



Universidad Autónoma de Yucatán

Tarea 1.1

Investigación de una Librería Gráfica para Web

Graficas por computadoras

PlayCanvas

Alumnos:

Azareel Gidalti Balan Alonzo
Santiago Efraín Itzincab Poot
Héctor Jesús Vela Acosta

Profesor:

Dr. Francisco Alejandro Madera
Ramírez



PLAYCANVAS

PlayCanvas es un motor WebGL diseñado específicamente para crear juegos 3D de alto rendimiento y contenido interactivo. Destaca porque no es solo una biblioteca, sino un motor de juego completo con un entorno de desarrollo integrado (IDE) alojado en la nube. Esto hace que PlayCanvas sea ideal para proyectos colaborativos en los que varios desarrolladores pueden trabajar juntos en tiempo real en un solo proyecto.

CARACTERÍSTICAS CLAVES



- **EDITOR BASADO EN LA NUBE:**

PlayCanvas proporciona un entorno de desarrollo basado en la nube, lo que le permite crear, editar y obtener una vista previa de sus juegos 3D directamente en su navegador sin necesidad de instalar nada.



- **ALTO RENDIMIENTO:**

PlayCanvas está optimizado para el rendimiento, lo que lo hace adecuado para aplicaciones exigentes como juegos 3D o experiencias interactivas de alta calidad.



- **MOTOR DE FÍSICA:**

PlayCanvas tiene soporte físico integrado, lo que permite colisiones y movimientos de objetos realistas.

- **SOPORTE MULTIJUGADOR:**

El motor está diseñado para juegos multijugador en tiempo real, ofreciendo funciones de red listas para usar.

- **GESTIÓN DE ACTIVOS INTEGRADA:**

el editor basado en la nube facilita la gestión de activos como modelos 3D, texturas y sonidos, todo en un solo lugar.



OTRAS CARACTERÍSTICAS



- **ECOSISTEMA DE COMPONENTES (ENTITY-COMPONENT-SYSTEM, ECS):**

Manejo eficiente de entidades con componentes como cámara, modelo, luz, sonido, física, scripts, partículas, etc.



- **AUDIO POSICIONABLE:**

Sonido 3D gracias a Web Audio API



- **CÓDIGO ABIERTO Y FÁCIL DE USAR:**

Se usa JavaScript (también compatible con TypeScript) y no requiere compilación; esto permite iterar rápido y es fácilmente integrable



- **RENDERIZADO 3D AVANZADO CON WEBGL Y WEBGPU:**

Soporta ambos y cambia automáticamente al que mejor funcione en el navegador

- **ANIMACIÓN Y FÍSICA INTEGRADAS:**

Animaciones basadas en estado y física con rigid-body mediante la integración de ammo.js

- **SISTEMA DE CARGA DE RECUSROS:**

Asíncrono, con soporte para glTF, compresión (Draco, Basis), y variantes según dispositivo

Cuándo usar PlayCanvas

PlayCanvas es el más adecuado para crear juegos 3D basados en navegador o experiencias interactivas que requieren colaboración en tiempo real. Su IDE basado en la nube facilita el trabajo con un equipo de desarrolladores y diseñadores. Si está creando un juego multijugador o una aplicación web altamente interactiva, el conjunto de funciones de PlayCanvas y las herramientas de red integradas le ahorrarán tiempo y esfuerzo.

CAPACIDADES ADICIONALES DESTACADAS

MOTOR GRÁFICO MODERNO:

- PBR (Physically Based Rendering) con iluminación realista: incluye workflows metal/roughness y specular/glossiness, materiales avanzados como anisotropía, clearcoat, etc.
- Iluminación eficiente: sistema clustered para cientos de luces dinámicas, luz ambiental HDR, lightmaps en runtime, distintos tipos de luces (area, direccional, puntual).
- HDR, blooms, SSAO, TAA, efectos post-processing como DoF, color grading, viñeta.
- Pipeline moderno: múltiples pasadas de render, MRT, instanciación por hardware, batching, shaders personalizables (GLSL, WGLSL), splatting, partículas aceleradas por GPU.

SCRIPTING ROBUSTO

- Usa scripts basados en módulos modernos (ESM/.mjs) o el sistema clásico en .js. Pueden coexistir.
- Buen acceso al API del motor: clases clave como AppBase, Entity, Component, Vec3, Color, etc. Existen patrones comunes para acceder a entidades, assets, eventos, etc.



