

# CHAPTER - 1

## Introduction:

Games entertain by several means: gameplay, novelty, social interaction (if it is a multiplayer game), and so on. In a game such as chess, almost all the entertainment value is in the gameplay; few people think of it as a game about medieval warfare.

Online gaming is simply **the playing of a video game over the internet, usually with friends**. Online games can be played on any number of devices from dedicated video games consoles such as PlayStations, Xboxes, and Nintendo Switches, to PCs, laptops and mobile phones

The best way to learn any programming language is through hands-on projects. The Snake Game is a simple game you can make using the basics of JavaScript ,CSS and HTML.The basic goal is to navigate a snake and eat as many food as possible without touching the walls or the snake's body.In this project, i will go over step-by-step how to create this Snake Game using JavaScript , CSS and HTML.

### **What is meant by game development?**

Game Development is **the art of creating games and describes the design, development and release of a game**. It may involve concept generation, design, build, test and release. While you create a game, it is important to think about the game mechanics, rewards, player engagement and level design.

# CHAPTER -2

## Required Technologies:

### 1.HTML

HTML is the **language in which most websites are written**. HTML is used to create pages and make them functional. The code used to make them visually appealing is known as CSS and we shall focus on this in a later tutorial. For now, we will focus on **teaching you how to build rather than design**. HTML was first created by Tim Berners-Lee, Robert Cailliau, and others starting in **1989**. It stands for Hyper Text Markup Language. Hypertext means that the document contains **links that allow the reader to jump to other places** in the document or to another document altogether. The latest version is known as [HTML5](#). A **Markup Language** is a way that computers speak to each other to control how text is processed and presented.

### 2.CSS

**Cascading Style Sheets**, fondly referred to as **CSS**, is a simple design language intended to simplify the process of making web pages presentable. **CSS** is a **MUST** for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning CSS:

- **Create Stunning Web site** - CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, variations in display for different devices and screen sizes as well as a variety of other effects.

- **Become a web designer** - If you want to start a career as a professional web designer, HTML and CSS designing is a must skill.
- **Control web** - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.
- **Learn other languages** - Once you understand the basic of HTML and CSS then other related technologies like javascript, php, or angular are become easier to understand.

### **3.JAVA SCRIPT**

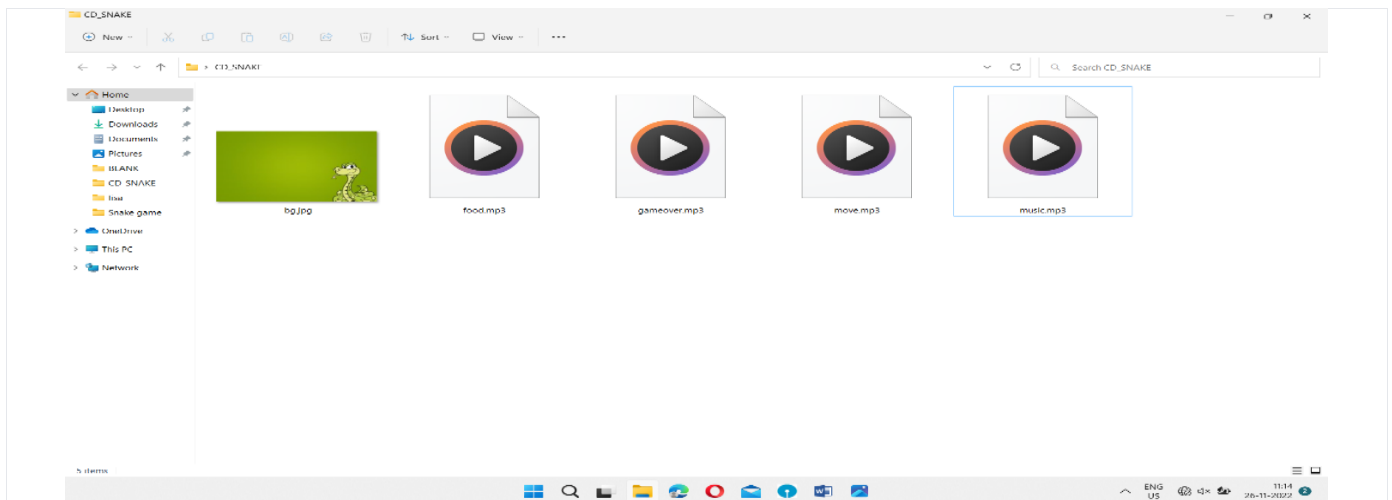
JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity

