# SNAKE GAME-FINAL


## What is our GOAL for this CLASS?

In this class, we completed the snake game. We incorporated winning and losing conditions to the game. Additionally, we wrote the code to reset the game once the game is over.

## What did we ACHIEVE in the class TODAY?

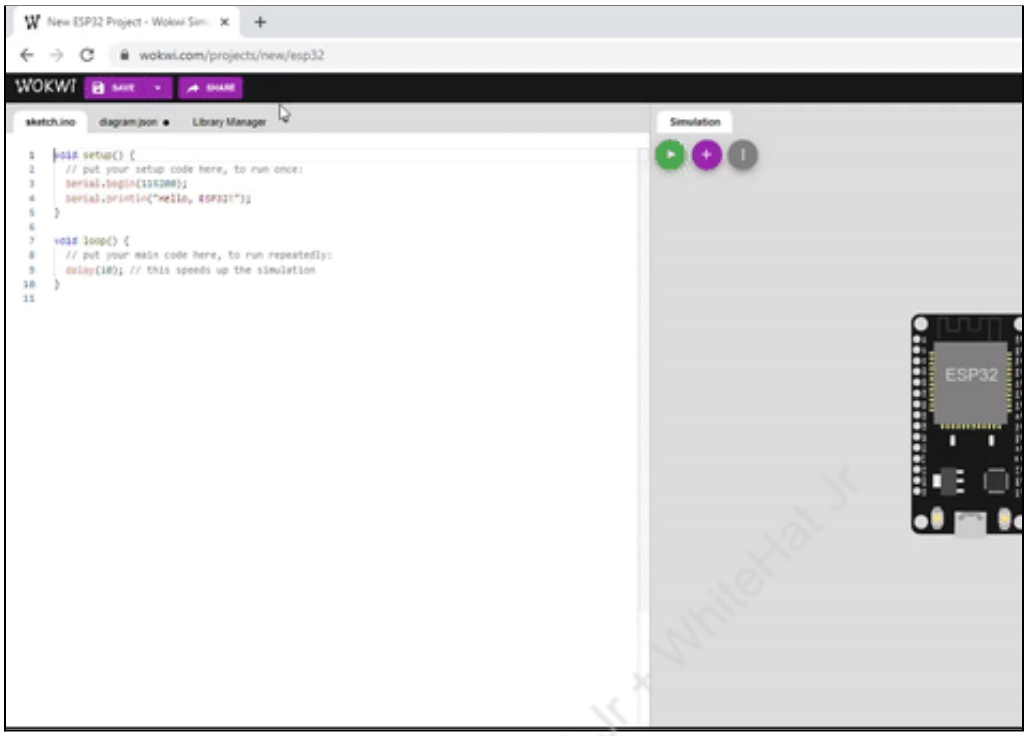
- Learned to use game states.
- Incorporated winning and losing conditions.
- Wrote code to show winning and losing messages..
- Learned to write code to reset the game using the joystick's push button.

## Which CONCEPTS/ CODING BLOCKS did we cover today?

- Concepts : conditional statements, functions. joysticks, LED Dot Matrix display..
- 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.




Click here to view the reference video.

2. Added welcome animation and text for the player:

   a. Define a function named **all_animate().** This function will switch on all the LEDs for 500 milliseconds and switch off the LEDs for 500. It will do this thrice to alert the user.

      - Define a new function named **all_animate().**
      - Use the **setPoint()** function to light up the LEDs.
      - To light up all the LEDs, use a for loop()

```
for (int x = 0; x < 8; x++){
   for (int y = 0; y < 8; y++){
     matrix.setPoint(y,x,true);
   }
 }
```

      - Now, let's keep these lit up for 500 milliseconds.

```
delay(500);
```

- Then, switch off the LEDs for 500 milliseconds.

```
matrix.clear();
delay(500);
```

- Write a **while()** loop around these lines of code to make all the LEDs light up thrice.

The code should look like this-

```
void all_animate(){
  int i=0;
  while(i<3){

    for (int x = 0; x < 8; x++){
      for (int y = 0; y < 8; y++){
        matrix.setPoint(y,x,true);
      }
    }
    delay(500);
    matrix.clear();
    delay(500);
    i++;
  }

}
```

b. Now, let's write the code to show some welcome text.

- Define **show_message()** method.
- **show_message()** method should have one parameter which should be a character array or a string. The message can be passed to the method through this parameter.

```
void show_message(char messageToShow[]){



}
```

- Call the **all_animate()** method in the **show_message()** method.

```
void show_message(char messageToShow[]){

  all_animate();

}
```

- Use the **setChar()** method to print each letter of the message passed through the method. Each letter should be shown for 500 milliseconds.

