



# **TECNOLÓGICO DE ESTUDIOS SUPERIORES DE ECATEPEC**

---

**INGENIERÍA INFORMÁTICA**

**PRESENTA**

**Uriel Gael Gómez Ferreyra**

**MATRICULA**

**202221988**

**MATERIA**

**Desarrollo de Aplicaciones Web**

**PROFESOR**

**Leonardo Miguel Moreno Villalba**

**GRUPO**

**15601**

**023/09/2025**

# Proyecto Integrador - Juego tipo “Dinosaurio de Google”

## Objetivo

Aplicar los conocimientos de HTML5, CSS3, JavaScript y un servidor en Node.js para desarrollar un proyecto completo que incluya tanto la parte visual/interactiva como la parte de almacenamiento de datos. El reto consiste en crear un juego estilo “dinosaurio de Google” (endless runner), que cuente con niveles progresivos de dificultad y un sistema de puntuaciones (scores) que se guarde en un servidor y se muestre en un Leaderboard (tabla de las mejores puntuaciones).

## Descripción del Proyecto

El proyecto cuenta con una Landing Page (index.html) que incluye un menú de navegación hacia las prácticas, el juego integrador y enlaces al repositorio en GitHub. El juego, desarrollado en canvas, permite al jugador saltar obstáculos mientras la velocidad y la dificultad aumentan progresivamente. Se incluye un sistema de puntuación y leaderboard conectado a una API en Node.js que guarda los puntajes en un archivo JSON, con soporte a localStorage como respaldo.

# Explicación de los temas aplicados

En el desarrollo de este proyecto se aplicaron diversos temas y conceptos fundamentales tanto de programación web como de diseño de videojuegos sencillos, los cuales se integraron de manera práctica para lograr un producto funcional. A continuación, se describen con detalle los principales:

## 1. Uso del elemento Canvas de HTML5

El canvas fue el núcleo visual del juego. Se utilizó para renderizar en tiempo real todos los gráficos: personaje, obstáculos, plataformas y escenarios. Mediante el contexto 2D (`getContext("2d")`), se emplearon métodos como `fillRect`, `drawImage` y `clearRect` para dibujar y actualizar la escena de manera dinámica. Esta técnica permitió simular movimiento fluido, detectar colisiones y ofrecer una experiencia de juego interactiva directamente desde el navegador sin necesidad de librerías externas.

## 2. Programación con JavaScript orientada a eventos

El juego se basó en la captura de eventos del teclado (por ejemplo, saltar con la tecla de espacio). Para ello, se usaron listeners como `addEventListener("keydown", ...)` que permitieron controlar la interacción del jugador en tiempo real. Esto garantizó que la respuesta del juego fuera inmediata y natural, simulando la lógica de interacción típica de los videojuegos.

## 3. Gestión de la lógica de niveles y dificultad

Se implementó un sistema de progresión de niveles en el que, conforme el jugador avanzaba, aumentaba la velocidad de los obstáculos y la complejidad de las plataformas. Este mecanismo fue diseñado para mantener la motivación del usuario, iniciando con un nivel accesible y aumentando gradualmente la dificultad, lo que promueve la rejugabilidad y el desafío constante.

## 4. Detección de colisiones

Para determinar cuándo el personaje chocaba con un obstáculo o caía fuera de la plataforma, se aplicaron métodos de detección de colisiones basados en bounding boxes rectangulares. Concretamente, se compararon las coordenadas de los objetos en el lienzo para identificar intersecciones. Esta técnica, aunque sencilla, resultó suficiente para un juego en 2D y permitió mantener un rendimiento adecuado.

## **5. Almacenamiento de puntuaciones con JSON**

Las puntuaciones y registros de jugadores se manejaron en formato JSON. Esto facilitó guardar información estructurada con campos como name, score, level y date. El formato JSON fue elegido por su simplicidad, su compatibilidad con JavaScript y su facilidad para persistir datos en archivos o APIs, garantizando además la posibilidad de ampliarlo en el futuro.