# CHAPTER - 3

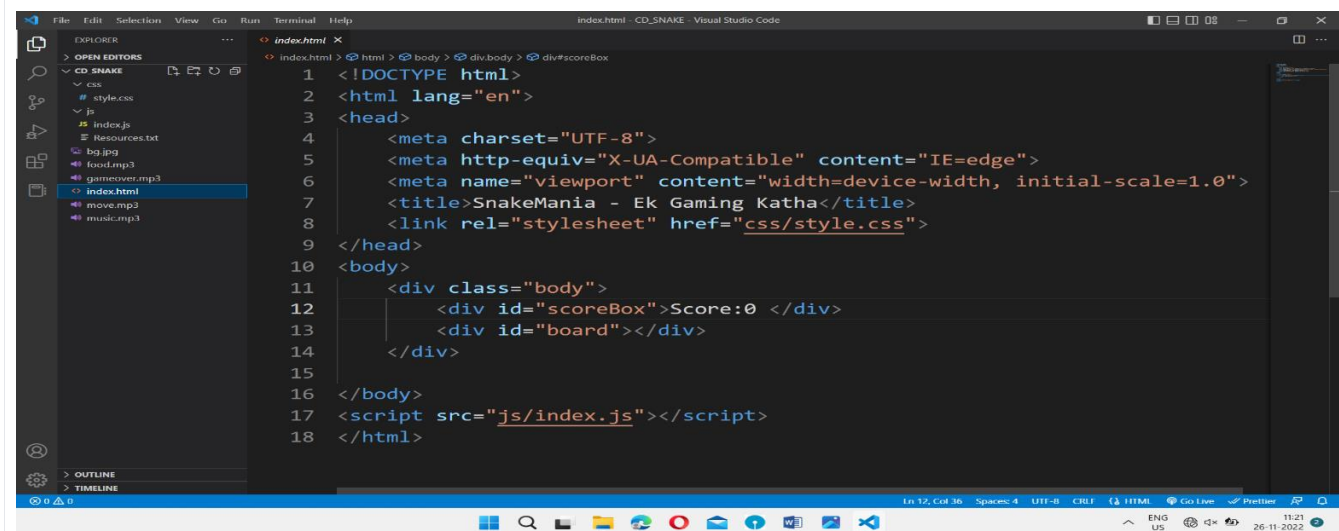
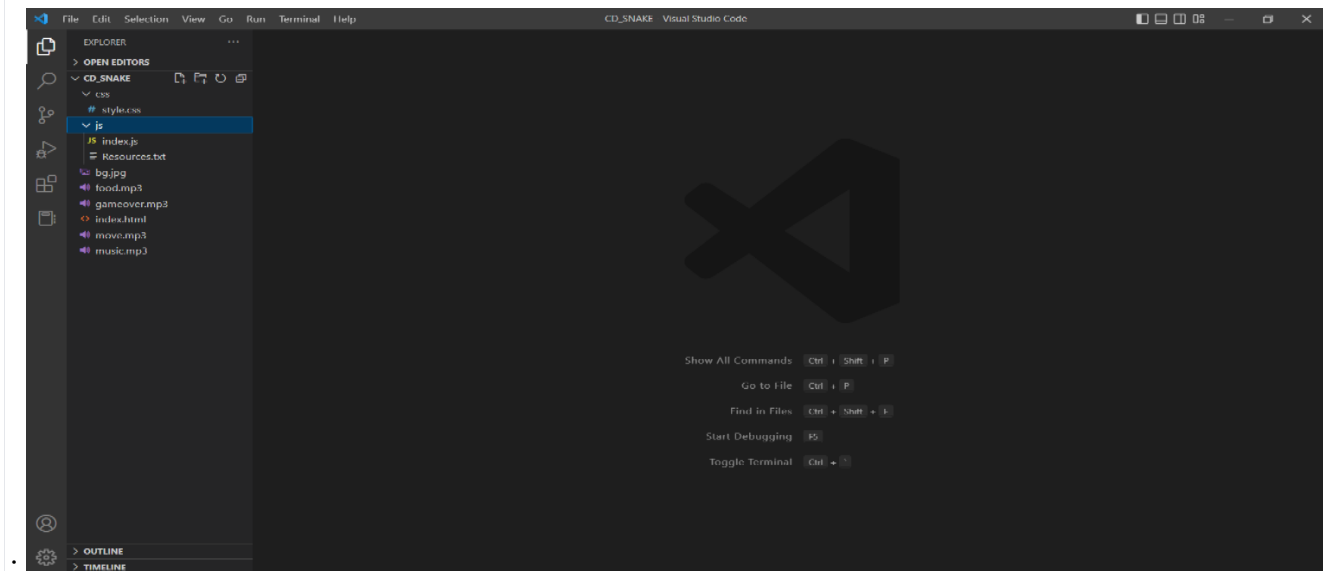
## Code Snippets

I will produce a snake game with the ease of HTML, CSS, and Javascript. Building this game will push you to boost your web development skills. It's very easy to make this game just by using the right tools at the right time. We will use sounds, animations, user input like upward, downward keys, etc. So let's begin with some starter files that I have downloaded in the folder from the internet such as :

1. bg.jpg (will use for the background of the game)
  2. food.mp3 (will play when the snake eats the food)
  3. gameover.mp3 (This sound will be played when the game is over)
  4. move.mp3 (will be played when the snake changes the direction)
- music.mp3 (We will use this file for the background music)



Let's open VS code in the folder then we will create one HTML file named index.html and four folders named CSS, js, img, and music. Now inside the folder named css, we will create a CSS file named style.css(for styling) and in the folder named js, we will create the JavaScript file named script.js. Afterward, we will move the image named bg.jpg into the folder named img and all the sounds into the folder named



Now before moving ahead, let me tell you that I will be using the live server extension to live reload the features of our game. I recommend you to use the same.

