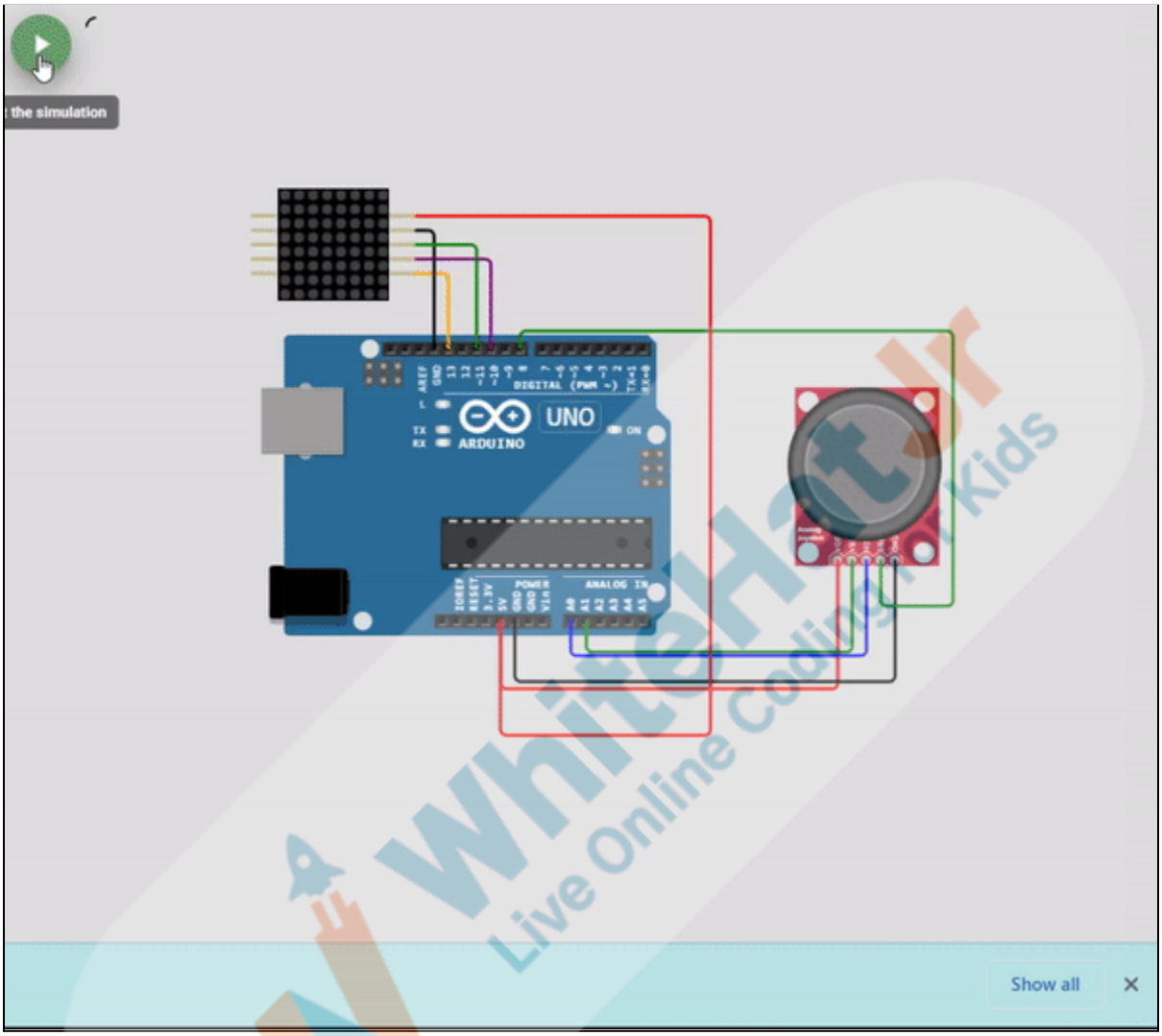
Exactly! There should be winning and losing conditions too. What can be the winning condition? What should be the losing condition? Perfect! Ready to start the new project?	 **ESR:** When the snake eats a certain number of food or it has reached a certain length, it will win the game. **ESR:** When it touches its tail by mistake while moving. **ESR:** Yes. Let's start.
Let's open the code from our previous class. *The teacher downloads the boilerplate code and uploads these files to the wokwi simulator.*	
In the last class, our output looked almost like a length increasing snake, isn't it? Let me show you one thing.	**ESR:** Yes.

```
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);

}
```