## **6. Leaderboard conectado al servidor**

Para la gestión de los puntajes más altos, se diseñó un Leaderboard (tabla de clasificación) que no solo dependía de localStorage, sino que además se conectaba con un archivo scores.json simulado como API. Esto permitió comprender la diferencia entre guardar datos localmente en el navegador y almacenarlos en el servidor, reforzando la importancia de la persistencia de información y la integridad de los datos.

## **7. Modularización del código**

Aunque se trató de un proyecto compacto, se aplicaron principios de modularización en el código. La lógica del juego, el renderizado, la detección de colisiones y la gestión de puntajes se organizaron en funciones separadas. Esto se hizo con el objetivo de mejorar la legibilidad, la mantenibilidad y la escalabilidad del proyecto. Gracias a esta separación, el código puede ser modificado o ampliado sin afectar otras partes del sistema.

## **8. Buenas prácticas de desarrollo**

Además de la funcionalidad, se aplicaron prácticas como el comentario del código para explicar secciones clave, el uso de nombres de variables descriptivos, y la validación de datos en el Leaderboard para evitar entradas incorrectas. Estas prácticas no solo facilitan la comprensión para otros desarrolladores, sino que también contribuyen a la calidad general del proyecto.

En resumen, el proyecto integra de manera práctica y didáctica los conceptos de programación en Canvas, manejo de eventos, diseño de niveles, colisiones, persistencia de datos en JSON, almacenamiento en servidor y buenas prácticas de modularización, logrando un ejemplo funcional de videojuego 2D que puede ser expandido con nuevas características en el futuro.

# **Diagramas de Maquetado**

bosquejo de la Landing Page con menú de navegación y tarjetas

## HEADER

Inicio

Practicas

Proyecto

### Main

Bienvenido

Este repositorio contiene tus practicas y el proyecto integrador: juego Tipo Dinosaurio