To know the process of installation and to run the Live Server look at the video given below:

In the index.html file, we will create a boilerplate. If you are using a text editor and if there is no emmet abbreviation, then you can type it there, and for your time convenience there is one shortcut in VS code, that is '!+Enter'. This will help you to create a boilerplate automatically. Then we will give the title as “SnakeMania - Ek Gaming Katha”. Now inside the <head> tag we will link a CSS file named style.css using <link> tag by giving the path of the file and after the <body> tag we will link a JavaScript file named script.js with <link> tag by giving the path of the file. **CREATING STRUCTURE OF GAME**

Inside the <body> tag of the index.html file,

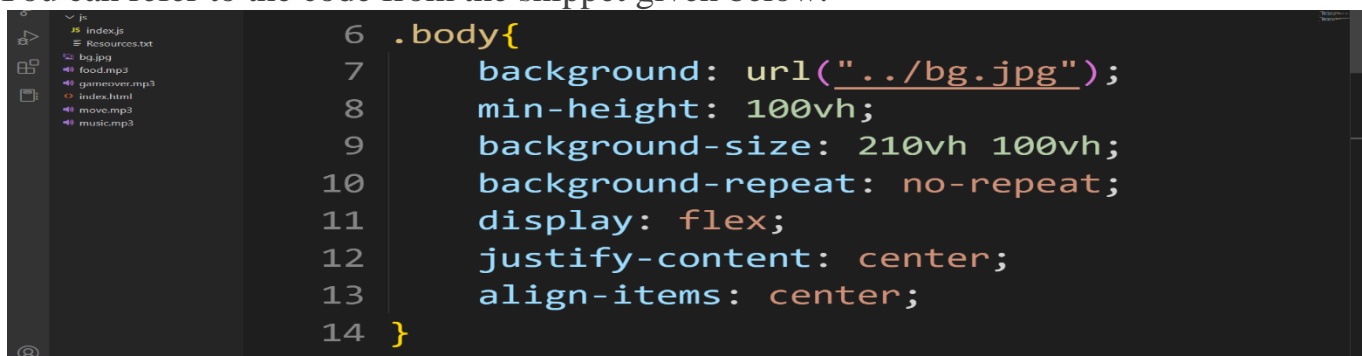
- We will create a <div> block with the class named body and inside it, we will create one child <div> block with the id named board. This div block will be our board (playing area of the game).
- Now we will create two more child <div> blocks with id named scorebox inside <div> block with the class named body and above the <div> block with id named board. These <div> blocks will display the current score of the game.
- We will write the text “Score: 0” inside the <div> block with id named scorebox .

## 1. Styling body

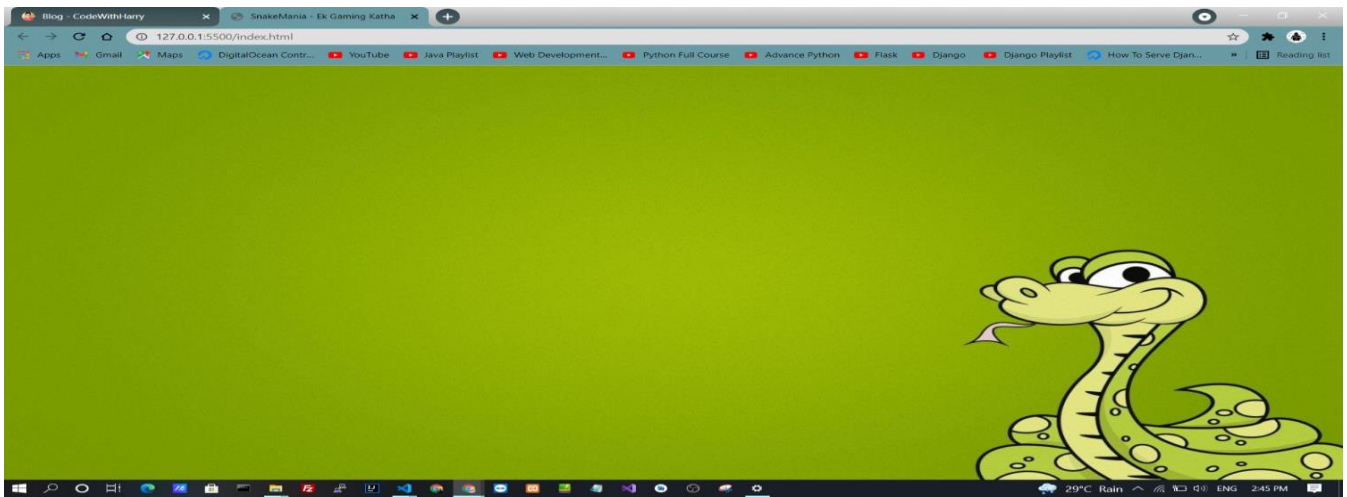
We will use an image named bg.jpg for the background. You can also use any image of your choice or if you are willing to use the same image, it is available in the source code at the end of the description. So let's start styling in the style.css file by adding a universal selector to reset CSS, then we will target the <div> block by a class named body and will add some CSS properties in it such as :

- background;(sets the background image).
- min-height: (defines the minimum height of an element).
- background-size: (specifies the size of the background images).
- background-repeat: no-repeat;(sets if/how a background image will be repeated).
- display: flex;(specifies the display behavior (the type of rendering box) of an element).
- justify-content: center;(aligns the flexible container's items when the items do not use all available space on the main-axis horizontally).
- align-items: center;(specifies the default alignment for items inside the flexible container).

You can refer to the code from the snippet given below.