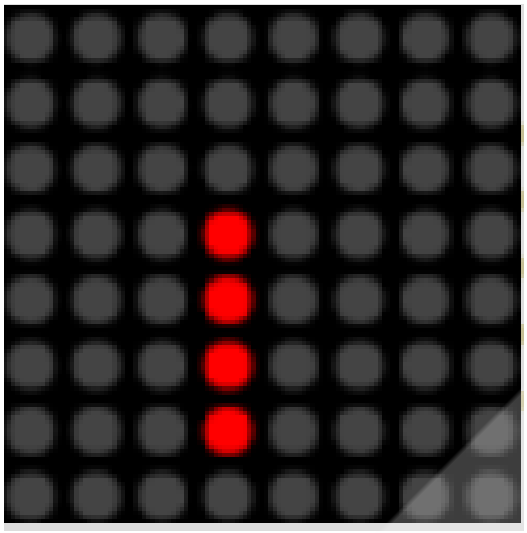


Click here to view the reference video.

We can see that it's just one LED that lights up. The snake-like output was because we didn't clear up our LED Dot Matrix in every loop. So, the lit up LEDs from the previous loops were still glowing when we lit up the next LED.

So, this doesn't look like the snake object that we were expecting.

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

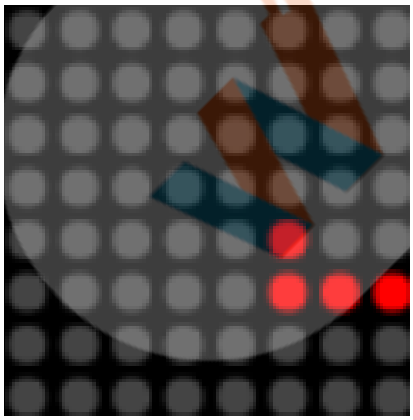


Student can view this image in Student Activity 1

Here, 4 consecutive LEDs are lit which makes it look like a simple snake sprite.

This snake has a length of 4.

Also, this snake should be able to turn.



Students can view this image in Student Activity 2.

Let's assume the current position of the snake is like

the given image. 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).

So, we have multiple values for rows and multiple values for columns.

How can we store multiple values in one variable?

ESR: We can use arrays.

Exactly! In case of the snake position, we will 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.

Also, an array is stored in contiguous memory locations. Contiguous memory locations mean memory locations which are next to each other.

1. To make it a snake-like sprite, we will initiate two arrays.

   ```
   int snake_x[64];

   int snake_y[64];
   ```

   These 2 arrays will hold the x and y coordinates of each dot in the snake's body.

2. We will also need a variable to store the length of the snake. Because we want its length to increase later.

   ```
   int snake_length = 0;
   ```

3. We will store each position of the snake in the array.

   If we take the previous 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.

   So, we will need to write something like the following code-

   ```
   matrix.setPoint( snake_y[0], snake_x[0], true);
   matrix.setPoint( snake_y[1], snake_x[1], true);
   ```

matrix.setPoint(snake_y[2], snake_x[2], true);
matrix.setPoint(snake_y[3], snake_x[3], true);

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);
  }

}
```

4. Call the **draw_sprites()** method in the **loop()** method. Delete the **matrix.setPoint()** method from the **loop()** method.
5. If we try running the code now, we can see that it's not moving.

   Why do you think this is happening?

ESR: Using the **move_sprite()** function, we only updated the **head_x** and **head_y** in the **move_sprite()** function. But the **draw_sprite()** function uses the **snake_y**, **snake_x** array to draw the snake.

Exactly, let's write a function which will update the snake positions in the **snake_x** and **snake_y** array.

6. First, 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

Students can view this video in Student Activity 3

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

- So, we 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;
    }
  }
}
```

Now, your task will be to make sure that if the snake goes beyond the screen, it should re-enter the screen again from the opposite side.

Teacher Stops Screen Share

Can you solve it? Let's try. I will guide you through it.	**ESR:** Yes, sure!

