Mover player ()

```
this.keys = this.input.keyboard.addKeys({
  left: Phaser.Input.Keyboard.KeyCodes.A,
  right: Phaser.Input.Keyboard.KeyCodes.D,
  space: Phaser.Input.Keyboard.KeyCodes.SPACE,
  use: Phaser.Input.Keyboard.KeyCodes.E,
});
```

```
    if (cursors.left.isDown) {
      direction = -1;
      this.moveLeft();
    } else if (cursors.right.isDown) {
      direction = 1;
      this.moveRight();
    } else {
      direction = 0;
      this.body.setVelocityX(0);
      this.sprite.play('idleM', true);
    }
  moveLeft() {
    this.body.setVelocityX(-this.speed * this.momentum);
    this.sprite.play('walkM', true);
    this.sprite.flipX = false;
  }

  moveRight() {
    this.body.setVelocityX(this.speed * this.momentum);
    this.sprite.play('walkM', true);
    this.sprite.flipX = true;
  }



  jump() {
    if (this.body.blocked.down || this.body.touching.down) {
      this.body.setVelocityY(-this.jumpForce);
      this.sprite.play('jumpM');
    }
  }
```

Construir fisicas en container

```
  constructor(scene, x, y, texture) {
    super(scene, x, y);
    this.scene = scene;
```

```
    this.lifes = 3;
    this.speed = 160;
    this.jumpForce = 250;
    this.item = null;



    this.momentum = 1;
    this.lastDirection = 0;

    this.golpeado = false;
    this.controlBlocked = false;
    this.muerto = false;

    // Sprite
    this.sprite = scene.add.sprite(0, 0, texture, 3);
    this.sprite.setOrigin(0.5, 0.5);
    this.add(this.sprite);

    scene.add.existing(this);
    scene.physics.world.enable(this);

    this.body.setSize(this.sprite.width, this.sprite.height);
    this.body.setOffset(-this.sprite.width / 2, -this.sprite.height /
2);
    this.body.setAllowGravity(true);
    this.animator();
  }
```

Invincibilidad

```
     this.scene.time.delayedCall(2000, () => {
        this.golpeado = false;
      });
```

animaciones

```

    this.scene.anims.create({
      key: 'DieM',
      frames: [{ key: 'player', frame: 6 }],
      frameRate: 10 ,
      repeat: -1
    });
    this.scene.anims.create({
```

```
      key: 'walkM',
      frames: this.scene.anims.generateFrameNumbers('player', { start:
1, end: 3 }),
      frameRate: 10,
      repeat: -1
    });
this.anims.play(key, true);

this.destroy()
```

add items

```
  addItem(type) {
    if (this.item) {
      this.item.destroy();
    }

    if (type === 0) {
      this.item = new Hammer(this.scene, -10, -5,'hammer',this);
  if (!this.scene.itemGroup) {
    this.scene.itemGroup = this.scene.physics.add.group();
  }

  this.scene.itemGroup.add(this.item);


    } else if (type === 1) {
      this.item = new IceFlower(this.scene, 3, -5,'flower',this);

    }

    if (this.item) {
      this.add(this.item);
    }
  }
```

Enemigos

```
export default class Fireball extends Phaser.Physics.Arcade.Sprite {
  constructor(scene, x, y, texture = 'fire') {
    super(scene, x, y, texture, 0);
    this.scene = scene;
```

```
    this.speed = 100;
    this.direction = 1;
    this.isDying = false;
    // Agregar a escena y sistema de físicas
    scene.add.existing(this);
    scene.physics.add.existing(this);

    this.setBounce(0);
    this.setCollideWorldBounds(true);
    this.setGravityY(0);
    this.setScale(2, 3);
    this.refreshBody();

    this.animator();
  }
```