A screenshot of a code editor with a dark theme. On the left, a file explorer shows a project structure with files like index.js, Resources.txt, bg.jpg, food.mp3, gameover.mp3, index.html, move.mp3, and music.mp3. The main editor area displays CSS code for the .body class, with line numbers 6 through 14 on the left margin. The code sets background, min-height, background-size, background-repeat, display, justify-content, and align-items.

```
6 .body{
7     background: url("../bg.jpg");
8     min-height: 100vh;
9     background-size: 210vh 100vh;
10    background-repeat: no-repeat;
11    display: flex;
12    justify-content: center;
13    align-items: center;
14 }
```

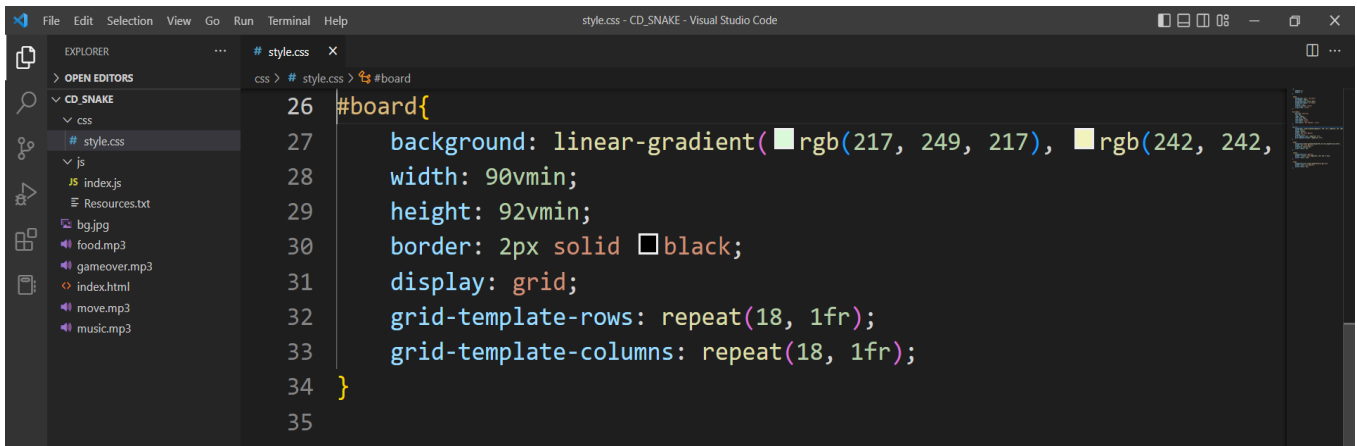


## 2. Styling board (playing area of the game)

Let's style the `<div>` block with the class named `board` which is the child of the `<div>` block with class named `body` in the `index.html` file, so we will target that `<div>` block in `style.css` file by class named `board` and in it we will add some CSS properties such as :

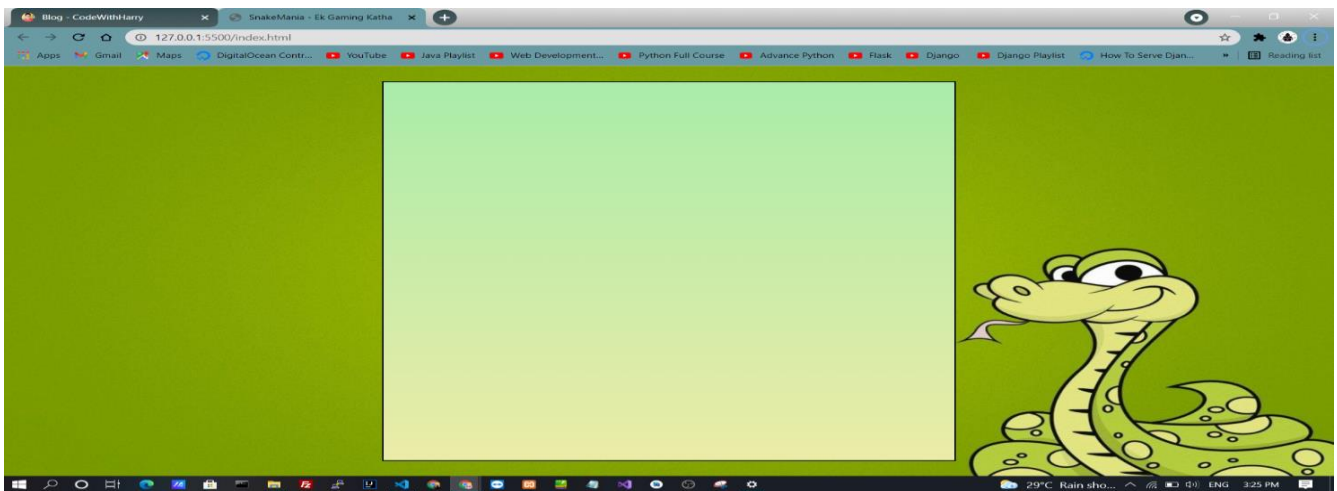
- `background: linear-gradient(rgb(170, 236, 170), rgb(236, 236, 167));`(sets the background using `linear-gradient()` function).
- `width: 90vmin;`(sets the width of an element).
- `height: 92vmin;`(sets the height of an element).
- `border: 2px solid black;`(sets the border on element).
- `display: grid;`(specifies the display behavior (the type of rendering box) of an element).
- `grid-template-rows: repeat(18, 1fr);`(specifies the number and the heights of the rows in a grid layout).
- `grid-template-columns: repeat(18, 1fr);`(specifies the number and the widths of columns in a grid layout).

You can refer to the code from the snippet given below.



The screenshot shows the Visual Studio Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'CD\_SNAKE' with files like 'style.css', 'index.js', 'Resources.txt', 'bg.jpg', 'food.mp3', 'gameover.mp3', 'index.html', 'move.mp3', and 'music.mp3'. The code editor shows the following CSS code for a board:

```
#board{
  background: linear-gradient(■rgb(217, 249, 217), ■rgb(242, 242, 242));
  width: 90vmin;
  height: 92vmin;
  border: 2px solid ■black;
  display: grid;
  grid-template-rows: repeat(18, 1fr);
  grid-template-columns: repeat(18, 1fr);
}
```

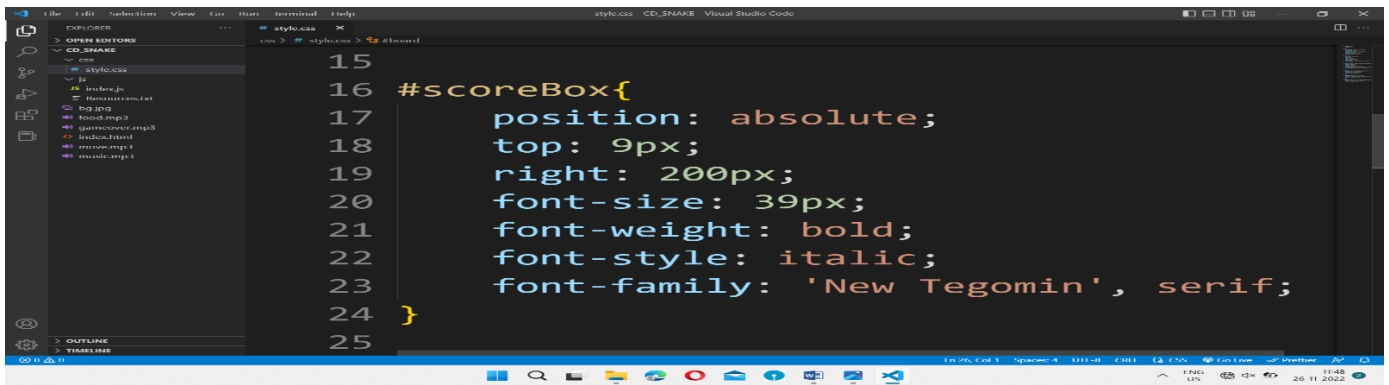


Now we will target the `<div>` block of a score box by id named `scorebox` and we will add some CSS properties such as :

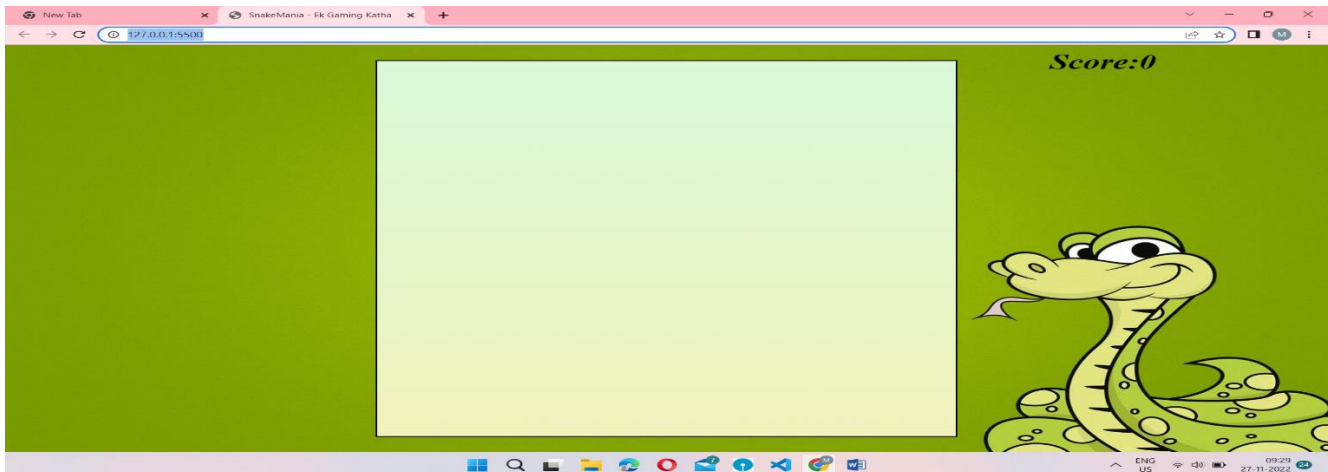
- `position: absolute;`(specifies the type of positioning method used for an element).
- `top:` (affects the vertical position of a positioned element).
- `right:` (affects the horizontal position of a positioned element).
- `font-size:` (sets the size of a font).
- `font-weight: bold;`(sets how thick or thin characters in text should be displayed).
- `font-family: 'New Tegomin', serif;`(specifies the font for an element).

You can refer to the code from the snippet given below.





```
15
16 #scoreBox{
17     position: absolute;
18     top: 9px;
19     right: 200px;
20     font-size: 39px;
21     font-weight: bold;
22     font-style: italic;
23     font-family: 'New Tegomin', serif;
24 }
25
```



#### 4. Styling of head, snake and food

We will style the head which will be used in our JavaScript file to display the head of the snake so we will style the head which will add some CSS properties in head such as

```
.head{
    background:linear-gradient(rgb(240,124,124),rgb(228,228,129));
    border: 2px solid purple;
    transform: scale(1.02);
    border-radius: 9px;
}
```

Let's style the snake which will be used in our JavaScript in our JavaScript file to display the snake so we add some CSS properties in the snake such as:

```
.snake{
    background-color: purple;
    border: .25vmin solid rgba(236, 230, 230, 0.333);
    border-radius: 15px;
}
```

And the last one is food which will be used in our JavaScript file to display the food so we will add some CSS properties in food such as:

```
.food{
  background-color:linear-gradient(red,purple);
  border: 2vmin solid black;
  border-radius: 8px;
}
```

## **5. Game Constants & Variables**

We will create all constants and variables in a JavaScript file named java.script that will be used in our game as shown below.

