

JAVASCRIPT GAME

**IC 1203 - Web Application Development**

**Level 1 – Semester 2**

**Student Index: 2021T01229**

**Student Name: W.M.D.PRIYANATH**

**Department of Information and Communication Technology**

**Faculty of Technology**

**University of Colombo**

**Table of Contents**

[**Introduction** 1](#_Toc139144385)

[**Methodology** 2](#_Toc139144386)

[**User Interface** 2](#_Toc139144387)

[**Program Flow** 5](#_Toc139144388)

[**Game Mechanics and Logic** 9](#_Toc139144389)

[**Appendix A: The Python program** 13](#_Toc139144390)

# **Introduction**

This project is creating a simple JavaScript game. In this game player has to avoid getting hit by objects that falling above game character. This game Application has developed using Html, CSS and JavaScript languages. User can input their name and start the game.in the end the final score will be printed on a text file.



# **Methodology**

## **User Interface**

1. This section focuses on describing the user interface elements of the "Doodle Hopper" program. It explains the different components that make up the program's user interface, providing details on their appearance and functionality.

<div id="game-container">

    <div id="player"></div>

    <div id="score-display"></div>

  </div>

  <div>

    <label for="name-input" id="label">Enter your name:</label>

    <input type="text" id="name-input">

    <button id="start-button">Click Here to start the game</button>

    <pre><b>Instructions

             - avoid collisions

             - use left and right arrow keys to move

    </b></pre>

  </div>

The **<div>** elements with IDs “game-container”, “player” and “score-display” represent the main game area and score display area, respectively.

The **<div>** elements with “label”, “name-input” and “start-button” represent the user input area ,instructions on how to play and start game button

A white rectangular object with black text

Description automatically generated with low confidence

Figure 1 :instructions, input name and start game button

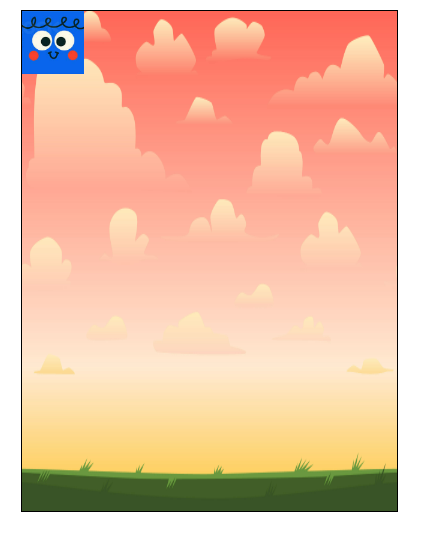


Figure 2 : game container

1. CSS styling

<style>

    #game-container {

      width: 300px;

      height: 400px;

      border: 1px solid black;

      position: relative;

      margin: 0 auto;

      background-image: url("888.jpg");

      background-size: 100% 100%;

      overflow: hidden;

    }

    #player {

      width: 50px;

      height: 50px;

      position: absolute;

      background-image: url("Screenshot 2023-06-26 104131.png");

      background-size: 100% 100%;

    }

    #score-display {

      position: absolute;

      top: 10px;

      left: 10px;

    }

  </style>

The CSS code defines the visual styling of various elements in the program.

**Program Flow**

This section focuses on describing the flow of the "Doodle Hopper" program. It explains the sequence of actions and events that occur during the execution of the program, including user interactions and game mechanics.

    // Get references to HTML elements

    const gameContainer = document.getElementById('game-container');

    const player = document.getElementById('player');

    const scoreDisplay = document.getElementById('score-display');

    const nameInput = document.getElementById('name-input');

    const startButton = document.getElementById('start-button');

    const label = document.getElementById('label');

    // Set initial player position

    let playerX = 125;

    let playerY = 350;

    // Set initial game state

    let isGameOver = false;

    let score = 0;

    let playerName = '';

This code segment initializes various variables and obtains references to important HTML elements needed to manage and control the game.

Next code segment will be used to track the vertical and horizontal position of the player's avatar on the game container.

And the last code segment initializes the variable “**isGameOver**” to false. This variable will be used to track whether the game is over or still in progress. “**let score = 0**;” This line initializes the variable score to 0. This variable will be used to track the player's score during the game. **let playerName = '';** This line initializes the variable playerName as an empty string. This variable will be used to store the player's name, which will be entered through the name input before starting the game.

    // Start the game when the button is clicked

    startButton.addEventListener('click', function() {

      playerName = nameInput.value;

      if (playerName.trim() !== '') startGame();

    });

When the button is clicked, the listener function is triggered. It retrieves the value from the *nameInput field*, assigns it to the *playerName* variable, and initiates the game by calling the **startGame ()** function if a non-empty name is provided.

    // Function to start the game

    function startGame() {

      // Hide the name input and start button

      nameInput.style.display = 'none';

      startButton.style.display = 'none';

      label.style.display = 'none';

this is the main function of the game. At first it hides the text field and start game button when playing the game.

      // Set up keyboard controls

      document.addEventListener('keydown', function(event) {

        if (!isGameOver) {

          if (event.key === 'ArrowLeft' && playerX > 0) moveLeft();

          else if (event.key === 'ArrowRight' && playerX < 250) moveRight();

        }

      });

Another event listener is added to the document for the **keydown** event. It captures keyboard inputs. Within the listener function, it checks if the game is not over **(!isGameOver flag)** and performs actions based on the pressed arrow keys. If the left arrow key is pressed and the player's position **(playerX)** or player’s x axis is greater than 0, the **moveLeft()** function is called to move the player to the left. Similarly, if the right arrow key is pressed and the player's position is less than 250, the **moveRight()** function is called to move the player to the right.