Grupos :

```
 this.Spawners = this.physics.add.group();
   this.Spawners.add(new FlowerSpawner(this,this.scale.width / 2 +
300,this.scale.height - 124, 'spawner'));
   this.Spawners.add(new HammerSpawner(this,this.scale.width / 2 -
460,this.scale.height - 304, 'spawner'));
```

excepto tipo container,para esos mejor usar

```
this.kongs = [new Kong(this, this.scale.width / 2 - 300,
```

colisiones

```
    this.Koopas.forEach(k => this.physics.add.collider(k,
this.flores));
    this.physics.add.collider(this.Spawners, this.flores);
```

overlap

```
  colisiones() {
    //fireballs
```

```
    this.fireballes.forEach(fireball => {
      this.roses.children.iterate(rose => {
        if (this.physics.overlap(fireball, rose)) {
          fireball.cambiarDireccion();
        }
      });

      if (this.physics.overlap(fireball, this.player) &&
!this.player.golpeado) {
        this.player.hurt();
      }
    });
//koopas
    this.Koopas.forEach(koopa => {
      this.roses.children.iterate(rose => {
        if (this.physics.overlap(koopa, rose)) {
          koopa.cambiarDireccion();
        }
      });

      if (this.physics.overlap(koopa, this.player) &&
!this.player.golpeado) {
        this.player.hurt();
      }
    });
//boos
    this.Boos.forEach(boo => {
      if (this.physics.overlap(boo, this.player) &&
!this.player.golpeado) {
        this.player.hurt();
      }
    });
//barriles
    this.barrelGroup.children.iterate(barril => {
      this.roses.children.iterate(rose => {
        if (this.physics.overlap(barril, rose)) {
          barril.changeDirection();
        }
      });

      if (this.physics.overlap(barril, this.player) &&
!this.player.golpeado) {
        this.player.hurt();
```

```
      }
    });

//snowball
this.SnowballGroup.children.iterate(iceball => {
  this.Koopas.forEach(koopa => {
    if (this.physics.overlap(iceball, koopa)) {
      console.log('Iceball tocó a Koopa');
      iceball.destroy();
koopa.Iced();
this.Koopas = this.Koopas.filter(k => k !== koopa);
this.puntos += 200;

    }
  });

  this.Boos.forEach(boo => {
    if (this.physics.overlap(iceball, boo)) {
      console.log('Iceball tocó a Boo');
      iceball.destroy();
      boo.Iced();
      this.Boos = this.Boos.filter(k => k !== boo);
       this.puntos += 200;
    }
  });

  this.fireballes.forEach(fireball => {
    if (this.physics.overlap(iceball, fireball)) {
      console.log('Iceball tocó a Kong');
      iceball.destroy();

    }
  });
});
//items
this.itemGroup.children.iterate(item => {
  this.Koopas.forEach(koopa => {
    if (this.physics.overlap(item, koopa)) {
      console.log('Iceball tocó a Koopa');
koopa.Iced();
this.Koopas = this.Koopas.filter(k => k !== koopa);
this.puntos += 200;
```

```
    }
  });

  this.fireballes.forEach(fireball => {
    if (this.physics.overlap(item, fireball)) {
      console.log('Iceball tocó a Kong');
      fireball.Die();
     this.fireballes = this.fireballes.filter(k => k !== fireball);

    }
  });
});



    this.ThrowingGroup.children.iterate(barril => {
      if (this.physics.overlap(barril, this.flores)) {
        barril.cambiarDirection(Phaser.Math.Between(-15, 15));
      }

      if (this.physics.overlap(barril, this.player) &&
!this.player.golpeado) {
        this.player.hurt();
      }
    });




this.Spawners.children.iterate(spawner => {
  if (this.physics.overlap(spawner, this.player) &&
spawner.getCreado()) {
    console.log("tocado");
    console.log(spawner.getItem());
    this.player.addItem(spawner.getItem());
  }
});


    this.kongs.forEach(kong => {
      if (this.physics.overlap(kong, this.player)) {
        kong.Die();
```

```
        this.gameOver = true;
        this.puntos += this.counter * 10;
        this.music.stop();
        this.music3.play();

        this.player.forceDeathState();

        this.time.delayedCall(2000, () => {
          if (this.music3.isPlaying) this.music3.stop();
          this.scene.start('NextScene', { puntos: this.puntos , nivel:
1 });
           this.scene.stop();
        });
      }
    });
  }
```