Tarjetas de practica y GitHub

Practica

Abrir

GitHub

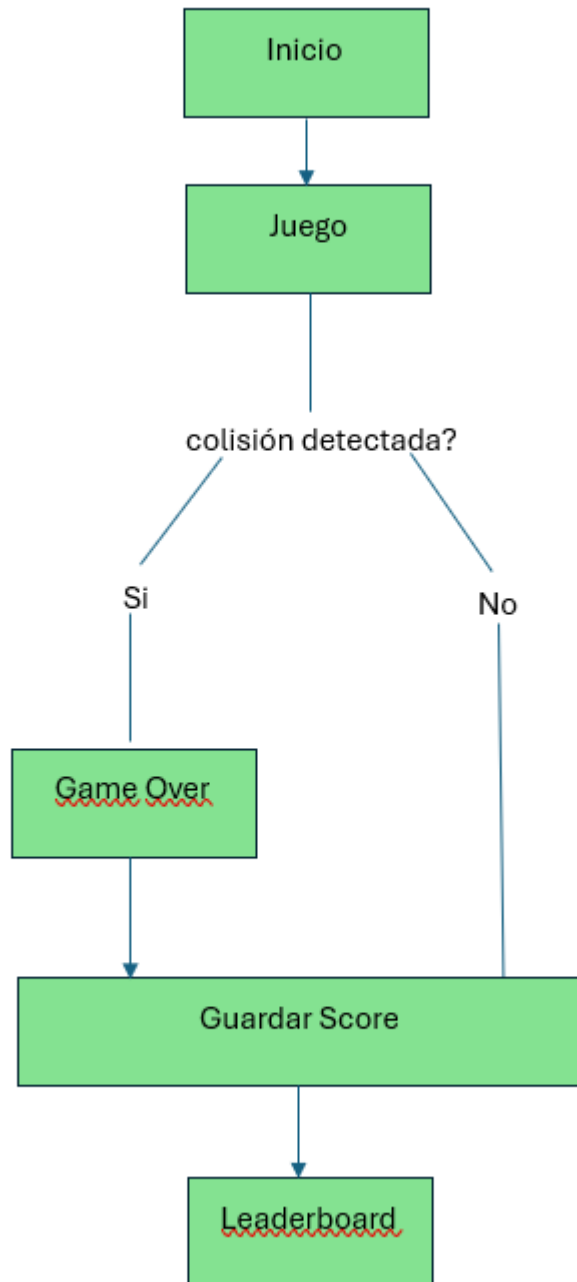
Abrir

Tarjeta del juego

RunnerJS – Juego Integrador

Jugar

diagrama de flujo del juego



## Código fuente comentado

**Index.html**

```
<!doctype html>
```

```
<html lang="es">

<head>

  <meta charset="utf-8" />

  <meta name="viewport" content="width=device-width,initial-scale=1" />

  <title>Proyecto Integrador - RunnerJS</title>

  <link rel="stylesheet" href="proyecto/game.css">

  <style>

    /* Estilos basicos del landing para que sea funcional */

    body{font-family: Arial, Helvetica, sans-serif; margin:0; background:#f4f7fb;
color:#222}

    header{background:#178b62; padding:16px; color:white; display:flex; justify-
content:space-between; align-items:center}

    nav a{color:white; margin:0 10px; text-decoration:none; font-weight:600}

    .container{max-width:1000px; margin:24px auto; padding:0 16px}

    .cards{display:grid; grid-template-columns:repeat(auto-fit,minmax(220px,1fr));
gap:16px}

    .card{background:white; padding:14px; border-radius:10px; box-shadow:0 6px 18px
rgba(0,0,0,.06)}

    .card a{display:block; text-decoration:none; color:#178b62; font-weight:700}

    .play-btn{display:inline-block; margin-top:8px; padding:8px 12px;
background:#21acbb; color:white; border-radius:8px}

  </style>

</head>

<body>

  <header>

    <div><strong>RunnerJS - Proyecto Integrador</strong></div>
```

```
<nav>
```

```
<a href="#inicio">Inicio</a>
```

```
<a href="/practicas">Practicas</a>
```

```
<a href="#proyecto">Proyecto</a>
```

```
</nav>
```

```
</header>
```

```
<main class="container">
```

```
<h1 id="inicio">Bienvenido</h1>
```

```
<p>Este repositorio contiene tus practicas y el Proyecto Integrador: Juego tipo  
"Dinosaurio".</p>
```

```
<h2 id="practicas">Practicas</h2>
```

```
<div class="cards">
```

```
<div class="card">
```

```
<div>Ver todas las prácticas</div>
```

```
<a href="/practicas" target="_blank">Abrir Practicas</a>
```

```
</div>
```

```
<!-- 📌 Nuevo cuadrito para GitHub -->
```

```
<div class="card">
```

```
<div>Archivos en GitHub</div>
```

```
<a href="https://github.com/urielZTN/Juego-tipo-Dinosaurio-de-Google-"  
target="_blank">Abrir en GitHub</a>
```

```
</div>
```



```
</div>

<h2 id="proyecto">Proyecto</h2>

<div class="cards" style="margin-top:12px">

  <div class="card">

    <div>RunnerJS - Juego Integrador</div>

    <a class="play-btn" href="proyecto/game.html" target="_blank">Jugar</a>

    <p style="margin-top:8px">Entrar al juego y leaderboard</p>

  </div>

</div>

</main>

</body>

</html>
```