Code java.script as described/written in the blog:

```
let inputDir = {x: 0, y: 0};
const foodSound = new Audio('food.mp3');
const gameOverSound = new Audio('gameover.mp3');
const moveSound = new Audio('move.mp3');
const musicSound = new Audio('music.mp3');
let speed = 5;
let score = 0;
let lastPaintTime = 0;
let snakeArr = [
  {x:13, y:15}];
Food = {x:6, y:7};
```

## **Game Functions**

### **1. Main Function**

We will create the main function for rendering and here instead of the setInterval() method we will use the window.requestAnimationFrame() function for the reason [click here](#). The ctime is our current time and we will create an if statement for time such that it will render the game when the condition is false and after that, this function will call gameEngine() function which will be created later. You can refer to the code from the snippet given below.

```
function main(ctime){
  window.requestAnimationFrame(main);
  //console.log(ctime)
```

```

if((ctime - lastPaintTime)/1000 < 1/speed){
    return;
}
lastPaintTime = ctime;
gameEngine();
}

```

## 2. Function for collision

We will create the function named `isCollide` and give parameter `snake`. First, we will write the logic for if a snake bumps into itself, for that we will create a loop in `snakeArr` and then create the if statement in it with the condition for collision into itself and if the condition is true it will return true. Now we will write the logic for if the snake bumps into the wall, for that we will create one more if statement with the condition for collision into the wall and if the condition is true it will return true. You can refer to the code from the snippet given below

```

function isCollide(snake){
    // If you bump into yourself
    for(let i = 1; i < snakeArr.length; i++){
        if(snake[i].x === snake[0].x && snake[i].y === snake[0].y){
            return true;
        }
    }
    // If you bump into wall
    if(snake[0].x >= 18 || snake[0].x <= 0 || snake[0].y >= 18 || snake[0].y <= 0){
        return true;
    }
    return false;
}

```

## 3. gameEngine() Function

We will create the function named `gameEngine()` as shown below.

```

function gameEngine(){

```

Inside this function, we will write the logic for,

- updating the snake array and food.
- if the snake has eaten the food, increment the score and regenerate the food.

- moving the snake.
- displaying the snake.
- displaying the food.

All these points will see one by one as follows :

#### •Logic for updating the snake array and food

This function will update the snake array and food. At first, we will create an if statement with condition that if the function isCollide() with parameter snakeArr returns true means the snake collides with itself or with the wall, then it will play gameOverSound and the background music that is musicSound will be paused and will set the inputDir to its initial value then we will create an alert with text message “Game Over. Press any key to play again!”. After that we will reset the snakearr then it will play musicSound again and it will set the score equal to zero. You can refer to the code from the snippet given below.

```
// Part 1: Updating the snake array & Food
if(isCollide(snakeArr)){
    gameOverSound.play();
    musicSound.pause();
    inputDir = {x: 0, y: 0};
    alert("Game Over. Press any key to play again!");
    snakeArr = [{x: 13, y: 15}];
    musicSound.play();
    score = 0;
}
```

#### • Logic for if the snake has eaten the food, increment the score and regenerate the food

Now we will create an if statement with the condition that if the snake has eaten the food then we will play the foodSound and increment the score by 1. We will store the highscore in localStorage and Whenever the snake eats the food we will add one head to it using the unshift() function. After the snake eats the food we will regenerate the food by generating random values between a and b which are the variables created with the values such as a=2 and b=16. You can refer to the code from the snippet given below.

```
// If you have eaten the food, increment the score and regenerate the food
if(snakeArr[0].y === food.y && snakeArr[0].x === food.x){
    foodSound.play();
    score +=1;
}
```

```

scoreBox.innerHTML="Score:" + score;
snakeArr.unshift({x: snakeArr[0].x + inputDir.x, y: snakeArr[0].y + inputDir.y});
let a = 2;
let b = 16;
food = {x: Math.round(a +(b - a)* Math.random()), y: Math.round(a + (b-a)*
Math.random())}
}

```

#### • Logic for moving the snake

To move the snake we will iterate the whole snake body for that we will create a decrementing for loop in snakeArr and will write some logic to shift the element one by one from the end. You can refer to the code from the snippet given below.

```

// Moving the Snake
for(let i = snakeArr.length-2; i >= 0;i--){
    snakeArr[i+1] = {...snakeArr[i]};
}
snakeArr[0].x += inputDir.x;
snakeArr[0].y += inputDir.y;

```

#### • Logic for displaying the snake

For displaying the snake first we will clear the innerHTML of the board so when the game loads the snake will render only once. Then we will create a forEach loop in snakeArr and inside it, we will create a new element div inside a variable named snakeElement, then giving style to it such as gridRowStart is e.y and gridCloumnStart is e.x. Now we will create an if-else statement such as if the index is equal to 0 then the class named head will be added to snakeElement else the class named snake will be added to snakeElement. After that we will append the snakeElement as a child inside the board. You can refer to the code from the snippet given below.

```

// Display the snake
board.innerHTML = "";
snakeArr.forEach((e, index)=>{
    snakeElement = document.createElement('div');