  Let's say, we want to print "W", we can write-

  ```
  matrix.setChar(6 , 'W');

  delay(500);
  ```

- But the message is stored as a string. Use a loop to access each index of the string.

  In c, every string ends with **'\0'**. **'\0'** is called the null character. This null character helps us to understand where the string ends and it is added automatically at the end of the string. Use this character to stop the loop.

- Initiate an integer named i to 0. This variable will help us hold the index of each character of the string.

```
int i=0;
```

- Add a **while** loop. This **while** loop should run until the last character of the string is accessed. So, the condition should be **messageToShow[i] != '\0'.**

```
void show_message(char messageToShow[]){

  all_animate();

  int i=0;

  while(messageToShow[i]!='\0'){

  }


}
```

- Inside the loop, use **setChar()** code to show each character of the string for 500 milliseconds.

  **matrix.setChar(6 , messageToShow[i]);**
  **delay(500);**

- Use the **clear()** method to clear up the display for 500 milliseconds.

  **matrix.clear();**
  **delay(500);**

- Lastly, increase the index variable i by one so that we can access the next character.

  **i++;**

The code should look like this-

```
void show_message(char messageToShow[]){

  all_animate();

  int i=0;

  while(messageToShow[i]!='\0'){
    matrix.setChar(6 , messageToShow[i]);
    delay(500);
    matrix.clear();
    delay(500);
    i++;
  }

}
```

- Call the **show_message()** method in the **setup()** method. Pass "WELCOME" as the argument.

```
void setup(){

  Serial.begin(9600);

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

  //  setting debounce time
  button.setDebounceTime(25);

  //  welcome text
  show_message("WELCOME");
}
```

3. Game states:

   Let's add game states to decrease complexity of our code when winning and losing conditions are added.

   a. There will be three gameStates in this game.

      - **Play**, which we will indicate with **0,**
      - **Win,** which we will indicate with **1,**
      - **Lose,** which we will indicate with **2.**

   b. Define a variable named gameState at the top. We will initiate it at 0.

```
int gameState=0;
```

   c. Let's look at the **loop()** method now and incorporate the gameState there. The code that we have written till now in the loop method should only run in the **play** state. So, let's put this section of code in an **if** condition.

```
void loop(){

  //  start noting down the time
  if(gameState==0){
    current_time = millis();
    check_direction();  //  check direction of joystick rotation
    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();  //  draw sprites
    }
    food_eat_check();  //  checking if snake has eaten food

  }
  //  for better working of simulator and controlling snake speed
  delay(10);

}
```

4. Game winning condition:

Let's implement the winning condition.When the player eats 15 food objects, the player should win the game

a. Write the code for the winning condition inside the **food_eat_check()** method.

```
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++;
    if (snake_length >= 15){



    }

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

b. If **snake_length** is equal or greater than 15, change the **gameState** to **1.**

```
//  changing snake_length
snake_length++;
if (snake_length >= 3){
    gameState=1;
}
```

c. Let's show a message saying "**YOU WON**" when **gameState** to **1**.

```
void loop(){

  //  start noting down the time
  if(gameState==0){
    current_time = millis();
    check_direction();  //  check direction of joystick rot
    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();  //  draw sprites
    }
    food_eat_check();  //  checking if snake has eaten food

  }

  if(gameState==1){
    show_message("YOU WON");
  }
```

5. Game losing condition:

   End the game when the snake's head touches its own body. To do that, define a new method called **snake_collision_check().**

   - Define the **snake_collision_check()** method.
   - Snake's head's index is stored at the 0th index of **snake_x** and **snake_y**. Compare the head's position with the rest of the body's position i.e. each dot in the snake's body except the head.

   If the position of the head and one of the dots from the snake's rest of the body has the same x,y value, then we can say that the snake has collided.

   Use a **for** loop which will loop through the 1st to last index of the snake's body. Compare each of these positions with the 0th index.

```
void snake_collision_check(){

  //  checking if head collides with any other body part
  for (int i = 1; i <= snake_length; i++){
    if (snake_x[0]  ==  snake_x[i]  &&  snake_y[0]  ==  snake_y[i]){
      gameState=2;
    }
  }

}
```

- When **gameState** is 2 or the player has lost, show the “game over” message.

```
void loop(){

  //  start noting down the time
  if(gameState==0){
    current_time = millis();
    check_direction();  //  check direction of joystick rotati
    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();  //  draw sprites
    }
    food_eat_check();  //  checking if snake has eaten food

  }

  if(gameState==1){
    show_message("YOU WON");
  }

  if(gameState==2){
    show_message("GAME OVER");
  }
```

- Call the **snake_collision_check()** method in the **play** state.

