JUEGO PHASER

CREADO POR

FRANK RUBÉN ANTHONY

INDICE

Explicacion juego Phaser

INTRODUCCION

Phaser es un framework de desarrollo de videojuegos en 2D, basado en JavaScript. Para importar la librería y poder utilizarla es necesario importar la a nuestro archivo html de esta manera.

Generando el archivo game.js podremos empezar a trabajar en nuestro juego, el html simplente sera la carcasa donde nosotros podremos decorar lo que rodea a la pantalla de nuestro juego

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>GAME</title>
  <script src="./phaser.min.js"></script>
  <script type="module" src="./game.js"></script>
  <style>
    body {
      display: grid;
      align-items: center;
      margin-left: 30%;
      height: 100vh;
      background: white;
      padding-top: 5px;
    }
  </style>
</head>
<body>


  <!--Donde vamos a ver nuestro juego prueba 1-->

  <div id="game"></div>

</html>
```

Generando el archivo game.js podremos empezar a trabajar en nuestro juego, el html simplente sera la carcasa donde nosotros podremos decorar lo que rodea a la pantalla de nuestro juego(es la imagen es la parte blanca de la pagina, el juego

evidentemente se ejecuta en el cuadrado del centro es decir las lineas que aparecen en game.js)

Dentro de nuestro archivo Game.js

```
const config = {
    type: Phaser.AUTO,
    width: 500,
    height: 500,
    parent: 'game',
    physics: {
      default: 'arcade',
      arcade: {
        gravity: { y: 400 },
        debug: false
      }
    },
```

```
new Phaser.Game(config);
```

Hay que imponer unas condiciones del juego y del mundo. En este caso el ancho y el alto donde vamos a ver el juego, el parent es el id que le pusimos al div en el html por lo que es donde vamos a meter el juego para que se vea y dentro de físicas en default, es el modo de juego que queremos. Phaser nos da muchas opciones para escoger, pero en ese caso escogeremos el modo arcade ya que es el que más nos conviene. Dentro de esta opción podremos escoger la gravedad del mundo por lo que ya empezaremos a configurar las físicas de nuestro juego desde un principio.

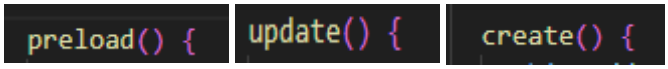
Luego habrá que inicializar el juego pero antes hay que tener claro una cosa muy importante las escenas que va a tener nuestro juego en este caso tendrá la del juego y la de cuando morimos que aparece nuestra puntuación respectivamente. Aquí podríamos hacer un menú de carga, créditos o más pantallas a más niveles es decir un sinfín de opciones.

```
  },
  scene: [MainScene, GameOverScene]
};
```

```
class MainScene extends Phaser.Scene {
  constructor() {
    super({ key: 'MainScene' });
  }
```

Para la primera escena la del juego respectivamente hay que tener claro algunos conceptos que te ayudaran a que programar sea mucho más fácil. Por ejemplo, Phaser.Secene después de extends hace que Phaser reconozca alguna de sus funciones que ya tenemos creadas en nuestro fichero que descargamos e importamos anteriormente. Estas funciones son clave para el desarrollo de nuestro juego que usaremos en nuestras escenas ya creadas con anterioridad y son

Créate, preload y update



Preload dentro de la escena MainScene

```
preload() {
  this.load.image('background', './oak_woods_v1.0/background/background_layer_1.png');
  this.load.spritesheet('suelo', './LEGACY/Assets/Hive.png', {
    frameWidth: 80,
    frameHeight: 64
  });
  this.load.spritesheet('hero', './oak_woods_v1.0/character/char_blue.png', {
    frameWidth: 56,
    frameHeight: 56
  });
  this.load.spritesheet('monster', './2D Pixel Dungeon Asset Pack/character and tileset/Dungeon_Character.png', {
    frameWidth: 16,
    frameHeight: 16
  });
  this.load.spritesheet('monster2', 'Monster_Creatures_Fantasy(Version 1.3)\\Skeleton\\Attack3.png', {
    frameWidth: 16,
    frameHeight: 16
  });
}
```

Si nos fijamos esta función se utiliza para cargar todos los recursos que vamos a utilizar en nuestro juego es decir imágenes, sonidos, sprites fondos del juego ...

Si nos fiamos cada imagen tiene un framewidth y un frameheight esto se utiliza para medir el Sprite y luego poder hacer animaciones con ellas que se explicara mas adelante. Por ejemplo:

```
class MainScene extends Phaser.Scene {
  constructor() {
    super({ key: 'MainScene' });
  }

  preload() {
    this.load.image('background', './oak_woods_v1.0
    this.load.spritesheet('suelo', './LEGACY/Assets
      frameWidth: 80,
      frameHeight: 64
    });
    this.load.spritesheet('hero', './oak_woods_v1.0/character/char_blue.png', {
      frameWidth: 56,
      frameHeight: 56
```

Reveal in Side Bar
Open Containing Folder
448x392
22.05 KB

En este caso es el personaje principal vemos (gracias a la extensión de vscode de image preview) que la imagen es de 448x392 esto significa que como hay 8x7

imágenes en cada filaxcolumna respectivamente cada una es de 56x64, así cortaremos nuestros sprites para que podamos utilizar el que queramos y la secuencia que queramos a nuestro antojo haciéndonos mucho mas fácil el hacer nuestro juego.

CREATE EN MY SCENE:

```
create() {
  this.background = this.add.image(0, 0, 'background')
    .setOrigin(0, 0)
    .setDisplaySize(this.sys.game.config.width, this.sys.game.config.height);
  this.background.setScrollFactor(0);

  this.floor = this.physics.add.staticGroup();
  for (let i = 0; i < 30; i++) {
    if (i === 10 || i === 11) continue;
    this.floor.create(i * 80, 500, 'suelo')
      .setOrigin(0, 1)
      .setScale(1)
      .refreshBody();
  }

  this.hero = this.physics.add.sprite(100, 350, 'hero')
    .setOrigin(0, 1)
    .setScale(1.5);
  this.hero.isDead = false;
  this.hero.isAttacking = false;
  this.hero.body.setSize(30, 50);
  this.hero.body.setOffset(10, 6);
  this.hero.lives = 3;
  this.hero.score = 0;
  this.physics.add.collider(this.hero, this.floor);

  this.monster = this.physics.add.sprite(300, 350, 'monster')
    .setScale(2)
    .setBounce(0)
    .setCollideWorldBounds(true)
    .setVelocityX(50);
  this.monster.isDead = false;
  this.monster.body.setSize(16, 16);
  this.monster.body.setOffset(0, 0);
  this.physics.add.collider(this.monster, this.floor);
```

En el créate ya utilizaremos todos los recursos que hemos cargado para utilizarlos en juego, aquí podemos también crear las colisiones y animaciones (que utilizaremos mas adelante).

Por ejemplo:

```
this.floor = this.physics.add.staticGroup();
for (let i = 0; i < 30; i++) {
  if (i === 10 || i === 11) continue;
  this.floor.create(i * 80, 500, 'suelo')
    .setOrigin(0, 1)
    .setScale(1)
    .refreshBody();
```

Aquí vemos como creamos el suelo de nuestro mundo, vemos que this siempre se le pone a todos nuestros id en este caso floor ya que this se refiere a este juego. después se le añade al grupo de objetos estáticos (que no se mueven, como el suelo) que pueden colisionar con otros objetos físicos.

Para que no tengamos que crear suelo a suelo utilizamos la función for y hacemos que vaya de la 1 a la 30 pero antes que cuando llegue a la 10 haya un hueco hasta la 11 y que continue, las siguientes líneas es indicando el tamaño y el origen de donde lo queremos y también lo grande que es.