### **Server.js**

```
// backend/server.js

const express = require("express");
const bodyParser = require("body-parser");
const path = require("path");

const app = express();

const PORT = 3000;


const scoresFile = path.join(__dirname, "scores.json");
```

```
// Middleware para JSON
```

```
app.use(bodyParser.json());
```

```
// Servir archivos estáticos desde /frontend
```


```
app.use(express.static(path.join(__dirname, "../frontend")));
```

```
//  Ruta para el index.html (landing)
```

```
app.get("/", (req, res) => {
```

```
  res.sendFile(path.join(__dirname, "../frontend/index.html"));
```

```
});
```

```
//  Ruta directa para el game.html
```

```
app.get("/proyecto/game.html", (req, res) => {
```

```
  res.sendFile(path.join(__dirname, "../frontend/proyecto/game.html"));
```

```
});
```

```
// --- API de scores ---
```

```
let scores = [];
```

```
// Obtener scores
```

```
app.get("/scores", (req, res) => {
```

```
  res.json(scores);
```

```
});
```

```
// Guardar nuevo score

app.post("/scores", (req, res) => {

  const { name, score } = req.body;

  if (name && score !== undefined) {

    scores.push({ name, score });

    scores.sort((a, b) => b.score - a.score); // ordenar de mayor a menor

    res.json({ message: "Score guardado", scores });

  } else {

    res.status(400).json({ error: "Datos inválidos" });

  }

});

const serveIndex = require("serve-index");

// Ruta para ver todas las prácticas

app.use(

  "/practicas",


  express.static(path.join(__dirname, "../frontend/practicas")),

  serveIndex(path.join(__dirname, "../frontend/practicas"), { icons: true })

);

// Iniciar servidor

app.listen(PORT, () => {
```

```
console.log(`  Servidor corriendo en: http://localhost:${PORT} `);  
});
```

### Api.js

```
// API client para leaderboard
```

```
// Cambia API_URL por la URL donde despliegues el backend (ej: https://mi-backend.onrender.com)
```

```
const API_URL = "https://REPLACE_WITH_YOUR_BACKEND_URL"; // <-- cambiar
```

```
async function fetchScores(limit = 10) {
```

```
  try {
```

```
    const res = await fetch(`${API_URL}/api/scores?limit=${limit}`);
```

```
    if (!res.ok) throw new Error("Error al solicitar scores");
```

```
    const data = await res.json();
```

```
    return data;
```

```
  } catch (err) {
```

```
    console.warn("fetchScores fallo, usando localStorage:", err.message);
```

```
    // fallback a localStorage
```

```
    const local = localStorage.getItem("runner_local_scores");
```

```
    return local ? JSON.parse(local) : [];
```

```
  }
```

```
}
```

```
async function postScore(entry) {
```

```
  // entry: { name, score, level, date }
```

```

try {

  const res = await fetch(`${API_URL}/api/scores`, {

    method: "POST",

    headers: { "Content-Type": "application/json" },

    body: JSON.stringify(entry)

  });

  if (!res.ok) throw new Error("Error al guardar score en servidor");

  const data = await res.json();

  return data;

} catch (err) {

  console.warn("postScore fallo, guardando en localStorage:", err.message);

  // fallback: guardar en localStorage

  const key = "runner_local_scores";

  const arr = JSON.parse(localStorage.getItem(key) || "[]");

  arr.push(entry);

  // mantener solo top 50 localmente

  arr.sort((a,b)=> b.score - a.score);

  localStorage.setItem(key, JSON.stringify(arr.slice(0,50)));

  return { ok: false, fallback: true };

}

}

```

## Game.html

```
<!doctype html>
```

```
<html lang="es">

<head>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width,initial-scale=1">

  <title>RunnerJS - Juego</title>

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

</head>

<body>

  <header class="topbar">

    <div class="title">RunnerJS</div>

    <div class="hud">

      <span id="score">Score: 0</span>

      <span id="level">Nivel: 1</span>

      <span id="best">Max: 0</span>

    </div>

  </header>

  <main class="game-area">

    <canvas id="gameCanvas" width="800" height="200"></canvas>

    <div id="overlay" class="overlay hidden">

      <div class="modal">

        <h2 id="gameOverTitle">Game Over</h2>

        <p id="finalScore">Puntaje: 0</p>

      </div>

    </div>

  </main>

</body>

</html>
```

```
<p id="finalLevel">Nivel: 1</p>

<div class="actions">

  <button id="restartBtn">Reiniciar</button>

</div>

<div class="saveScoreSection">

  <input id="playerName" placeholder="Tu nombre (max 20)" maxlength="20">

  <button id="saveBtn">Guardar Score</button>

  <p id="saveMsg" class="muted"></p>

</div>

</div>

</div>

<section class="leaderboard">

  <h3>Leaderboard</h3>

  <div id="leaderboardList">Cargando...</div>

</section>

</main>

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

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

</body>

</html>
```