      // Function to move the player left

      function moveLeft() {

        playerX -= 5;

        player.style.left = playerX + 'px';

      }

      // Function to move the player right

      function moveRight() {

        playerX += 5;

        player.style.left = playerX + 'px';

      }

They update the playerX variable by subtracting or adding 5 units and adjust the position of the player element **(player.style.left)** accordingly.

      // Game loop

      function gameLoop() {

        movePlatforms();

        checkCollision();

        if (!isGameOver) requestAnimationFrame(gameLoop);

      }

The **gameLoop()** function represents the main game loop that continuously runs during gameplay. It calls the **movePlatforms()** function to move the platforms downward, followed by the **checkCollision()** function to detect collisions between the player and platforms. If the game is not over, the function requests the next frame of the animation using **requestAnimationFrame(gameLoop)** to create a smooth animation loop.

**Game Mechanics and Logic**

This section focuses on describing the game mechanics and logic implemented in the "Doodle Hopper" game. It explains the key functions and processes that govern the gameplay, including *platform generation, player movement, collision detection, scoring, and game over* conditions.

      // Function to generate random platforms

      function generatePlatforms() {

        const platformWidth = 75;

        const platformHeight = 10;

        const platformGap = 100;

        const platformCount = Math.floor(gameContainer.clientHeight / (platformHeight + platformGap));

        for (let i = 0; i < platformCount; i++) {

          const platform = document.createElement('div');

          platform.classList.add('platform');

          platform.style.width = platformWidth + 'px';

          platform.style.height = platformHeight + 'px';

          platform.style.backgroundColor = 'green';

          platform.style.position = 'absolute';

          platform.style.left = Math.random() \* (gameContainer.clientWidth - platformWidth) + 'px';

          platform.style.top = (platformHeight + platformGap) \* i + 'px';

          gameContainer.appendChild(platform);

        }

      }

The **generatePlatforms()** function handles the generation of platforms within the game container. It generates a specified number of platforms with random positions and adds them to the game container

      // Function to move platforms downwards

      function movePlatforms() {

        const platforms = gameContainer.querySelectorAll('.platform');

        platforms.forEach(function(platform) {

          let top = parseInt(platform.style.top);

          top += 2;

          if (top > gameContainer.clientHeight) {

            top = -10;

            platform.style.left = Math.random() \* (gameContainer.clientWidth - parseInt(platform.style.width)) + 'px';

          }

          platform.style.top = top + 'px';

        });

      }

The **movePlatforms()** function is responsible for moving the platforms downwards. It iterates over the platforms in the game container, updates their positions by incrementing the top value, and handles platform repositioning when they reach the bottom of the game container.

      // Function to check for collision between player and platforms

      function checkCollision() {

        const platforms = gameContainer.querySelectorAll('.platform');

        platforms.forEach(function(platform) {

          const playerRect = player.getBoundingClientRect();

          const platformRect = platform.getBoundingClientRect();

          if (

            playerRect.bottom >= platformRect.top &&

            playerRect.top <= platformRect.bottom &&

            playerRect.right >= platformRect.left &&

            playerRect.left <= platformRect.right

          ) {

              // Player hit the platform from the bottom

              playerY = platformRect.bottom;

              player.style.top = playerY + 'px';

          }

        });

        // Check if the player fell off the screen

        if (playerY > gameContainer.clientHeight) gameOver();

      }

The **checkCollision()** function detects collisions between the player and platforms. It uses the **getBoundingClientRect()** method to obtain the dimensions and positions of the player and each platform. By comparing the bounding rectangles of the player and platforms, it determines if a collision has occurred. If a collision is detected, the player's position is adjusted to align with the top of the platform.

      // Function to increase the score

      function increaseScore() {

        score++;

        scoreDisplay.textContent = playerName + "'s Score: " + score;

      }

The **increaseScore()** function is responsible for increasing the player's score. It increments the score variable and updates the score display element *(scoreDisplay.textContent)* with the updated score.

      // Function to end the game

      function gameOver() {

        isGameOver = true;

        clearInterval(scoreInterval); // Stop the score incrementing

        saveScore();

        alert('Game Over. Your score: ' + score);

      }

      // Function to save the score in a text file

      function saveScore() {

        const textToSave = playerName + "'s Score: " + score;

        const filename = 'score.txt';

        const blob = new Blob([textToSave], { type: 'text/plain' });

        // Create a temporary link element to trigger the file download

        const link = document.createElement('a');

        link.href = URL.createObjectURL(blob);

        link.download = filename;

        link.click();

      }

The **gameOver()** function handles the game over condition. It sets the isGameOver flag to true, stops the score incrementing by clearing the score interval, saves the player's score by calling the **saveScore()** function, and displays an alert with the player's final score. And The **saveScore()** function saves the player's score in a text file. It creates a Blob object with the score content and file type, then creates a temporary link element with the necessary attributes to trigger the file download.

      // Start the game

generatePlatforms();

      gameLoop();

      // Update the score every second

      const scoreInterval = setInterval(increaseScore, 1000);

    }

By calling **generatePlatforms();** and **gameLoop();** functionsthe game keeps running and adding 1 score every 1000 millisecond.

# **Appendix A: The JS program**

* The complete Program with functions folded. (200 Lines)

A screen shot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence