SNAKE GAME PROJECT AND FUNCTIONALITY

The classic Snake game is a fantastic project for practicing web development skills, utilizing a combination of Java for logic, HTML for structure, and CSS for styling. Let's delve into the functionalities and components of a basic Snake game implemented using these technologies:

1. \*\*HTML Structure\*\*:

- The HTML file lays out the basic structure of the game. It includes the canvas element where the game will be displayed.

- It also contains elements for displaying the score and any other necessary information.

2. \*\*CSS Styling\*\*:

- CSS is used to style the appearance of the game elements, including the canvas, score display, and any buttons or controls.

- Styling can include colors, fonts, borders, and layout adjustments to make the game visually appealing and user-friendly.

3. \*\*JavaScript (Java) Logic\*\*:

- JavaScript (not Java in this context) is used for the game logic. JavaScript handles user input, updates the game state, and renders the game on the canvas.

- The logic includes:

- Handling keyboard input to control the snake's movement.

- Managing the snake's position and length.

- Generating food for the snake to eat.

- Detecting collisions with the walls, itself, or the food.

- Updating the score and displaying it to the user.

- Rendering the game graphics on the canvas, including the snake, food, and background.

4. \*\*Game Initialization\*\*:

- When the game starts, the snake is initialized with a starting position and length.

- The food is randomly placed on the canvas.

- The game loop begins, where the game continuously updates and renders until the game ends.

5. \*\*Game Loop\*\*:

- The game loop is a fundamental concept in game development. It consists of repeatedly updating the game state and rendering it to the screen.

- In the Snake game, the loop performs the following steps:

- Check for user input to change the snake's direction.

- Move the snake one step in its current direction.

- Check for collisions with the walls, itself, or the food.

- If the snake collides with food, increase its length and generate new food.

- Update the score based on the snake's length.

- Render the updated game state to the canvas.

6. \*\*Collision Detection\*\*:

- Collision detection is crucial for determining when the game should end or when certain actions should occur.

- In Snake, collision detection includes:

- Checking if the snake collides with the walls of the canvas.

- Checking if the snake collides with itself.

- Checking if the snake's head overlaps with the food.

7. \*\*Game Over\*\*:

- The game ends when the snake collides with the walls or itself.

- When the game ends, the user is informed of their score and given the option to restart the game.

Overall, building the Snake game using Java, HTML, and CSS is a great way to practice web development skills while also learning fundamental game development concepts. It provides a hands-on experience in implementing user interaction, game logic, and visual rendering using these technologies.

## Features

- Classic Snake gameplay.

- Built with React.js and HTML canvas.

- No third-party libraries used.

- TypeScript for type safety.

- Responsive design.

- Score tracking.

- Saves HighScore

- Game over screen with the option to restart.

- Keyboard controls for navigation.

\*\*Observations and Additional Functionalities for Snake Game Project\*\*

\*\*Observations:\*\*

1. \*\*Basic Gameplay Elements\*\*: The Snake game project effectively implements the core gameplay elements, including controlling the snake's movement, collecting food, and growing in length.

2. \*\*User Interface Design\*\*: The use of HTML and CSS for structuring and styling the game interface provides a clean and visually appealing user experience. However, there's potential to enhance the UI further with more engaging graphics and animations.

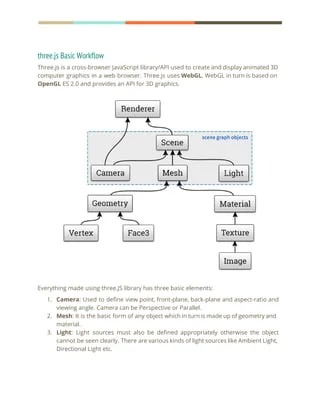
3. \*\*Game Logic Implementation\*\*: JavaScript handles the game logic efficiently, managing collision detection, score updates, and rendering on the canvas. The logic is well-structured and easy to understand, contributing to the overall functionality of the game.

4. \*\*Cross-browser Compatibility\*\*: It's essential to ensure cross-browser compatibility to reach a broader audience. Testing the game across different web browsers and devices can help identify and resolve any compatibility issues.

\*\*Possible Additional Functionalities:\*\*

1. \*\*Levels and Difficulty Settings\*\*: Implement multiple levels with increasing difficulty levels. This can include faster snake movement, additional obstacles, or changes in the game environment to provide a more challenging experience for players.