**Game.css**

```
/* estilos juego - sencillo y limpio */
```

```
body{font-family: Arial, Helvetica, sans-serif; margin:0; background:#eef3f8;  
color:#222}
```

```
.topbar{display:flex; justify-content:space-between; align-items:center; padding:12px  
20px; background:#fff; box-shadow:0 2px 6px rgba(0,0,0,.06)}
```

```
.title{font-weight:700}
```

```
.hud span{margin-left:12px; font-weight:600}
```

```
.game-area{max-width:900px; margin:18px auto; padding:12px}
```

```
canvas{display:block; width:100%; background:linear-gradient(#dff0ff,#fff); border-  
radius:8px; box-shadow:0 6px 18px rgba(23,139,98,0.06)}
```

```
.overlay{position:fixed; inset:0; display:flex; justify-content:center; align-  
items:center; background:rgba(0,0,0,0.4)}
```

```
.hidden{display:none}
```

```
.modal{background:#fff; padding:18px; border-radius:8px; width:320px; text-  
align:center}
```

```
.actions{margin:10px 0}
```

```
button{padding:8px 12px; border-radius:8px; border:none; cursor:pointer;  
background:#21acbb; color:white}
```

```
.saveScoreSection{margin-top:10px}
```

```
input{padding:8px; width:70%; margin-right:6px}
```

```
.leaderboard{margin-top:12px; background:#fff; padding:12px; border-radius:8px}
```

```
.leaderboard table{width:100%; border-collapse:collapse}
```

```
.leaderboard th, .leaderboard td{padding:6px 8px; text-align:left}
```



```
.muted{color:#666; font-size:13px}
```