```
this.hero = this.physics.add.sprite(100, 350, 'hero')
  .setOrigin(0, 1)
  .setScale(1.5);
this.hero.isDead = false;
this.hero.isAttacking = false;
this.hero.body.setSize(30, 50);
this.hero.body.setOffset(10, 6);
this.hero.lives = 3;
this.hero.score = 0;
this.physics.add.collider(this.hero, this.floor);
```

En este caso vamos a crear al héroe y vemos un sprite con físicas en la posición (100, 350) usando la imagen llamada 'hero'.

.setOrigin(0, 1) la referencia del sprite es en la esquina inferior izquierda.

.setScale(1.5) lo agranda el sprite un 50%.

this.hero.isDead = false;

this.hero.isAttacking = false;

Atributos personalizados para controlar el estado del personaje: si está muerto o atacando.

this.hero.body.setSize(30, 50);

this.hero.body.setOffset(10, 6);

Ajusta el tamaño del cuerpo físico del héroe (el área de colisión).

.setOffset(10, 6) mueve ese cuerpo un poco respecto al sprite, para que encaje bien.

this.hero.lives = 3;

this.hero.score = 0;

Inicializa vidas y puntuación del jugador.

this.physics.add.collider(this.hero, this.floor);

Añade una colisión física entre el héroe y el suelo, para que no lo atraviese y pueda caminar sobre él.

```
this.anims.create({ key: 'hero-walk', frames: this.anims.generateFrameNumbers('hero', { start: 17, end: 23 }), frameRate: 18, repeat: -1 });
this.anims.create({ key: 'hero-idle', frames: [{ key: 'hero', frame: 0 }] });
this.anims.create({ key: 'hero-jump', frames: this.anims.generateFrameNumbers('hero', { start: 24, end: 37 }), frameRate: 11, repeat: 0 });
this.anims.create({ key: 'hero-dead', frames: this.anims.generateFrameNumbers('hero', { start: 39, end: 48 }), frameRate: 11, repeat: 0 });
this.anims.create({ key: 'hero-attack', frames: this.anims.generateFrameNumbers('hero', { start: 8, end: 12 }), frameRate: 11, repeat: 0 });
```

Aquí podemos ver las animaciones creadas que luego utilizaremos en el update

Por ejemplo la animación de saltar, la creamos en con la id de hero-jum y luego le decimos que queremos un rango de frmas de la foto que cargamos antes y le decimos que sea del 24 al 37 y que la rapidez del cambio de frame sea 11

```
this.physics.add.overlap(this.hero, this.monster, () => {
  if (this.hero.isAttacking && !this.monster.isDead) {
    this.monster.isDead = true;
    this.monster.disableBody(true, true);
    this.hero.score += 100;
    this.updateHUD();
  } else if (!this.hero.isDead && !this.monster.isDead) {
    this.hero.lives--;
    this.updateHUD();
```

this.physics.add.overlap(this.hero, this.monster, () => {

add.overlap detecta cuando los dos personajes se tocan, pero no los bloquea físicamente como lo haría un collider.

Si el héroe está atacando:

if (this.hero.isAttacking && !this.monster.isDead) {

```
  this.monster.isDead = true;
  this.monster.disableBody(true, true);
  this.hero.score += 100;
  this.updateHUD();
}
```

Si el héroe está atacando y el monstruo no está muerto:

Se marca al monstruo como muerto.

disableBody(true, true) lo desactiva y oculta.

El jugador gana 100 puntos.

Se actualiza el HUD (vidas y puntuación en pantalla).

Si el héroe no ataca y choca con el monstruo:

```
} else if (!this.hero.isDead && !this.monster.isDead) {
  this.hero.lives--;
  this.updateHUD();
}
```

Si el héroe está vivo y el monstruo también, pero no se está atacando:

El héroe pierde una vida.

Se actualiza el HUD.

UPDATE EN MYSCNE