EDITOR COLABORATIVO EN LÍNEA

- Edición visual en navegador, con render en tiempo real, glTF nativo y compresión Draco, control de versiones, chat para equipos, y publicación con un clic.

COMPONENTES DECLARATIVOS(WEB COMPONENTS)

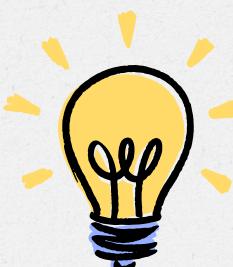
- PlayCanvas ofrece elementos HTML personalizados (<pc-app>, <pc-scene>, <pc-entity>, etc.) que permiten crear escenas 3D con solo HTML sin código JS
- Son personalizables, optimizados para rendimiento, responsivos y compatibles con WebGL/WebGPU

API REFERENCE Y EXTENSIBILIDAD

- Documentación completa del Engine, Editor, Web Components, React, UI, etc.

PROYECTOS REALES DE EJEMPLO

- La plataforma ha sido usada en juegos y aplicaciones web avanzadas en tiempo real, como multijugador, configuradores de vehículos (con PBR), metaversos y más.



DATOS ADICIONALES RELEVANTES



- Empresas como Disney, King, Mozilla, IGT, ARM, Samsung han usado la plataforma para proyectos reales

1

- PlayCanvas se abrió como open source en 2014 y fue adquirido por Snap Inc. tras eso

2



3

- Hay repositorios oficiales para el motor, editor, recursos, extensiones (como Spine, tweening, etc.)

4

- Puedes exportar proyectos vía URL o empaquetarlos para plataformas como Android (CocoonJS) o Steam (Electron)



DOCUMENTACIÓN

El código presentado corresponde a un juego 2D sencillo inspirado en Flappy Bird, desarrollado con la librería PlayCanvas. La estructura comienza en un documento HTML que incluye las etiquetas básicas de configuración. En la cabecera se define la codificación de caracteres UTF-8 para permitir el uso de acentos y símbolos, se establece el título de la pestaña del navegador y se importa la librería principal de PlayCanvas. Además, mediante hojas de estilo en línea, se ajusta el lienzo (canvas) para ocupar el 100% de la pantalla y eliminar márgenes o barras de desplazamiento, garantizando que el juego se ejecute a pantalla completa. Posteriormente, se inicializa la aplicación de PlayCanvas. Para ello, se obtiene el elemento canvas del documento y se crea una instancia de pc.Application, la cual representa la base del juego. A esta aplicación se le configura el modo de llenado de ventana y la resolución automática, asegurando que se adapte correctamente al tamaño de la pantalla del dispositivo donde se ejecute.

```
1  <html>
2  <head>
3  |  <meta charset="utf-8">
4  |  <title>TapTapBoom</title>
5  |  <script src="https://code.playcanvas.com/playcanvas-stable.min.js"></script>
6  |  <style>
7  |  |  body { margin: 0; overflow: hidden; }
8  |  |  canvas { width: 100%; height: 100%; display: block; }
9  |  </style>
10 </head>
11 <body>
12 <canvas id="application"></canvas>
13 <script>
14 |  const canvas = document.getElementById("application");
15 |  const app = new pc.Application(canvas, {});
16 |  app.setCanvasFillMode(pc.FILLMODE_FILL_WINDOW);
17 |  app.setCanvasResolution(pc.RESOLUTION_AUTO);
```

Una vez configurada la aplicación, se procede a crear la cámara del juego. Esta se define como una entidad con un componente de cámara en proyección ortográfica, lo cual permite trabajar en un entorno bidimensional sin perspectiva. Se establece un color de fondo azul claro que simula el cielo y se posiciona la cámara en el eje Z, para visualizar adecuadamente la escena.