STUDENT-LED ACTIVITY- 15 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

Student Initiates Screen Share

ACTIVITY

- **Writing code to make the snake stay within the LED Dot Matrix display.**
- **Resolve bugs.**

Teacher Action	Student Action
Teacher guides the student to download the boilerplate code from Student Activity 1	Student opens wokwi simulator.
Student runs the project and checks the movement of the snake. Observes the output and code for any probable errors.	
Did you observe any problem? Great observation! Why do you think this happens? Great try! This happens due to the **delay()** method that we have used inside the **loop()** method.	**ESR:** Yes. Sometimes when I click on the joystick, the snake doesn't move towards the given direction. I need to click multiple times to make the snake move towards that direction. **ESR:** Varied.

Let's increase the value passed through the **delay()** method and let's check if it increases the problem. Why do you think this is happening? Exactly! 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. This is a big problem. We want to check the joystick input without any delay, but we want the movement of the snake to be slow with a delay(). We need to resolve this problem.	**ESR:** yes, it does. **ESR:** The **delay()** method makes the **loop()** method wait for a certain time until it runs again. That means it cannot detect the joystick input for that waiting period as well.
We have a function named **millis().** **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.

On Arduino uno, an **int** stores 16-bit value. So, it can hold integers of range **-32,768** (i.e. -2^{15}) **to 32,767** (i.e. 2^{15} -1) **long int** stores 32-bit value. It can hold integers of range **-2,147,483,648** to **2,147,483,647**.

```
long int current_time , prev_time = 0;

int threshold = 800;
```

Let's delete the **delay()** method and rewrite the **loop()** method.

1. Add the following line of code in **loop()** method.

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

2. 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();

}
```

3. Now, initially **prev_time** is 0. And **current_time** holds the number of milliseconds passed since the beginning of the program.

We deduct **prev_time** from **current_time** and we will check if the value is greater than or equals 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;


  }

}
```

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

```
void loop(){

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

 if (current_time - prev_time  >=  threshold){
    prev_time = current_time;
    matrix.clear();  //  clear display before every frame
    move_sprite();  //  move the sprites
    update_snake();  //  updating snake array
    draw_sprite();

  }

}
```

Now, if you check the problem we were facing, it will not be there anymore.

Let's move on to our next task. What should be our next task?	**ESR:** We can make the food objects.
Great! And where should we place the food object?	**ESR:** We can place the food object at a random position.
Let's initiate 2 variables named **food_x**, **food_y**	

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

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

These two variables will now generate a random position for the food initially.

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

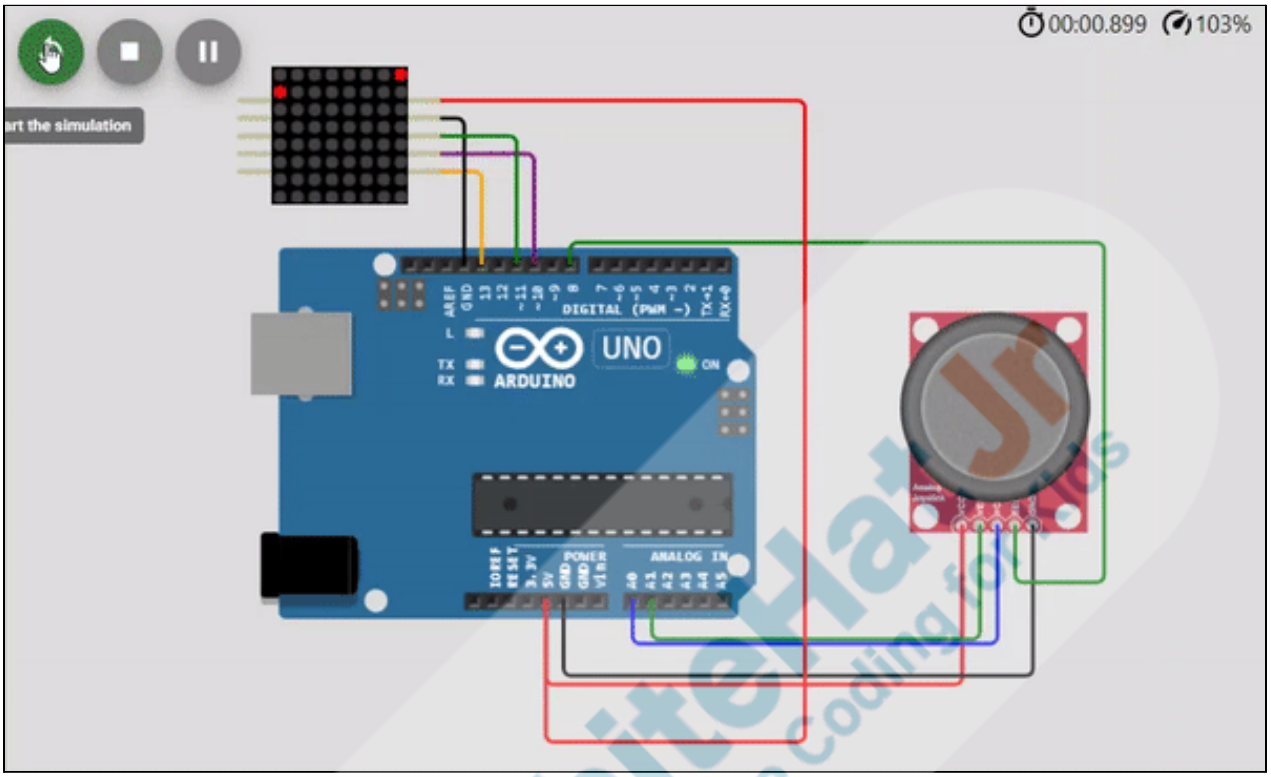
We will write the code in the **draw_sprites()** method.

```
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);
  }
}
```

