CREACIÓN JUEGO HTML5

1 CONCEPTOS BÁSICOS

- Canvas (Lienzo): Mapa de bits que se usa para renderizar gráficos (líneas, Círculos...) <canvas>
  - BeingPath(): Grabar figura nueva
  - moveTo(), lineTo(), arc() Crear forma.
  - ClosePath() Opcional, cerrar
  - var context = canvas.getContext('2d')
  - DrawImage ... muchos métodos
- Elemento Audio <audio src="" controls="" type="">
  - Se puede cargar dinámicamente en un objeto de JS
- Elemento Imagen
  - Como todo lo anterior, se puede hacer un precargador
    onload → Barra de progreso
  - DrawImage + Sprite y posiciones → 24 (Menos solicitudes HTTP, Mejor compresión, tiempos de carga más rápidos)
- Animar: Una función de dibujo que se llama X veces
  - Métodos setInterval() y setTimeout() llaman a drawingLoop() a intervalos regulares
  - Usar API → RequestAnimationFrame (Mejor que lo anterior)

CREACIÓN JUEGO HTML 5

1 CONCEPTOS BÁSICOS

- Canvas (Lienzo): Mapa de Bits que se usa para renderizar gráficos (Líneas, Círculos...) <canvas>
  - BeingPath(): Grabar figura nueva
  - moveTo(), lineTo(), arc() Crear forma.
  - ClosePath() Opcional, cerrar

```
var context = canvas.getContext('2d')
```

  - DrawImage ... muchos métodos
- Elemento Audio <audio src="" controls="" type="">
  - Se puede cargar dinámicamente en un objeto de JS
- Elemento Imagen
  - Como todo lo anterior, se puede hacer un precargador
    onload → Barra de progreso
  - DrawImage + Sprite y posiciones → 24 (Menos solicitudes HTTP, Mejor compresión, tiempos de carga más rápidos)
- Animar: Una función de dibujo que se llama varias veces
  - Métodos setInterval() y setTimeout() llaman a drawingLoop() a intervalos regulares
  - Usar API → RequestAnimationFrame (Mejor que lo anterior)

2. CREACIÓN DEL MUNDO BÁSICO DEL JUEGO.

- Pantalla de Bienvenida. (Index.html)
  - GameContainer, ScoreScreen, GameStartScreen, LevelSelectScreen, LoadingScreen, EndingScreen
  - Style.css → Capas juego en posición absoluta.
  - Game.js → Mostrar
    - game.init() Precarga. → var game
- Selección Nivel → Objeto Levels var levels (game.js)
  - Array con info de cada nivel → Data.
  - Genera Dinámicamente Botones para cada nivel
  - game.showLevelScreen() → Ocultar ppal y mostrar esta.
- Carga de Imágenes / Sonidos
  - GIF para visualizar la carga de imágenes
    #loadingscreen #loadingmessage.
  - En JS var loader → loadImage / loadSound
- Cargar Niveles
  - Definición load() en objeto levels
    - currentLevel: Almacenar los datos del nivel que queremos cargar
- Animación Del Juego
  - Se llama muchas veces por segundo a requestAnimationFrame
    - Al comienzo del .js
  - Vendors → Diferentes prefijos según el navegador
  - Puntuación → CSS

- Entradas de Ratón. (Objeto mouse en game.js
  - mouseup/down ... llama al mousehandler
  - En el handler usar getset() para calcular las coordenadas x e y respecto a la esquina superior izq.

- Definición de los estados de juego.
  - Se almacena el estado actual del juego en game.mode.
    - Intro, load-next-hero, wait-for-firing, firing y fired
    - pauto(): Desplaza pantalla suavemente → Intro.

3. FUNDAMENTOS DEL MOTOR DE FÍSICAS.

- Motor Box2D (Fundamentos)
  - World: Objeto Ppal. contiene todos los objetos del juego
  - Body: Cuerpo rígido.
  - Shape: Formas bidimensionales
  - Fixture: Detección de Colisiones
- Box2d.html → en el body inicializar, antes script ligado al js.
- Corazón Box2D.Dynamics.b2World → JS
  - Gravity: Como vector, 9,8 hacia abajo
  - allowSleep: Objetos en reposo o no (boolean)
  - Scale: Metros a pixeles (30px ≡ 1m)

4. AÑADIENDO PRIMER BODY (PISO)

- Declarar definición body → b2BodyDef (Estático)
- Fixture: Unir forma a body (Densidad, fricción, restitución)
  - Polígonos, y Círculos
- Create Floor
  - Densidad: Se utiliza para calcular el peso del cuerpo
  - Coeficiente de Fricción
  - Restitución: Cuerpo Rebote

- Dibujando el Mundo.
  - DrawDebugData: Dibuja una representación simple de los cuerpos. (Dibujos de depuración, b2DebugDraw())
  - Llamado en la función init();

- Animando el Mundo (A través de la función animate.js)
  - TimeStep: Tiempo que se quiere simular en Box2D

- Más elementos B2D
  - RectangularBody → b2PolygonShape (SetAsBox)
    → dynamicBody (Se verá afectado por colisiones)
  - CircleBody → Constructor sobre el radio del círculo.
  - PolygonBody → Crear un polígono definiendo las coordenadas para cada punto. (Polígonos convexos)
    Array de Objetos

  - Cuerpos Complejos

- Crear Cuerpos Complejos (CreateComplexPolygonBody()
  - Juntar multiplex fixtures (Accesorios)
- Conectar Cuerpos con Artefactos (Articulaciones, restringir los cuerpos al mundo)
  - CreateRevoluteJoint();
- Seguimiento Colisiones y Daños (Dar vida/Salud)
  - SetUserData() y GetUserData():
    name: life, ⟶ 2 parámetros (B3)
- Contact Listeners: Definir manejadores de Eventos
  - BeginContact: Dos fixtures entran en Contacto.
  - EndContact: Cesan contacto
  - PostSolve(): Inspeccionar un contacto.
  - PreSolve(): Igual antes de Solucionarlo.
  - Comprobar Vida con DestroyBody() (140)

INTEGRACIÓN CON EL MOTOR DE LA FÍSICA.

- Definir Entidades (153) Var Entidades ⟶ array
  - Block, Ground, Hero, Villain (164)
- Añadir entidades a los niveles (Var levels) ⟶ data
  - Un bucle for en load function itera por todas las entidades (entities.create)

- Configurar el dibujo de Depuración Box2D
  - Nueva canvas al final del body
  - Draw.DebugData();

- Dibujar entidades (180)
- Animar el mundo
  - Método Step() con intervalo de tiempo
  - requestAnimationFrame puede variar la frecuencia con la que se llama a animate();

- Estados → Cargar heroe (load-next-hero)
  - Almacena los héroes en array game.heroes y villanos en otro array
  - Disparar el Héroe

Diapo 200