Certainly! Here's an explanation of each line of the JavaScript code for the Snake game:

### **Code Breakdown:**

javascript

Copy code

const canvas = document.getElementById("gameCanvas");

const ctx = canvas.getContext("2d");

1. **const canvas**: This selects the <canvas> element from the HTML by its ID (gameCanvas) and stores it in the variable canvas.
2. **const ctx = canvas.getContext("2d")**: This initializes the 2D drawing context on the canvas. It allows us to draw shapes, text, images, and more on the canvas.

javascript

Copy code

const box = 20;

const canvasSize = 400;

1. **const box = 20**: This defines the size of each segment of the snake (and the food) as a 20x20 pixel square.
2. **const canvasSize = 400**: This sets the total size of the canvas to 400x400 pixels. The canvas will contain 20 rows and 20 columns of squares (since 400 / 20 = 20).

javascript

Copy code

const snake = [{ x: 200, y: 200 }];

1. **const snake = [{ x: 200, y: 200 }]**: This defines the initial position of the snake as an array of objects, where each object represents a segment of the snake. Here, the snake starts with just one segment, located at coordinates (200, 200).

javascript

Copy code

let food = { x: Math.floor(Math.random() \* (canvasSize / box)) \* box, y: Math.floor(Math.random() \* (canvasSize / box)) \* box };

1. **let food**: This defines the food's location on the canvas. The food is placed at a random location within the grid. The Math.floor(Math.random() \* (canvasSize / box)) \* box calculates a random position, making sure it fits inside the grid.

javascript

Copy code

let direction = { x: 0, y: 0 };

1. **let direction**: This object tracks the current movement direction of the snake. Initially, the snake is stationary, so both the x and y directions are set to 0.

javascript

Copy code

let score = 0;

1. **let score = 0**: This variable keeps track of the player's score, which increases whenever the snake eats food.

javascript

Copy code

let gameInterval;

1. **let gameInterval**: This will store the interval ID of the game loop, allowing us to stop the game loop later using clearInterval() when the game ends.

javascript

Copy code

document.addEventListener("keydown", changeDirection);

1. **document.addEventListener("keydown", changeDirection)**: This adds an event listener to the whole document, which listens for when a key is pressed on the keyboard. When a key is pressed, it calls the changeDirection function.

### **Function: changeDirection**

javascript

Copy code

function changeDirection(event) {

const keyPressed = event.key;

1. **const keyPressed = event.key**: This gets the key that was pressed and stores it in the keyPressed variable.

javascript

Copy code

if (keyPressed === "ArrowUp" && direction.y === 0) {

direction = { x: 0, y: -box };

} else if (keyPressed === "ArrowDown" && direction.y === 0) {

direction = { x: 0, y: box };

} else if (keyPressed === "ArrowLeft" && direction.x === 0) {

direction = { x: -box, y: 0 };

} else if (keyPressed === "ArrowRight" && direction.x === 0) {

direction = { x: box, y: 0 };

}

}

1. **Change Direction Logic**: Based on which arrow key is pressed, the direction of the snake is updated. The conditions (e.g., direction.y === 0) prevent the snake from moving in the opposite direction. This ensures, for example, that if the snake is moving up, it can't immediately move down.

* **ArrowUp**: If the up arrow is pressed, the snake moves upward by decreasing its y-coordinate (y: -box).
* **ArrowDown**: Moves downward by increasing the y-coordinate (y: box).
* **ArrowLeft**: Moves left by decreasing the x-coordinate (x: -box).
* **ArrowRight**: Moves right by increasing the x-coordinate (x: box).

### **Function: gameLoop**

javascript

Copy code

function gameLoop() {

if (isGameOver()) {

clearInterval(gameInterval);

setTimeout(() => {

alert(`Game Over! Your score: ${score}`);

location.reload();

}, 100);

return;

} else {

updateSnake();

draw();

}

}

1. **if (isGameOver())**: This checks if the game is over using the isGameOver function. If the game is over:
   * **clearInterval(gameInterval)**: Stops the game loop by clearing the interval.
   * **setTimeout()**: Delays the alert to allow any final drawing updates before the alert is shown.
   * **alert()**: Displays the "Game Over" message with the final score.
   * **location.reload()**: Reloads the page, resetting the game.
2. **return**: Ends the function to prevent further execution once the game is over.
3. **updateSnake()**: If the game isn’t over, this function updates the position of the snake.
4. **draw()**: Draws the updated game state (snake, food, and score) on the canvas.

### **Function: updateSnake**

javascript

Copy code

function updateSnake() {

const newHead = { x: snake[0].x + direction.x, y: snake[0].y + direction.y };

1. **const newHead**: This calculates the new head position of the snake based on the current direction.

javascript

Copy code

snake.unshift(newHead);

1. **snake.unshift(newHead)**: This adds the new head to the beginning of the snake array (moves the snake forward).

javascript

Copy code

if (newHead.x === food.x && newHead.y === food.y) {

score += 1;

food = { x: Math.floor(Math.random() \* (canvasSize / box)) \* box, y: Math.floor(Math.random() \* (canvasSize / box)) \* box };

} else {

snake.pop();

}

}

1. **Check if Snake Eats Food**: If the new head's position matches the food's position (newHead.x === food.x && newHead.y === food.y), the score increases by 1, and new food is placed randomly on the canvas.
2. **snake.pop()**: If the snake doesn’t eat food, it removes the last segment (tail) to maintain the snake's size.

### **Function: isGameOver**

javascript

Copy code

function isGameOver() {

const head = snake[0];

1. **const head = snake[0]**: This stores the current position of the snake's head (the first element in the snake array).

javascript

Copy code

if (head.x < 0 || head.x >= canvasSize || head.y < 0 || head.y >= canvasSize) {

return true;

}

1. **Check Wall Collision**: If the head moves outside the canvas boundaries, the game is over.

javascript

Copy code

for (let i = 1; i < snake.length; i++) {

if (head.x === snake[i].x && head.y === snake[i].y) {

return true;

}

}

return false;

}

1. **Check Self-Collision**: If the head's position overlaps with any part of the snake's body (i.e., the head collides with the body), the game is over.

### **Function: draw**

javascript

Copy code

function draw() {

ctx.clearRect(0, 0, canvasSize, canvasSize);

1. **ctx.clearRect()**: This clears the entire canvas, preparing it for the next frame of the game.

javascript

Copy code

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

ctx.fillStyle = i === 0 ? "green" : "lime";

ctx.fillRect(snake[i].x, snake[i].y, box, box);

}

1. **Draw the Snake**: This loop draws each segment of the snake on the canvas. The snake's head is colored green, and the rest of the body is lime.

javascript

Copy code

ctx.fillStyle = "red";

ctx.fillRect(food.x, food.y, box, box);

1. **Draw the Food**: The food is drawn as a red square at its current position.

javascript

Copy code

ctx.fillStyle = "black";

ctx.font = "20px Arial";

ctx.fillText(`Score: ${score}`, 10, 20);

}

1. **Draw the Score**: The score is displayed at the top-left corner of the canvas.

### **Starting the Game Loop**

javascript

Copy code

gameInterval = setInterval(gameLoop, 100);

1. **setInterval(gameLoop, 100)**: This starts the game loop, which calls the gameLoop() function every 100 milliseconds (10 frames per second). The gameInterval variable stores the interval ID, allowing the game to stop when the game ends.

This is how the entire snake game functions! It checks for key inputs, updates the snake's movement and game state, and renders the snake, food, and score on the canvas, while checking for collisions to determine if the game should end.