## **Game.js**

```
// Juego RunnerJS - endless runner simple
```

```
// Comentarios en español (sin acentos)
```

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

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

```
let WIDTH = canvas.width;
```

```
let HEIGHT = canvas.height;
```

```
const scoreEl = document.getElementById("score");
```

```
const levelEl = document.getElementById("level");
```

```
const bestEl = document.getElementById("best");
```

```
const overlay = document.getElementById("overlay");
```

```
const restartBtn = document.getElementById("restartBtn");
```

```
const saveBtn = document.getElementById("saveBtn");
```

```
const playerNameInput = document.getElementById("playerName");
```

```
const finalScore = document.getElementById("finalScore");
```

```
const finalLevel = document.getElementById("finalLevel");
```

```
const saveMsg = document.getElementById("saveMsg");
```

```
const leaderboardList = document.getElementById("leaderboardList");
```

```
let running = false;

let gameOver = false;

let score = 0;

let level = 1;

let best = parseInt(localStorage.getItem("runner_best") || "0", 10);
```

```
// parametros de juego (ajustables)
```

```
let gravity = 0.8;

let jumpVel = -12;

let groundY = HEIGHT - 40;
```

```
let speed = 4;

let spawnTimer = 0;

let spawnInterval = 90; // frames
```

```
// jugador
```

```
const player = {

  x: 60,

  y: groundY - 40,

  w: 40,

  h: 40,

  vy: 0,

  onGround: true,

  draw(){
```

```
    ctx.fillStyle = "#178b62";

    ctx.fillRect(this.x, this.y, this.w, this.h);

},

update(){

    this.vy += gravity;

    this.y += this.vy;

    if(this.y + this.h >= groundY){

        this.y = groundY - this.h;

        this.vy = 0;

        this.onGround = true;

    } else {

        this.onGround = false;

    }

},

jump(){

    if(this.onGround){

        this.vy = jumpVel;

        this.onGround = false;

    }

}

};
```

```
let obstacles = [];
```

```
// obstaculo simple: rectangulo
```

```
function spawnObstacle(){  
  // variacion en tamano y tipo  
  const h = 20 + Math.random()*40;  
  const w = 12 + Math.random()*28;  
  obstacles.push({  
    x: WIDTH + 20,  
    y: groundY - h,  
    w, h  
  });  
}
```

```
// deteccion AABB (rectangulos)
```

```
function isColliding(a, b){  
  return !(a.x + a.w < b.x || a.x > b.x + b.w || a.y + a.h < b.y || a.y > b.y + b.h);  
}
```

```
// reset
```

```
function resetGame(){  
  obstacles = [];  
  score = 0;  
  level = 1;  
  speed = 4;  
  spawnInterval = 90;
```

```
spawnTimer = 0;

gameOver = false;

overlay.classList.add("hidden");

running = true;

loop();

}


// game over

function doGameOver(){

    running = false;

    gameOver = true;

    overlay.classList.remove("hidden");

    finalScore.textContent = `Puntaje: ${Math.floor(score)}`;

    finalLevel.textContent = `Nivel: ${level}`;

    // guardar mejor local

    if(score > best){

        best = Math.floor(score);

        localStorage.setItem("runner_best", best);

    }

    bestEl.textContent = `Max: ${best}`;

    // actualizar leaderboard visible

    loadLeaderboard();

}
```

```
// game loop

function loop(){

    if(!running) return;

    // update

    // aumentar score con el tiempo y la velocidad

    score += 0.1 * speed;

    spawnTimer++;

    if(spawnTimer >= spawnInterval){

        spawnObstacle();

        spawnTimer = 0;

        // ajustar spawn segun nivel/aleatoriedad

        spawnInterval = Math.max(40, 90 - level*6 - Math.floor(Math.random()*15));

    }


    // aumentar dificultad segun score

    const newLevel = Math.floor(score / 200) + 1;

    if(newLevel > level){

        level = newLevel;

        speed += 0.8; // aumentar velocidad

    }


    // update player y obstacles

    player.update();

    for(let i = obstacles.length -1; i>=0; i--){
```

```
    obstacles[i].x -= speed;

    if(obstacles[i].x + obstacles[i].w < 0) obstacles.splice(i,1);
}

// check collision
for(const obs of obstacles){
    if(isColliding(player, obs)){
        doGameOver();
        return;
    }
}

// draw
draw();

// update HUD
scoreEl.textContent = `Score: ${Math.floor(score)} `;
levelEl.textContent = `Nivel: ${level}`;
bestEl.textContent = `Max: ${best}`;

requestAnimationFrame(loop);
}

function draw(){
```

```
// limpiar

ctx.clearRect(0,0, WIDTH, HEIGHT);


// suelo

ctx.fillStyle = "#cfcfcf";

ctx.fillRect(0, groundY, WIDTH, HEIGHT - groundY);


// player

player.draw();


// obstacles

ctx.fillStyle = "#a63d3d";

for(const obs of obstacles){

  ctx.fillRect(obs.x, obs.y, obs.w, obs.h);

}


// puntaje (canvas)

ctx.fillStyle = "#222";

ctx.font = "12px Arial";

ctx.fillText(` Score: ${Math.floor(score)} Nivel: ${level}`, WIDTH - 140, 20);

}


// input

window.addEventListener("keydown", e=>{
```



```
if(e.code === "Space" || e.code === "ArrowUp"){
    e.preventDefault();
    if(!running && !gameOver){
        running = true;
        loop();
    } else if(gameOver){
        // no hacer nada, usar botones
    } else {
        player.jump();
    }
}
});

canvas.addEventListener("click", ()=> {
    if(!running && !gameOver){ running = true; loop(); }
    player.jump();
});

// boton reiniciar
restartBtn.addEventListener("click", ()=>{
    resetGame();
});

// guardar score
saveBtn.addEventListener("click", async ()=>{
```

```
const name = (playerNameInput.value || "Anonimo").trim().slice(0,20) || "Anonimo";

const entry = { name, score: Math.floor(score), level, date: new Date().toISOString() };


saveMsg.textContent = "Guardando...";

saveBtn.disabled = true;

try {

    const res = await postScore(entry);

    if(res && res.ok === false && res.fallback){

        saveMsg.textContent = "Guardado correctamente";

    } else {

        saveMsg.textContent = "Score guardado en servidor.";

    }

    // actualizar best local

    if(entry.score > best){

        best = entry.score;

        localStorage.setItem("runner_best", best);

    }

    // recargar leaderboard

    await loadLeaderboard();

} catch(e){

    saveMsg.textContent = "Error guardando score.";

    console.error(e);

} finally {

    saveBtn.disabled = false;
```

```

    }

  });

  // cargar leaderboard

  async function loadLeaderboard(){

    leaderboardList.innerHTML = "Cargando...";

    try {

      const arr = await fetchScores(10);

      if(!arr || arr.length === 0){

        leaderboardList.innerHTML = "<p>No hay scores aun</p>";

        return;

      }

      // crear tabla

      let html =

      "<table><thead><tr><th>#</th><th>Nombre</th><th>Score</th><th>Nivel</th></tr>
      </thead><tbody>";

      arr.slice(0,10).forEach((s,i)=>{

        html +=

        `<tr><td>${i+1}</td><td>${escapeHtml(s.name)}</td><td>${s.score}</td><td>${s.level||"-"}</td></tr>`;

      });

      html += "</tbody></table>";

      leaderboardList.innerHTML = html;

    } catch(e){

      leaderboardList.innerHTML = "<p>Error cargando leaderboard</p>";

      console.error(e);
    }
  }
}