2. \*\*Customization Options\*\*: Allow players to customize their gaming experience by choosing different snake skins, backgrounds, or music tracks. Providing customization options adds personalization and enhances player engagement.



SOURCE CODE

HTML

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<meta name="viewport",

content="width=device-width, initial-scale=1.0">

<title>Snake Game with GFG</title>

<link rel="stylesheet" href="style.css">

<script src="script.js"></script>

</head>

<body>

<h1>Snake Game with

<div class="geeks">Geeks For Geeks</div>

</h1>

<canvas id="board"></canvas>

</body>

</html>

CSS

/\* Write CSS Here \*/

body {

text-align: center;

}

.geeks {

font-size: 40px;

font-weight: bold;

color: green;

}

JAVASCRIPT WITH REACT.JS

import React, { useState, useEffect } from "react";

const blockSize = 25;

const totalRow = 17;

const totalCol = 17;

const SnakeGame = () => {

const [snakeX, setSnakeX] = useState(blockSize \* 5);

const [snakeY, setSnakeY] = useState(blockSize \* 5);

const [speedX, setSpeedX] = useState(0);

const [speedY, setSpeedY] = useState(0);

const [snakeBody, setSnakeBody] = useState([]);

const [foodX, setFoodX] = useState(0);

const [foodY, setFoodY] = useState(0);

const [gameOver, setGameOver] = useState(false);

useEffect(() => {

const board = document.getElementById("board");

board.height = totalRow \* blockSize;

board.width = totalCol \* blockSize;

const context = board.getContext("2d");

placeFood();

const handleKeyDown = (e) => {

if (e.code === "ArrowUp" && speedY !== 1) {

setSpeedX(0);

setSpeedY(-1);

} else if (e.code === "ArrowDown" && speedY !== -1) {

setSpeedX(0);

setSpeedY(1);

} else if (e.code === "ArrowLeft" && speedX !== 1) {

setSpeedX(-1);

setSpeedY(0);

} else if (e.code === "ArrowRight" && speedX !== -1) {

setSpeedX(1);

setSpeedY(0);

}

};

document.addEventListener("keydown", handleKeyDown);

const update = () => {

if (gameOver) {

return;

}

context.fillStyle = "green";

context.fillRect(0, 0, board.width, board.height);

context.fillStyle = "yellow";

context.fillRect(foodX, foodY, blockSize, blockSize);

if (snakeX === foodX && snakeY === foodY) {

setSnakeBody([...snakeBody, [foodX, foodY]]);

placeFood();

}

const newSnakeBody = [...snakeBody];

if (newSnakeBody.length) {

newSnakeBody[0] = [snakeX, snakeY];

}

context.fillStyle = "white";

const newSnakeX = snakeX + speedX \* blockSize;

const newSnakeY = snakeY + speedY \* blockSize;

setSnakeX(newSnakeX);

setSnakeY(newSnakeY);

context.fillRect(newSnakeX, newSnakeY, blockSize, blockSize);

for (let i = 0; i < newSnakeBody.length; i++) {

context.fillRect(

newSnakeBody[i][0],

newSnakeBody[i][1],

blockSize,

blockSize

);

}

if (

newSnakeX < 0 ||

newSnakeX > totalCol \* blockSize ||

newSnakeY < 0 ||

newSnakeY > totalRow \* blockSize

) {

setGameOver(true);

alert("Game Over");

}

for (let i = 0; i < newSnakeBody.length; i++) {

if (newSnakeX === newSnakeBody[i][0] && newSnakeY === newSnakeBody[i][1]) {

setGameOver(true);

alert("Game Over");

}

}

};

const gameInterval = setInterval(update, 1000 / 10);

return () => {

clearInterval(gameInterval);

document.removeEventListener("keydown", handleKeyDown);

};

}, [snakeX, snakeY, speedX, speedY, snakeBody, foodX, foodY, gameOver]);

const placeFood = () => {

const newFoodX = Math.floor(Math.random() \* totalCol) \* blockSize;

const newFoodY = Math.floor(Math.random() \* totalRow) \* blockSize;

setFoodX(newFoodX);

setFoodY(newFoodY);

};

return (

<canvas id="board"></canvas>

);

};

export default SnakeGame;