contadores y puntos

```
    this.contadorTexto = this.add.text(20, 20, 'Tiempo: 120', {
fontSize: '24px', color: '#ffffff' });
    this.puntosTexto = this.add.text(20, 60, 'Puntos: 0', { fontSize:
'24px', color: '#ffffff' });
    this.vidasTexto = this.add.text(500, 20, 'Vidas: 3', { fontSize:
'24px', color: '#ffffff' });

    this.time.addEvent({
      delay: 1000,
      callback: this.actualizarContador,
      callbackScope: this,
      loop: true
    });
  actualizarContador() {
    this.counter--;
    this.contadorTexto.setText('Tiempo: ' + this.counter);
  }



    this.puntosTexto.setText('Puntos: ' + this.puntos);
    this.vidasTexto.setText('Vidas: ' + this.player.lifes);
```

Musica

```
    this.music = this.sound.add('nivel', { loop: true, volume: 0.5 });
    this.music.play();

    this.music2 = this.sound.add('MarioMuere', { loop: false, volume: 1
});
    this.music3 = this.sound.add('DonkeyMuere', { loop: false, volume:
1 });
```

camera

this.cameras.main.setBounds(0, 0, 2000, 600);

```
   this.cameras.main.scrollX = this.player.x - this.scale.width / 2;

    // Control vertical de la cámara permitiendo que suba hasta
worldTopLimit
    const targetY = this.player.y - this.scale.height / 2;

    if (targetY < this.cameras.main.scrollY) {
      // Suaviza la subida
      this.cameras.main.scrollY = Phaser.Math.Interpolation.Linear(
        [this.cameras.main.scrollY, targetY], 0.1);
    } else {
      // No dejar bajar la cámara demasiado abajo ni más abajo que
targetY
      this.cameras.main.scrollY = Math.min(this.cameras.main.scrollY,
targetY);
      this.cameras.main.scrollY = Math.max(this.cameras.main.scrollY,
this.worldTopLimit);
    }
```

horizontal

this.cameras.main.scrollX = this.player.x - this.scale.width / 2;

```
    this.cameras.main.scrollY = 0; // fija en vertical
ambos
update() {
  // Seguir al jugador en X e Y, centrando la cámara
  this.cameras.main.scrollX = this.player.x - this.scale.width / 2;
  this.cameras.main.scrollY = this.player.y - this.scale.height / 2;
}
```

Distancias

```
this.jarrones = this.physics.add.staticGroup();

for (let i = 40; i < recorridoTotal; i += 10) {
    const x = i * 80; // 800px = 10m → 80px = 1m
    const y = 500;    // Ajustar según el suelo
    this.jarrones.create(x, y, 'fire');
}

this.physics.add.collider(player, this.jarrones, this.perder, null, this);
```

Background tileao

```
this.background = this.add.tileSprite(
  0,                // x
  0,                // y
  this.scale.width,   // width
  this.scale.height * 3, // height (tile 3 veces hacia arriba)
  'background'        // texture key
).setOrigin(0, 0);
```

tilemaps

```
// Preload: cargar tileset y mapa
preload() {
    this.load.image('tiles', 'assets/tileset.png');
    this.load.tilemapTiledJSON('map', 'assets/map.json');
}

// Create: añadir el mapa a la escena
create() {
    const map = this.make.tilemap({ key: 'map' });
```

```
    const tileset = map.addTilesetImage('tileset', 'tiles'); // nombre coincidente con el definido en Tiled

    // Capa de fondo o suelo
    const backgroundLayer = map.createLayer('Background', tileset);
    backgroundLayer.setCollisionByExclusion([-1]); // Hacer que colisione todo menos los tiles vacíos
}
```