```

```
}  
}
```

```
function escapeHtml(text){  
    return text.replace(/[\&<>"]/g, m =>  
    ({'&':'&amp;','<':'&lt;','>':'&gt;','"':'&quot;','\'':'&#039;'}[m]));  
}
```

```
// inicia vista
```

```
bestEl.textContent = `Max: ${best}`;
```

```
loadLeaderboard();
```

```
// si el servidor no esta configurado, mostrar aviso en consola
```

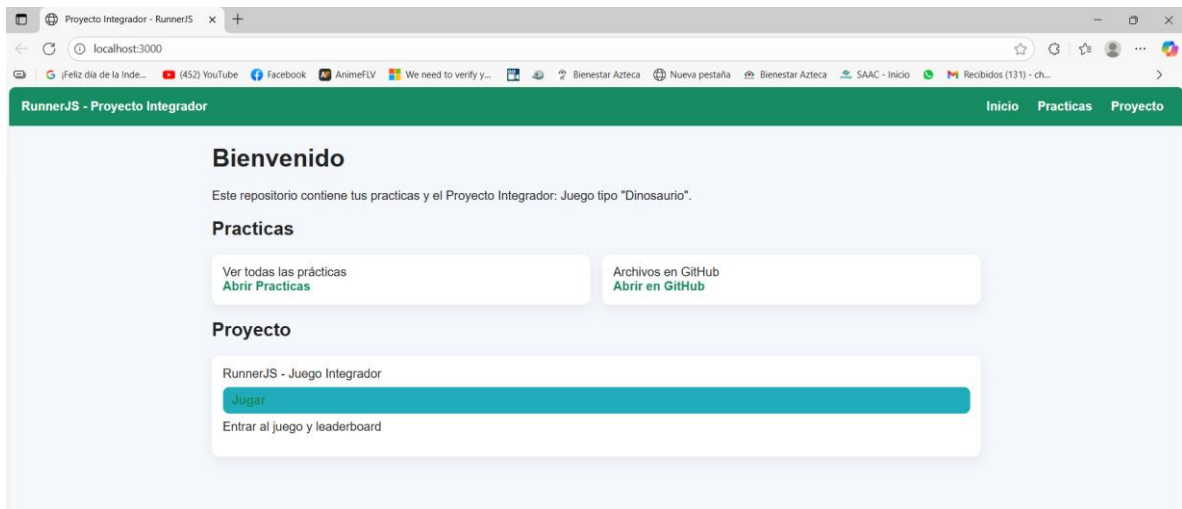
```
if(API_URL.includes("REPLACE_WITH")){
```

```
    console.warn("API_URL no configurada en api.js. Guarda scores en localStorage o  
    actualiza la URL.");
```