```
update() {
  let isOnGround = this.hero.body.blocked.down;
  if (this.hero.isDead) return;

  if (Phaser.Input.Keyboard.JustDown(this.attackKey) && !this.hero.isAttacking) {
    this.hero.isAttacking = true;
    this.hero.setVelocityX(0);
    this.hero.anims.play('hero-attack', true);
    this.time.delayedCall(300, () => {
      this.hero.isAttacking = false;
    });
    return;
  }

  if (!this.hero.isAttacking) {
    if (this.keys.left.isDown) {
      this.hero.setVelocityX(-160);
      if (isOnGround) this.hero.anims.play('hero-walk', true);
      this.hero.setFlipX(true);
    } else if (this.keys.right.isDown) {
      this.hero.setVelocityX(160);
      if (isOnGround) this.hero.anims.play('hero-walk', true);
      this.hero.setFlipX(false);
    } else {
      this.hero.setVelocityX(0);
      if (isOnGround) this.hero.anims.play('hero-idle', true);
    }

    if (this.keys.up.isDown && isOnGround) {
      this.hero.setVelocityY(-300);
      this.hero.anims.play('hero-jump', true);
    }
```

La función update() en Phaser se ejecuta en cada frame del juego, lo que permite controlar el comportamiento continuo de los elementos del juego, como el movimiento del personaje, las colisiones y las acciones del jugador. En este caso, la función revisa si el personaje está en el suelo (isOnGround), si está muerto o si está atacando, y responde al input del teclado para mover al personaje a la izquierda o derecha, hacerlo saltar o ejecutar un ataque. Esto asegura que el juego responda fluidamente a las acciones del jugador en tiempo real.

Las animaciones permiten que el personaje muestre diferentes acciones visuales, como caminar, saltar o atacar, usando secuencias de imágenes (frames) extraídas de un sprite. En este código, se usan animaciones como 'hero-walk', 'hero-idle', y 'hero-attack', que se activan según la acción que esté realizando el jugador. Esto mejora la experiencia visual del juego, haciendo que el personaje se vea más vivo y responda visualmente a cada acción que realiza.

Por ejemplo:

```
if (!this.hero.isAttacking) {
  if (this.keys.left.isDown) {
    this.hero.setVelocityX(-160);
    if (isOnGround) this.hero.anims.play('hero-walk', true);
    this.hero.setFlipX(true);
  } else if (this.keys.right.isDown) {
    this.hero.setVelocityX(160);
    if (isOnGround) this.hero.anims.play('hero-walk', true);
    this.hero.setFlipX(false);
  } else {
    this.hero.setVelocityX(0);
    if (isOnGround) this.hero.anims.play('hero-idle', true);
  }
```

gestiona el movimiento horizontal del héroe cuando no está atacando. Si se presiona la flecha izquierda, el héroe se mueve hacia la izquierda con velocidad negativa, se reproduce la animación de caminar si está en el suelo y se voltea el sprite para que mire en esa dirección. Si se presiona la flecha derecha, se mueve hacia la derecha con velocidad positiva, también se reproduce la animación de caminar y el sprite se asegura de no estar volteado. Si no se presiona ninguna de estas teclas, el héroe se queda quieto y, si está en el suelo, se reproduce la animación de estar parado (idle). Todo esto permite un control fluido del personaje mientras se evita que se mueva durante una animación de ataque.

ESCENA GAMEOVER

```
183
184  class GameOverScene extends Phaser.Scene {
185    constructor() {
186      super({ key: 'GameOverScene' });
187    }
188
189    init(data) {
190      this.finalScore = data.score || 0;
191    }
192
193    create() {
194      this.add.text(150, 180, 'GAME OVER', {
195        fontSize: '32px',
196        fill: '#ff0000',
197        fontFamily: 'Arial'
198      });
199
200      this.add.text(130, 230, `Puntuación final: ${this.finalScore}`, {
201        fontSize: '20px',
202        fill: '#ffffff'
203      });
204
205      this.add.text(100, 270, 'Presiona ESPACIO para reiniciar', {
206        fontSize: '16px',
207        fill: '#ffffff'
208      });
209
210      this.input.keyboard.once('keydown-SPACE', () => {
211        this.scene.start('MainScene');
212      });
213    }
214  }
215
```

La clase GameOverScene en Phaser muestra la pantalla de "Game Over" cuando el jugador pierde todas sus vidas. En su método init, recibe la puntuación final desde la escena principal, y en create, muestra mensajes en pantalla con el texto "GAME OVER", la puntuación lograda y una instrucción para reiniciar. Al presionar la tecla espacio, el juego vuelve a empezar desde la escena principal (MainScene), permitiendo al jugador intentarlo de nuevo.