GAME MANUAL

FLAPPY FROG

MANUAL AND GAME CREATED BY: YAZU BAUTISTA

DESIGN AND LAYOUT: MAURICIO CASTILLO

TRANSLATED BY: ALEXANDRA DIAZ

"FLAPPY FROG" Game Manual

Introduction

The Flappy Frog game is a version inspired by Flappy Bird, but starring a frog. The objective is for the player to keep the frog flying between green pipes without crashing or falling to the ground. Each pipe that is successfully passed increases the score.

The game is developed using HTML, CSS, and JavaScript, three essential languages for creating video games and web pages.

1. Required documents

For the game to work correctly, the following three files and two images must be in the same folder:

index.html → Contains the main structure of the game.

style.css → Defines the colors, sizes, and positions (the visual style)

game.js → Contains the game's programming and functions.

1.png → Image of the frog (player)

2.png → Image of the stage background.

General Structure

```
<!DOCTYPE html>
<html lang="es">
```

● <!DOCTYPE html> → Indicates that the document is written in HTML5, the most modern version.
● <html lang="es"> → Indicates that the primary language of the page is Spanish. <head> Section

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flappy Frog</title>
    <link rel="stylesheet" href="style.css">
</head>
```

This section contains the document's internal configuration, which is not visible on the screen but is very important.
<meta charset="UTF-8"> → Allows the use of special characters (like "ñ" or accents)
<meta name="viewport"...> → Ensures the game displays correctly on phones and computers (responsive adjustment)
<title> → Shows the name "Flappy Frog" in the browser tab.
<link rel="stylesheet" href="style.css"> → Connects the CSS style file with the HTML to apply colors and positions.

Main Body <body>

```
dy>
 <div id="game-container">
     <canvas id="gameCanvas"></canvas>
     <div id="score-container">
         Puntuación: <span id="score">0</span>
     </div>
     <button id="start-btn" onclick="startGame()">Iniciar</button>
     <button id="restart-btn" style="display:none" onclick="restartGame()">Reiniciar</but
 </div>

 <script src="game.js"></script>
ody>
tml>
```

Everything the player will see on the screen is placed inside the <body>

This is the main container for the game
◯ It holds all the elements that compose the game: the game area (canvas), the score, and the buttons.
<canvas id="gameCanvas"></canvas>
● This is the canvas where the frog, pipes, and background are drawn.
● All game movement occurs here thanks to the JavaScript code.
● It has no content of its own because everything is drawn dynamically.

```
<div id="score-container">
    Puntuación: <span id="score">0</span>
</div>
```

● Displays the player's current score.
● The text "Puntuación:" ("Score:") is fixed, while the number inside increases with every pipe passed.
● The initial value is 0

•

"Start" Button

```
<button id="start-btn" onclick="startGame()">Iniciar</button>
```

This is the green button that appears at the start.
Clicking it executes the startGame() function (defined in game.js) to start the game.
"Restart" Button

```
<button id="restart-btn" style="display:none" onclick="restartGame()">Reiniciar</button
```

This is the red button that only appears when the player loses.
It has an initial style of display:none, meaning it is hidden at the beginning. When displayed, clicking it calls the restartGame() function to reset the game and start over.
JavaScript Link

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

● Connects the programming file (game.js) with the HTML.
● This file contains all the game logic (movement, collisions, scoring, etc.) ● It is the last element in the document, ensuring it loads after all the HTML is ready.
Document Closure

```
</body>
</html>
```

● </body> → Closes the document body60.
- ● </html> → Marks the complete end of the HTML file61.

2. CSS Document – style.css

This document controls the visual appearance of the game: colors, element positions, sizes, and fonts.

General Configuration

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

● Removes default margins and padding from all elements.

● Ensures the design is uniform across any browser.

Site Body Design

```css
body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #87CEFA; /* Cielo celeste */
    font-family: Arial, sans-serif;
}
```

● Uses flexbox to center the content.

● background-color applies a light blue background simulating the sky. ● height: 100vh ensures it occupies the entire screen.

Game Container

```css
#game-container {
    position: relative;
    width: 100%;
    height: 100%;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
```

● Keeps everything centered on the screen.

● position: relative allows the buttons and score to be placed on top of the canvas.

Game Area (Canvas)

```css
#gameCanvas {
    border: 2px solid #000;
    display: block;
    background-color: #87CEFA;
    position: absolute;
}
```

● border: gives a frame to the game area.
● position: absolute: allows other elements (like buttons) to be positioned on top.

Score

```css
#score-container {
    font-size: 24px;
    color: #FFFFFF;
    position: absolute;
    top: 10px;
}
```

● Displays the text “Puntuación: 0” in the upper part.
● position: absolute makes it fixed without moving with the canvas.

Buttons

```css
#start-btn, #restart-btn {
    padding: 10px 20px;
    font-size: 16px;
    border-radius: 5px;
    cursor: pointer;
    position: absolute;
    bottom: 50px;
}
```

● Both buttons are positioned at the bottom
● cursor: pointer changes the cursor when hovering over the button. ● border-radius gives them rounded edges.

"Start" Button

```css
#start-btn {
    background-color: #28a745;
    color: white;
}
#start-btn:hover {
    background-color: #218838;
}
#start-btn:active {
    background-color: #1e7e34;
}
```

● Green = starts the game

- ● Uses different shades when hovered over or pressed.

"Restart" Button

```css
#restart-btn {
    background-color: #dc3545;
    color: white;
    display: none;
}
#restart-btn:hover {
    background-color: #c82333;
}
#restart-btn:active {
    background-color: #bd2130;
}
```

● Red = restarts the game after losing.
● It is hidden initially (display: none)

3. JavaScript Document – game.js

This file contains the game's programming: control of movement, gravity, collisions, pipes, scoring, and restarting.

Initial Setup

```js
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');
canvas.width = window.innerWidth;
canvas.height = window.innerHeight;
```

● Gets the canvas where the game is drawn102.
● ctx is the 2D context for drawing images and shapes. ●
Adjusts the game size to the full screen.

Image Loading

```
const backgroundImage = new Image();
backgroundImage.src = '2.png'; // Fondo del juego

const frogImage = new Image();
frogImage.src = '1.png'; // Imagen de la rana
```

These images must be saved in the same folder as the other files. Object “Frog”

```
let frog = {
    x: 50,
    y: canvas.height / 2,
    width: 100,
    height: 100,
    velocity: 0,
    flap() {
        this.velocity = -10; // Salto
    },
    update() {
        this.velocity += 0.5; // Gravedad
        this.y += this.velocity;
    },
    draw() {
        ctx.drawImage(frogImage, this.x, this.y, this.width, this.height);
    }
};
```

● The frog has a position (x, y), size, and movement. ● flap(): simulates the jump.
● update(): applies gravity.
● draw(): draws the frog on the screen.
Pipe Creation and Drawing

```
let pipes = [];

function createPipe() {
    const gap = 250;
    const height = Math.random() * (canvas.height - gap - 100) + 50;
    const pipeTop = { x: canvas.width, y: 0, width: 60, height: height };
    const pipeBottom = { x: canvas.width, y: height + gap, width: 60, height: canvas.he
    pipes.push(pipeTop, pipeBottom);
}
```

● Creates pipes with a gap between them.
● Math.random() ensures each pipe has a different height.

```
function drawPipes() {
    ctx.fillStyle = '#00FF00';
    pipes.forEach(pipe => {
        ctx.fillRect(pipe.x, pipe.y, pipe.width, pipe.height);
    });
}
```

● drawPipes(): Paints the pipes green.
● They are updated in every frame to simulate movement.

Collisions

```
function detectCollisions() {
    pipes.forEach(pipe => {
        if (
            frog.x < pipe.x + pipe.width &&
            frog.x + frog.width > pipe.x &&
            frog.y < pipe.y + pipe.height &&
            frog.y + frog.height > pipe.y
        ) {
            gameOver = true;
            showGameOver();
        }
    });
}
```

```
let score = 0;

function updateScore() {
    pipes.forEach(pipe => {
        if (pipe.x + pipe.width < frog.x && !pipe.passed) {
            score++;
            pipe.passed = true;
        }
    });
    document.getElementById('score').textContent = score;
}
```

● Detects if the frog touches any pipe.
● If it crashes, it shows the Game Over message.
Scoreboard and Restart
● updateScore(): Increases the score every time a pipe is passed ● Displays the updated value on the screen
Main functions of the game

```
function startGame() {
    gameStarted = true;
    document.getElementById('start-btn').style.display = 'none';
    gameLoop();
}

function restartGame() {
    pipes = [];
    frog.y = canvas.height / 2;
    frog.velocity = 0;
    score = 0;
    gameOver = false;
    frame = 0;
    document.getElementById('score').textContent = score;
    document.getElementById('restart-btn').style.display = 'none';
    gameLoop();
}
```

● startGame() → starts the game.
● restartGame() → resets everything when the player loses.

Main Loop

```
function gameLoop() {
    frame++;
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.drawImage(backgroundImage, 0, 0, canvas.width, canvas.height);
    if (frame % 90 === 0 && !gameOver) createPipe();
    frog.update(); frog.draw();
    drawPipes(); detectCollisions(); updateScore();
    if (!gameOver) requestAnimationFrame(gameLoop);
}
```

- This loop constantly updates and redraws everything.
- It controls the game's pace and generates fluid movement.

Conclusion

The "Flappy Frog" game demonstrates how to combine HTML, CSS, and JavaScript to create a simple, fun, and visually appealing video game.

Each document fulfills an essential role:

- HTML → Game structure
- CSS → Appearance and design
- JS → Movement, collisions, and logic

The result is a complete interactive experience developed with basic web technologies.