When you run the code now, you should be able to see a random food object.



Click here to view the reference video.

Now, 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. To achieve this, we will define a new method named **food_eat_check().** But how do we check if the snake has eaten the food? Exactly! Great job.	**ESR:** We can compare the food object's position with the position of the snake's head. If both of these positions are the same, we can say the snake has eaten the food.

Can you try to write the code for it?	**ESR:** sure!

1. Define a new method named **food_eat_check().**
2. 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){`
`}`
3. 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++;`
4. Call the **food_eat_check()** method in the **loop()** method.	

The code should look like this-	

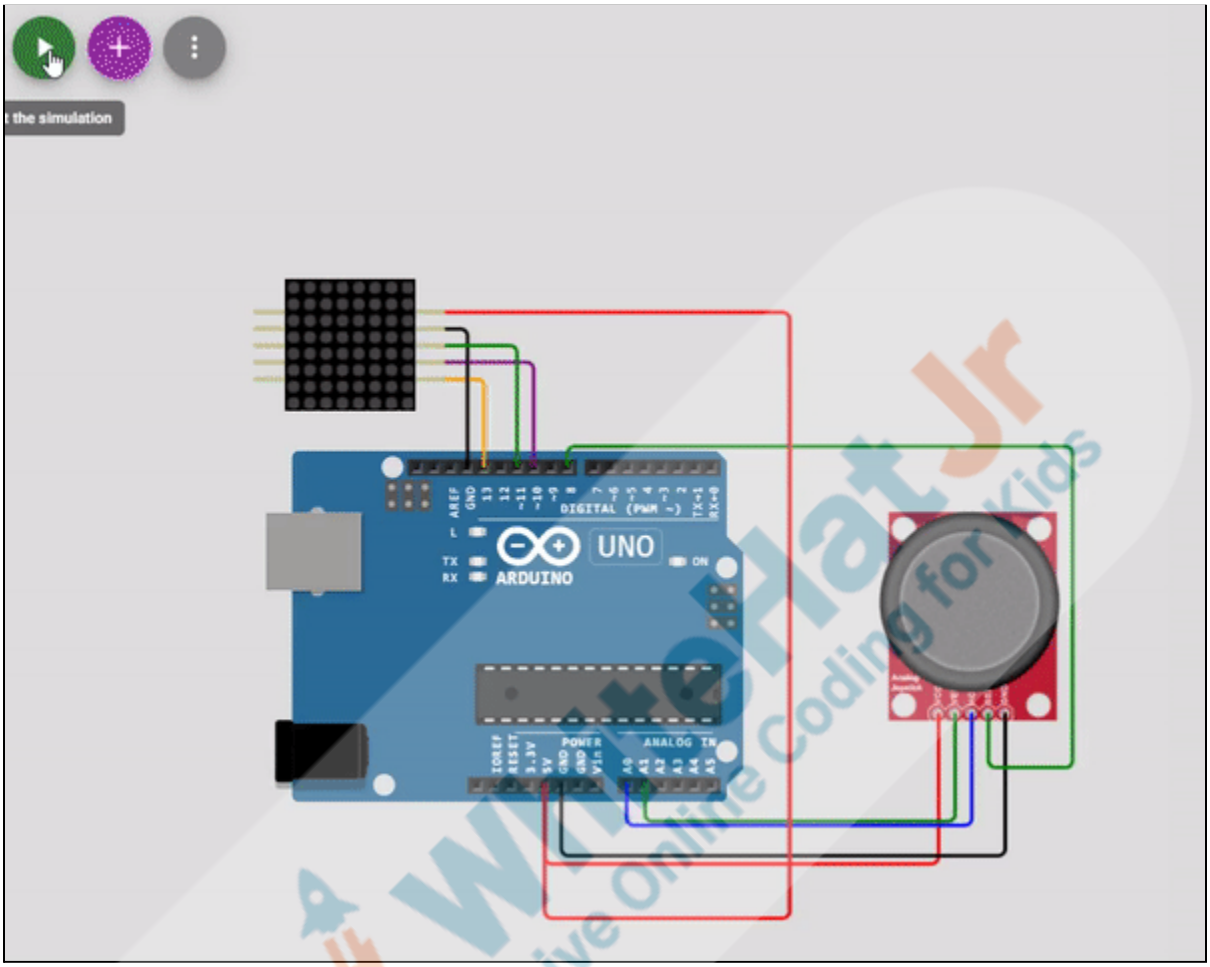
```
void food_eat_check(){

  //  check if head collides with food
  if (head_x  ==  food_x  &&  head_y  ==  food_y){

    //  change the food coordiantes
    food_x = random(0,8);
    food_y = random(0,8);

    //  changing snake_length
    snake_length++;
  }
}
```

Let’s observe the output now-



Click here to view the reference output.

We will add one last thing- we will make the snake move a little faster every time it eats a food object. How did we control the speed of the snake, can you tell me?	
	ESR: Yes. We controlled the speed of the snake using **current_time**, **prev_time** and **threshold** variables.
Exactly. When the **threshold** is 800 milliseconds now. So, our snake is moving one block per 800 milliseconds. What should we do to make it faster?	
	ESR: We can decrease the

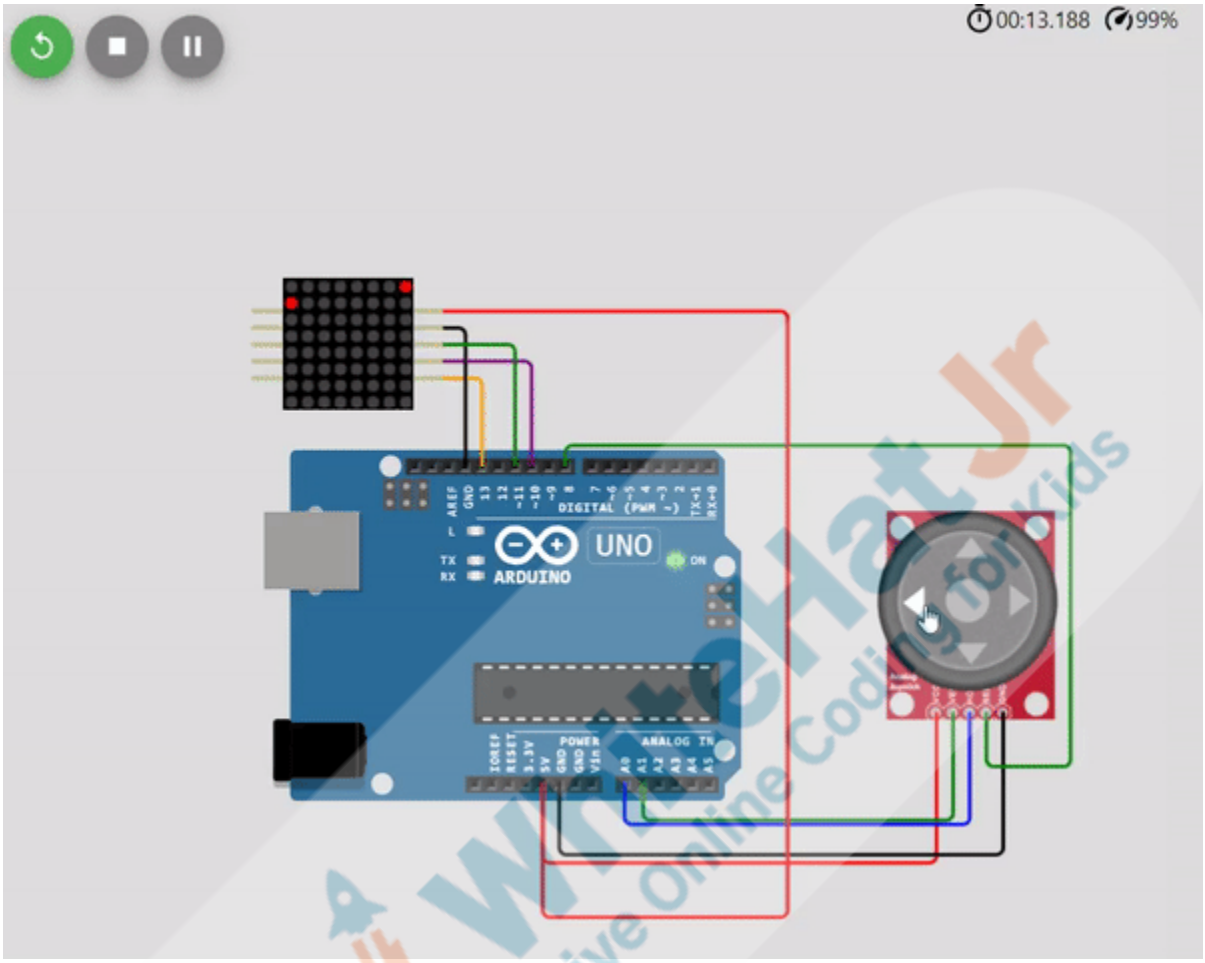
<table>
<tr>
<td>Exactly! Every time the snake eats a food object, we will reduce the threshold by 50.


```
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 lesser than 100.


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

</td>
<td>threshold value.</td>
</tr>
<tr>
<td colspan="2">Reference Output:</td>
</tr>
</table>



Click here to view the reference output.

You have done a great job today! What's left in the snake game? Yes. Great! You will complete this game in the next class.	**ESR:** we need to add winning or losing conditions.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

Activity details

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	Student Action
You get “hats-off” for your excellent work! In the next class, we will complete the snake game.	*Make sure you have given at least 2 hats-off during the class for:* 

PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks ✖ End Class	
ADDITIONAL ACTIVITIES (Optional)	
Additional Activities	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Simulator	https://wokwi.com/
Teacher Activity 2	Boilerplate code for Teacher Activity	https://github.com/procodingclass/PRO-C271-Teacher-Boilerplate
Teacher Reference 1	Teacher Reference code	https://github.com/procodingclass/PRO-C271-Reference-Code
Teacher Reference 2	Project	https://s3-whjr-curriculum-uploads.whjr.online/0e2b4d59-2cf7-4ef3-a600-a8d823689c58.pdf
Teacher Reference 3	Project Solution	https://github.com/procodingclass/PRO-C271-Project-Solution
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/3acf6225-8b08-465a-8450-6342220499bd.pdf
Student Activity 1	Snake on LED Dot Matrix Display example 1	https://s3-whjr-curriculum-uploads.whjr.online/acf9e489-282a-4891-b78f-4a8e20ea07b4.png
Student Activity 2	Snake on LED Dot Matrix	https://s3-whjr-curriculum-uploads.

	Display example 2	whjr.online/0728c4c0-be19-4457-933c-c67bec8a3c30.png
Student Activity 3	Snake movement reference	https://s3-whjr-curriculum-uploads.whjr.online/8b6c664a-475a-4277-9c26-84a433d17366.gif
Student Activity 4	Boilerplate code for Student Activity	https://github.com/procodingclass/PRO-C271-Student-Boilerplate