```
if(gameState==0){
  current_time = millis();
  check_direction();  //  check direction of joystick rotation
  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();  //  draw sprites
  }
  food_eat_check();  //  checking if snake has eaten food
  snake_collision_check();
}
```

6. Reset the game:

   Once the game is over, add a reset option for the game.

   a. Write another **if** condition and pass **gameState** as the condition. This condition will work when the **gameState** is not 0 i.e. it will work in both **Won** and **Lost** states. Write the code at the of **loop()** method so that it works after the "GAME OVER" message is displayed.

```
if (gameState){

}
```

   b. Write an infinite loop inside this **if** statement. This is because the game will wait here until the game restarts.

```
if (gameState){
  while(true){

  }
}
```

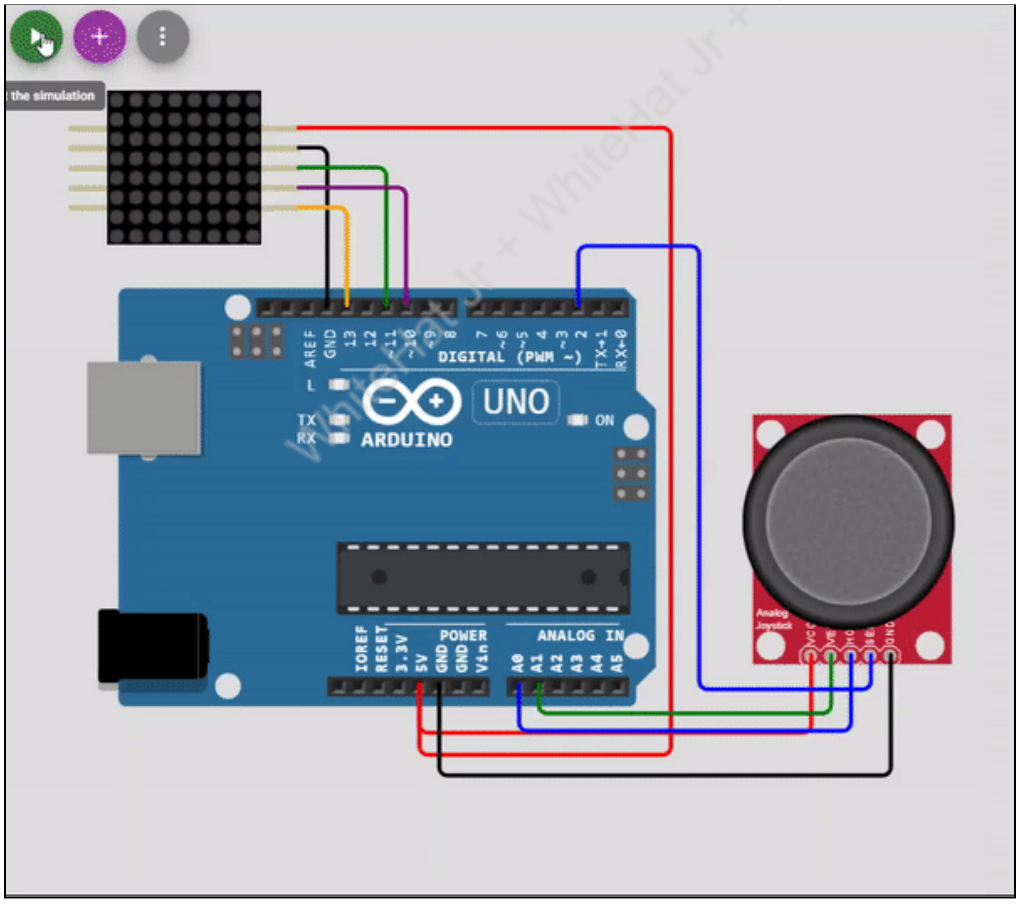
   c. Restart the game when the joystick's push button is pressed. Call the **button.loop()** function here.
   d. Now, if the button is pressed, reset the **gameState** to 0, so the game is in **play**

state again. Also, reset all the other variables.

```
if (button.isPressed()){

  //  resetting variables
  head_x = 0, head_y = 0, snake_length = 0, threshold = 800;
  direction = "" , prev_direction = "";
  for (int i = 0; i < 64; i++){
    snake_x[i] = 0;
    snake_y[i] = 0;
  }
  gameState = 0;
  break;
}
```

**Reference Output:**




Click here to view the full output video.

## What's NEXT?

In the **next class**, we will start to build a digital reading device.