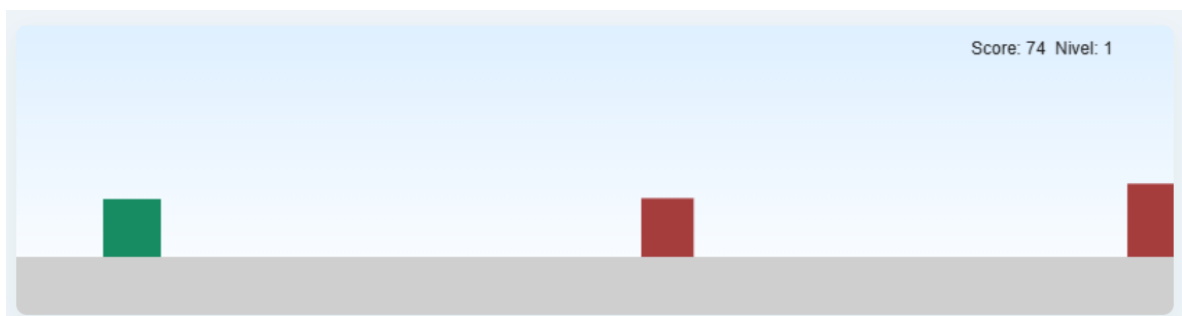
```
}
```

## Capturas de pantalla de funcionamiento

captura Landing Page



captura Juego en ejecución



captura Game Over con opción de guardar score



captura Leaderboard con puntuaciones

#### Leaderboard

#	Nombre	Score	Nivel
1	yogano	605	4
2	aiuuuuuu	300	2
3	dfghjkl	143	1
4	a	112	1

## Links

Repositorio GitHub: <https://github.com/urielZTN/Juego-tipo-Dinosaurio-de-Google->

## Preguntas de reflexión

### ¿Qué ventajas y limitaciones tiene usar Canvas para este juego?

Ventajas: permite gráficos fluidos y control total del renderizado. Limitaciones: mayor dificultad de accesibilidad y requiere más código que elementos HTML convencionales.

### ¿Cómo se diseñó la progresión de los niveles y la dificultad?

Se aumentó gradualmente la velocidad del juego y la frecuencia de aparición de obstáculos conforme avanza la partida.

### ¿Qué método de detección de colisiones se usó y por qué?

Se utilizó detección de colisiones basada en bounding boxes (rectángulos), por ser eficiente y suficiente para sprites simples.

### ¿Qué mejoras harías en la accesibilidad del juego?

Agregar controles alternativos (ej. teclas configurables), soporte para narración y modo de alto contraste.

### ¿Qué validaciones aplicaste en el Leaderboard?

Se validó que el nombre del jugador no esté vacío y que el puntaje sea un número válido antes de guardarlo.

## ¿Cómo asegurarías la integridad de los scores guardados en el servidor?

Controlando el acceso a la API, sanitizando la entrada y validando los datos antes de persistirlos en JSON.

## ¿Qué diferencias notaste entre guardar datos en localStorage y en la API?

localStorage solo guarda en el navegador del usuario, mientras que la API permite persistencia global y consulta desde cualquier cliente.

## ¿Cómo modularizaste el código del juego para que fuera más fácil de mantener?

Separando lógica del juego (game.js), comunicación con la API (api.js) y estilos (game.css).

## ¿Qué pruebas hiciste para validar que las colisiones y puntuaciones funcionaran correctamente?

Se probaron múltiples partidas, forzando choques en distintos momentos y validando que los puntajes y niveles aumentaran de forma correcta.

## ¿Qué mejora de alto impacto implementarías si tuvieras más tiempo?

Agregar enemigos con diferentes patrones de movimiento, power-ups y un sistema de usuarios con autenticación.