El jugador se representa mediante un cubo rojo. Para ello se genera una entidad con un modelo de tipo caja, a la cual se le asigna un material de color rojo brillante, aprovechando la propiedad "emissive" que hace que el objeto resalte independientemente de la luz de la escena. El cubo se escala a un tamaño más pequeño y se ubica hacia el lado izquierdo de la pantalla, listo para que el usuario lo controle.

```
19 // Cámara ortográfica (2D)
20 const camera = new pc.Entity("Camera");
21 camera.addComponent("camera", {
22   projection: pc.PROJECTION_ORTHOGRAPHIC,
23   orthoHeight: 5,
24   clearColor: new pc.color(0.7, 0.7, 1)
25 });
26 camera.setLocalPosition(0, 0, 10);
27 app.root.addChild(camera);
28
29 // Jugador (cubo rojo)
30 const player = new pc.Entity("Player");
31 player.addComponent("model", { type: "box" });
32 const playerMat = new pc.StandardMaterial();
33 playerMat.emissive = new pc.Color(1, 0, 0); // Rojo
34 playerMat.emissiveIntensity = 1;
35 playerMat.update();
36 player.model.material = playerMat;
37 player.setScale(0.5, 0.5, 0.5);
38 player.setLocalPosition(-3, 0, 0);
39 app.root.addChild(player);
```

```

41 // Obstáculos
42 const obstacles = [];
43 const greenMat = new pc.StandardMaterial();
44 greenMat.emissive = new pc.Color(0.1, 0.4, 0.1); // Verde
45 greenMat.emissiveIntensity = 1;
46 greenMat.update();
47
48 function createObstacle(xPos) {
49     const gap = 2; // espacio en el medio
50     const topHeight = Math.random() * 5 + 1;
51     const bottomHeight = 8 - topHeight - gap;
52
53     // Tubo superior
54     const top = new pc.Entity("TopPipe");
55     top.addComponent("model", { type: "box" });
56     top.model.material = greenMat;
57     top.setLocalScale(1, topHeight, 0.5);
58     top.setLocalPosition(xPos, 5 - topHeight/2, 0);
59     app.root.addChild(top);
60
61     // Tubo inferior
62     const bottom = new pc.Entity("BottomPipe");
63     bottom.addComponent("model", { type: "box" });
64     bottom.model.material = greenMat;
65     bottom.setLocalScale(1, bottomHeight, 0.5);
66     bottom.setLocalPosition(xPos, -5 + bottomHeight/2, 0);
67     app.root.addChild(bottom);
68
69     obstacles.push({ top, bottom });
70 }

```

El juego cuenta con varias variables de control. Entre ellas, se define la velocidad y la gravedad que afectan al movimiento vertical del jugador, así como la fuerza de salto que se activa cuando el usuario presiona la tecla de espacio. También se incluyen variables para determinar el estado de la partida (si continúa o si ha terminado), el puntaje acumulado y los temporizadores que controlan tanto la generación de obstáculos como el incremento progresivo de la dificultad con el paso del tiempo.

El control del jugador se implementa mediante un evento de teclado. Cuando el usuario presiona la tecla de espacio, la velocidad vertical del cubo se ajusta para simular un salto hacia arriba, siempre y cuando el juego no haya terminado. Esta interacción es fundamental para dar vida a la mecánica principal del juego.

```

89 // Loop de actualización
90 app.on("update", function (dt) {
91     if (gameOver) return;
92
93     // Movimiento jugador (gravedad + salto)
94     velocity += gravity * dt;
95     let pos = player.getLocalPosition();
96     pos.y += velocity * dt;
97     player.setLocalPosition(pos);
98
99     // Generar obstáculos
100    obstacleTimer += dt;
101    difficultyTimer += dt;
102    if (obstacleTimer > obstacleInterval) {
103        createObstacle(8);
104        obstacleTimer = 0;
105    }
106
107    // Aumentar dificultad cada 10 segundos
108    if (difficultyTimer > 5 && obstacleInterval > 0.5) {
109        obstacleInterval -= 0.2; // reduce el intervalo
110        difficultyTimer = 0;
111        console.log("Dificultad aumentada. Nuevo intervalo:", obstacleInterval.toFixed(2));
112    }

```

En cuanto a los obstáculos, estos se construyen a partir de tubos verdes que se generan de manera dinámica. Se utiliza una función llamada `createObstacle` que recibe una posición en el eje X y crea dos cubos, uno superior y otro inferior, dejando un espacio libre entre ellos que el jugador debe atravesar. Las alturas de los tubos se asignan de manera aleatoria para aumentar la dificultad y la variedad de cada intento. Cada obstáculo creado se almacena en un arreglo para poder manipularlo más adelante.

```

72 // Variables de juego
73 let velocity = 0;
74 let gravity = -10;
75 let jumpForce = 5;
76 let gameOver = false;
77 let score = 0;
78 let obstacleTimer = 0;
79 let obstacleInterval = 2; // segundos entre obstáculos
80 let difficultyTimer = 0;
81
82 // Input
83 window.addEventListener("keydown", (e) => {
84     if (e.code === "Space" && !gameOver) {
85         velocity = jumpForce;
86     }
87 });

