Tarea individual 11 - Menú y Salto con gravedad

Irene Jurado Castillo

Objetivo

Crear un juego en LibGDX que implemente una acción física en la que el jugador salta, afectado por la gravedad, utilizando las interfaces `Screen` y `Game` para manejar un menú dentro del juego.

Instrucciones

1. **Crea un proyecto de LibGDX.**
   Inicia el proyecto en LibGDX y configura el entorno de desarrollo necesario.
2. **Implementación de la acción física de salto:**

   Haz que el jugador pueda saltar con una acción física real, en la que la gravedad afecte al personaje.

   - Usa un `Sprite` para representar al jugador (por ejemplo, Mario).
   - Aplica un comportamiento de salto, donde el jugador puede saltar cuando presiona una tecla (por ejemplo, la tecla de **espacio**).
   - La gravedad debe actuar sobre el personaje, haciendo que suba al saltar y luego baje cuando la gravedad lo atrae hacia el suelo.
   - Controla el movimiento del jugador con las teclas de flecha o **A** y **D** para moverlo lateralmente.
   - El salto debe permitir que el jugador se mueva lateralmente mientras está en el aire.
3. **Implementación de la interfaz `Screen` y `Game` para el menú:**

   Implementa un sistema de menú usando las interfaces `Screen` y `Game` en LibGDX.

   - Crea una pantalla de inicio donde el jugador pueda presionar un botón para empezar el juego.
   - Utiliza `Game.setScreen()` para cambiar entre las pantallas de menú y juego.
   - Define una pantalla de juego donde se manejará la lógica de la física del salto y el movimiento.
   - La pantalla del menú debe incluir un botón para empezar el juego y otro para salir.
4. **Física del salto:**

   Para implementar la física del salto, realiza lo siguiente:

   1. Parámetros de la función:

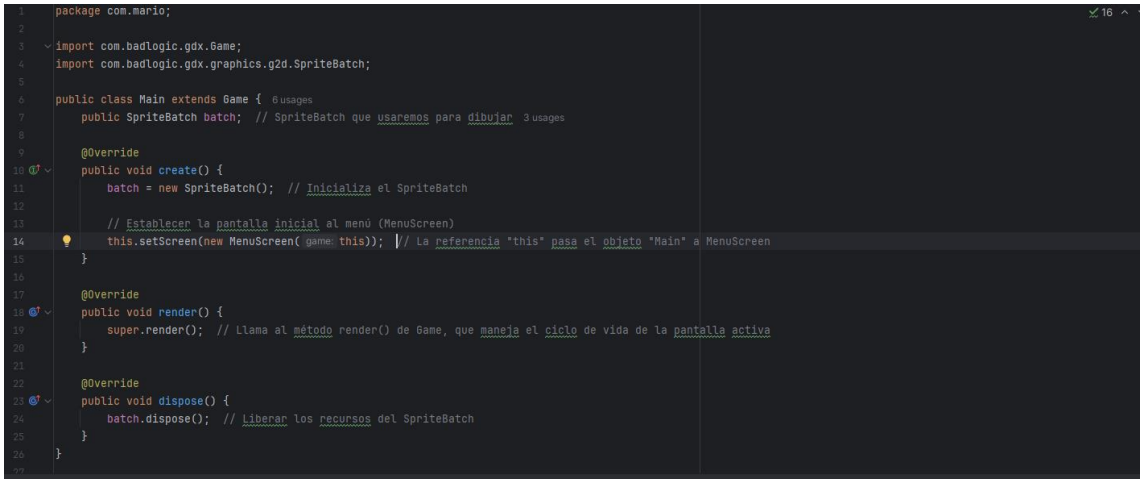
      1. **velocidadInicial**: La velocidad inicial con la que el objeto comienza el salto (hacia arriba).
      2. **gravedad**: La constante que representa la aceleración debido a la gravedad (generalmente, -9.81 m/s²).

3. **tiempoDelta**: El incremento de tiempo por cada ciclo de simulación.
4. **posicionVertical**: La posición del objeto en el eje vertical (y), que inicialmente es 0 cuando el objeto comienza en el suelo.