```

```

snakeElement.style.gridRowStart = e.y;
snakeElement.style.gridColumnStart = e.x;

if(index === 0){
    snakeElement.classList.add('head');
}
else{
    snakeElement.classList.add('snake')
}
board . appendChild(snakeElement);

});

```

#### • Logic for displaying the food

Let's create an element div in a variable named foodElement and will give the style such as gridRowStart is food.y and gridColumnStart is food.x then inside foodElement we will add the class named food. After that, we will append the foodElement as a child inside the board. You can refer to the code from the snippet given below.

```

// Display the food

foodElement = document.createElement('div');
foodElement.style.gridRowStart = food.y;
foodElement.style.gridColumnStart = food.x;
foodElement.classList.add('food')
board.appendChild(foodElement);

```

**Whole gameEngine( ) code in javascript as described below:**

```

function gameEngine(){

    // Part 1: Updating the snake array & Food

    if(isCollide(snakeArr)){

        gameOverSound.play();
        musicSound.pause();
    }
}

```

```

inputDir = {x: 0, y: 0};
alert("Game Over. Press any key to play again!");
snakeArr = [{x: 13, y: 15}];
musicSound.play();
score = 0;
}
// If you have eaten the food, increment the score and regenerate the food
if(snakeArr[0].y === food.y && snakeArr[0].x === food.x){
    foodSound.play();
    score +=1;
    scoreBox.innerHTML="Score:" + score;
    snakeArr.unshift({x: snakeArr[0].x + inputDir.x, y: snakeArr[0].y + inputDir.y});
    let a = 2;
    let b = 16;
    food ={x: Math.round(a +(b - a)* Math.random()), y: Math.round(a + (b-a)*
Math.random())}
}

```

### **// Moving the Snake**

```

for(let i = snakeArr.length-2; i >= 0;i--){
    snakeArr[i+1] = {...snakeArr[i]};
}
snakeArr[0].x += inputDir.x;
snakeArr[0].y += inputDir.y;

```

### **// Part 2: Display the snake and Food**

#### **// Display the snake**

```

board.innerHTML = "";
snakeArr.forEach((e, index)=>{
    snakeElement = document.createElement('div');

```

```

snakeElement.style.gridRowStart = e.y;
snakeElement.style.gridColumnStart = e.x;
if(index === 0){
    snakeElement.classList.add('head');
}
else{
    snakeElement.classList.add('snake')
}
board . appendChild(snakeElement);
});

```

### **// Display the food**

```

foodElement = document.createElement('div');
foodElement.style.gridRowStart = food.y;
foodElement.style.gridColumnStart = food.x;
foodElement.classList.add('food')
board.appendChild(foodElement);
}

```

## **• Main logic of game**

### **1. Background music**

When the game starts we will play musicSound, which is the background music of our game as shown below.

```
musicSound.play();
```

### **2. Creating controls of the game**

At first, we will put windowAnimationFrame() with parameter main. Then we will create an addEventListener as keydown then inside it we will set the initial value of snake in inputDir and will create the switch case with giving parameter e.key and inside it will set the cases for keys such as when,

- ArrowUp key is clicked the snake will move in an upper direction
- ArrowDown key is clicked snake will move towards to downwards direction
- ArrowLeft key is clicked snake will move in the left direction



- ArrowRight key is clicked and the snake will move in the right direction.

Remember to add the break keyword after every case. You can refer to the code from the snippet given below.

```
window.requestAnimationFrame(main);
window.addEventListener('keydown', e =>{
  inputDir = { x: 0 , y:1 } // Start the game
  moveSound.play();
  switch(e.key){
    case "ArrowUp":
      console.log("ArrowUp");
      inputDir.x = 0;
      inputDir.y = -1;
      break;
    case "ArrowDown":
      console.log("ArrowDown");
      inputDir.x = 0;
      inputDir.y = 1;
      break;
    case "ArrowLeft":
      console.log("ArrowLeft");
      inputDir.x = -1;
      inputDir.y = 0;
      break;
    case "ArrowRight":
      console.log("ArrowRight");
      inputDir.x = 1;
      inputDir.y = 0;
      break;
    default:
```