```

La lógica central se encuentra en el bucle de actualización, el cual se ejecuta de manera continua en cada fotograma gracias a la función `app.on("update")`. Dentro de este bucle, primero se actualiza la posición del jugador aplicando la gravedad y la fuerza de salto. Luego se controlan los temporizadores para generar nuevos obstáculos cada cierto intervalo y, además, se ajusta automáticamente el nivel de dificultad reduciendo el tiempo entre la aparición de los tubos conforme pasan los segundos.

Cada obstáculo se mueve hacia la izquierda para simular que el jugador avanza. Dentro de este proceso, también se detectan las colisiones. Si el cubo del jugador choca con alguno de los tubos, se muestra un mensaje de "Game Over" junto con el puntaje obtenido y la partida se da por terminada. Asimismo, cuando el jugador logra pasar un obstáculo sin colisionar, el puntaje aumenta automáticamente. Los obstáculos que ya han salido de la pantalla son eliminados de la escena para optimizar el rendimiento.

```

114 // Mover obstáculos
115 for (let i = obstacles.length - 1; i >= 0; i--) {
116     let obs = obstacles[i];
117     obs.top.translateLocal(-3 * dt, 0, 0);
118     obs.bottom.translateLocal(-3 * dt, 0, 0);
119
120     // Colisión simple
121     if (
122         Math.abs(obs.top.getLocalPosition().x - pos.x) < 0.5 &&
123         (pos.y > obs.top.getLocalPosition().y - obs.top.getScale().y / 2 ||
124          pos.y < obs.bottom.getLocalPosition().y + obs.bottom.getScale().y / 2)
125     ) {
126         alert("💀 Game Over! Puntuación: " + score);
127         gameOver = true;
128     }
129
130     // Aumentar puntaje
131     if (obs.top.getLocalPosition().x < pos.x && !obs.scored) {
132         score++;
133         console.log("Puntaje:", score);
134         obs.scored = true;
135     }
136
137     // Eliminar obstáculos viejos
138     if (obs.top.getLocalPosition().x < -10) {
139         app.root.removeChild(obs.top);
140         app.root.removeChild(obs.bottom);
141         obstacles.splice(i, 1);
142     }
143 }
```

salido de la pantalla son eliminados de la escena para optimizar el rendimiento.

Finalmente, se incluyen límites en la parte superior e inferior del área de juego. Si el cubo se desplaza demasiado hacia arriba o hacia abajo, el juego también finaliza mostrando la puntuación alcanzada. Para concluir la configuración, se ejecuta la función `app.start()`, que inicia oficialmente la aplicación y mantiene activo el ciclo del juego.

En resumen, este código implementa un juego estilo Flappy Bird en el que un cubo rojo controlado por el jugador debe atravesar huecos entre tubos verdes generados aleatoriamente. El sistema de gravedad, salto, obstáculos dinámicos, detección de colisiones, puntaje y dificultad progresiva conforman los elementos esenciales que hacen posible esta experiencia interactiva dentro de un navegador web.

```

145 // Límite del mundo
146 if (pos.y < -5 || pos.y > 5) {
147     alert("💀 Game Over! Puntuación: " + score);
148     gameOver = true;
149 }
150 );
151 app.start();
152 </script>
```

Programa: <https://juniorstorm.github.io/Gr-ficas-por-computadora/>

REFERENCIAS



- <https://blog.pixelfreestudio.com/top-webgl-libraries-for-building-3d-interactive-websites/>
- <https://dev.playcanvas.com/>
- <https://developer.playcanvas.com/user-manual/graphics/>
- <https://en.wikipedia.org/wiki/PlayCanvas>
- <https://developer.playcanvas.com/user-manual/web-components/>
- <https://github.com/JuniorStorm/Gr-ficas-por-computadora>
- <https://juniorstorm.github.io/Gr-ficas-por-computadora/>