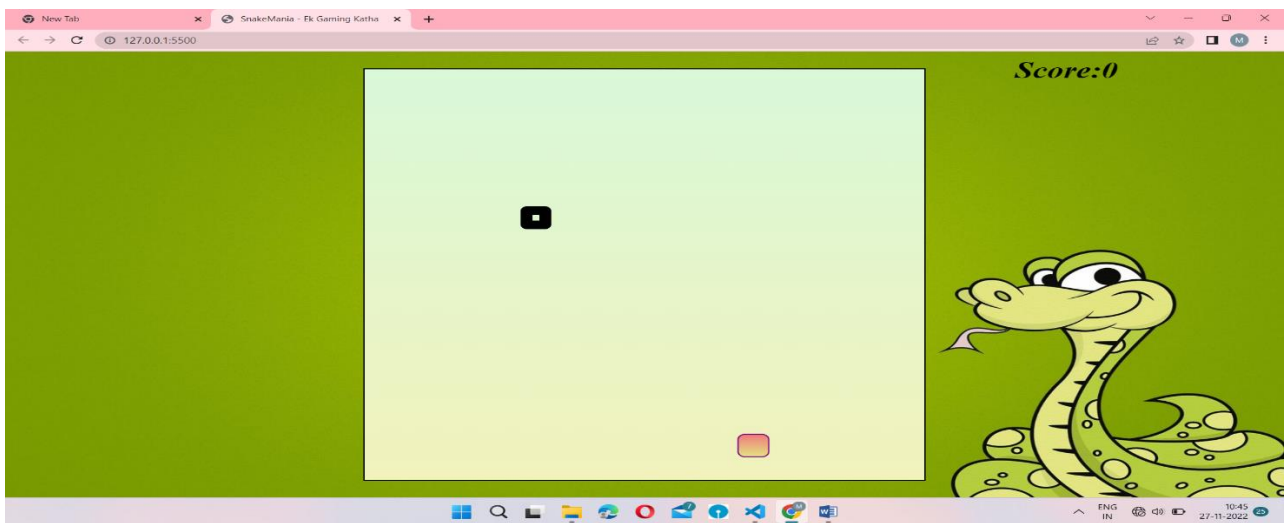
```
        break;
    }
});
```

### **Main Logic code java.script as described/written in the blog:**

```
//Main logic starts here
musicSound.play();
window.requestAnimationFrame(main);
window.addEventListener('keydown', e =>{
    inputDir = {x: 0 , y:1} // Start the game
    moveSound.play();
    switch(e.key){
        case "ArrowUp":
            console.log("ArrowUp");
            inputDir.x = 0;
            inputDir.y = -1;
            break;
        case "ArrowDown":
            console.log("ArrowDown");
            inputDir.x = 0;
            inputDir.y = 1;
            break;
        case "ArrowLeft":
            console.log("ArrowLeft");
            inputDir.x = -1;
            inputDir.y = 0;
            break;
        case "ArrowRight":
            console.log("ArrowRight");
            inputDir.x = 1;
```

```
    inputDir.y = 0;  
    break;  
default:  
    break;  
}  
});
```

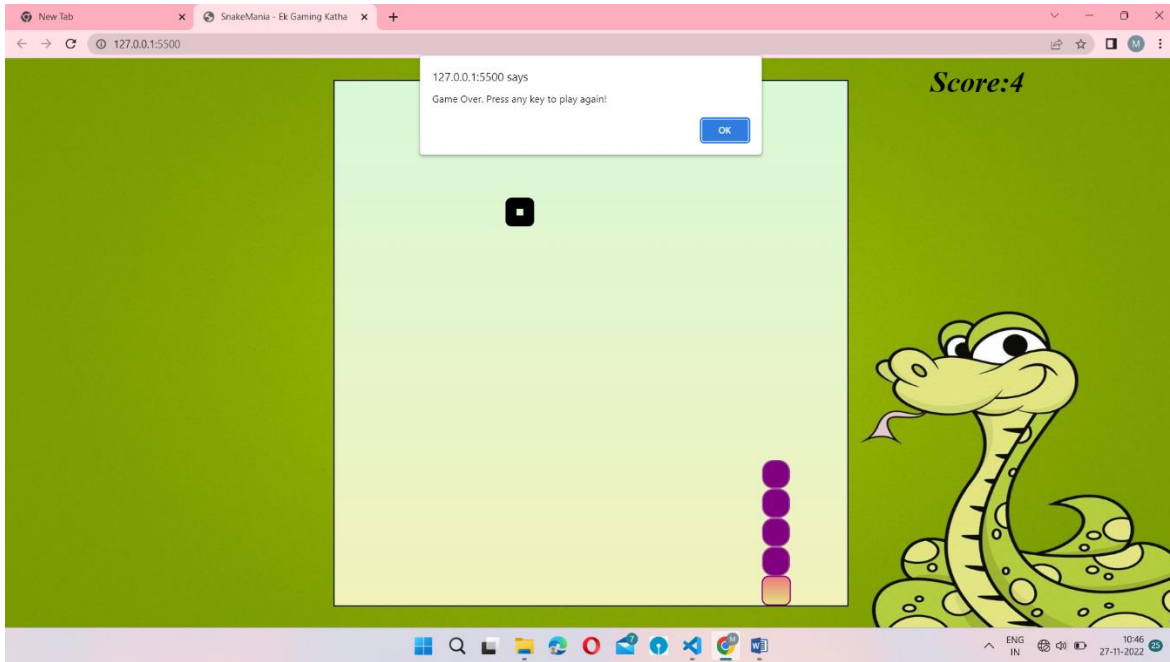
So our game is ready now it's time to show the gameplay of this game and I will make my Score. It will be a pleasure for everyone.



Here the score is increase and if you touch the boundary and touch the body of snake itself the game will over. To restart the game again we have to press any key of the keyboard and enjoy the game.

# CHAPTER - 4

## *OUTPUT*



# **CHAPTER – 5**

## ***REFERENCES***

- 1.) <https://www.codewithharry.com/videos/snake-game-in-javascript/>
- 2.) <https://www.codereview.stackexchange.com/questions/252641/a-snake-game-in-html-css-javascript>
- 3.) <https://www.studytonight.com/post/snake-game-in-html-and-javascript>
- 4.) <https://www.webtips.dev/how-i-made-a-snake-game-out-of-checkboxes>
- 5.) <https://www.freecodecamp.org/news/how-to-build-a-snake-game-in-javascript/>