2. Procedimiento:

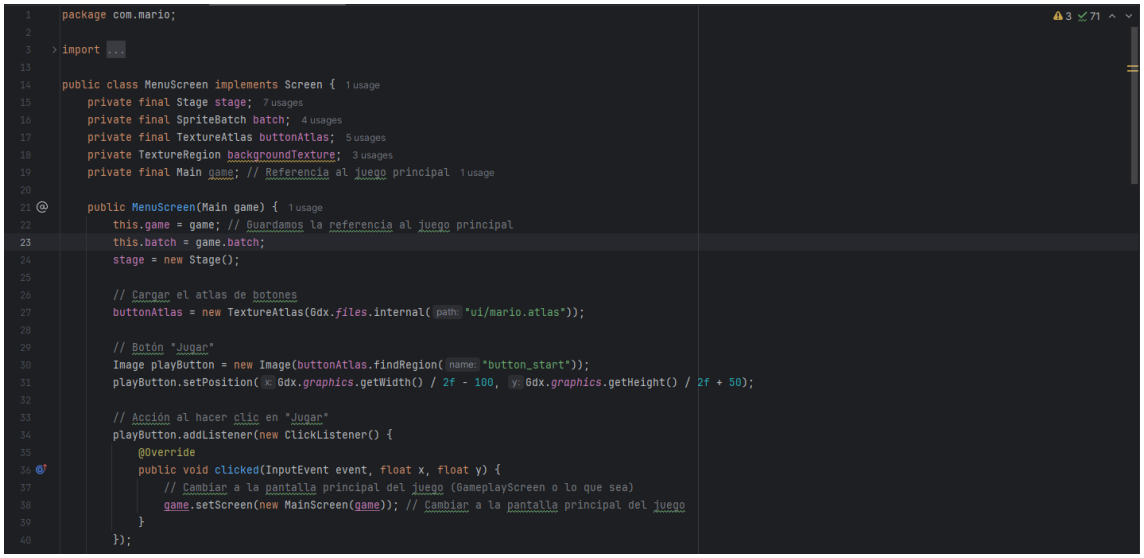
1. Inicializa la velocidad con la **velocidadInicial** y **posiciónVertical** con 0 o el suelo.
2. En cada paso de simulación: Se actualiza la velocidad restando la gravedad multiplicada por el **tiempoDelta**.
3. Se calcula la nueva posición Vertical sumando la velocidad por el tiempo Delta.
4. El objeto sigue subiendo mientras la velocidad sea positiva.
5. Cuando la velocidad se hace negativa, el objeto empieza a descender.
6. El ciclo termina cuando el posición Vertical alcanza 0 (es decir, cuando el objeto toca el suelo).

CODIGO DE LA CLASE MAIN

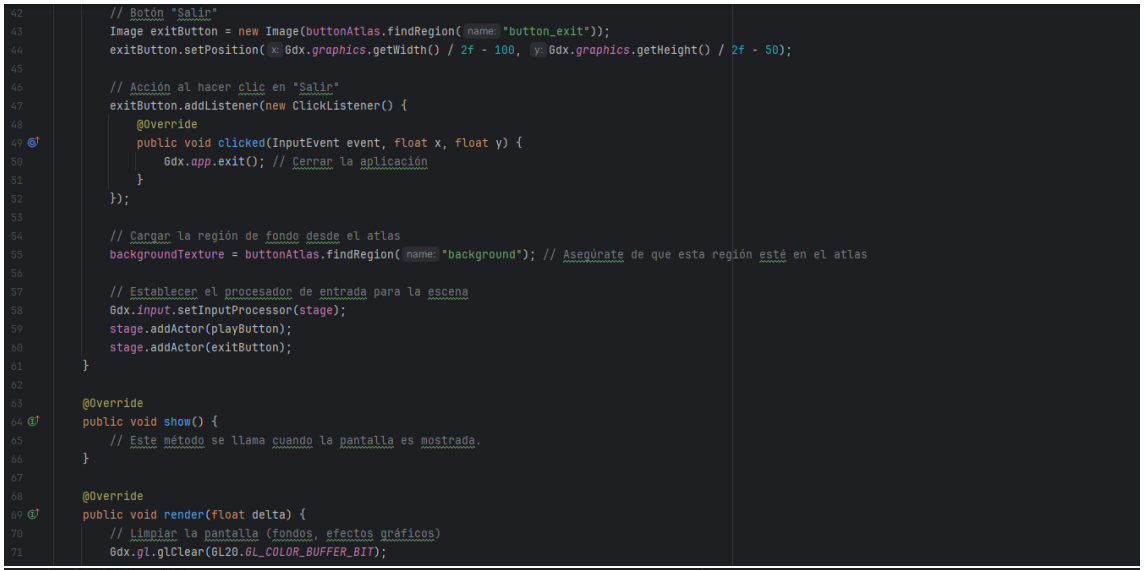


```
package com.mario;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

public class Main extends Game {  6 usages
    public SpriteBatch batch;  // SpriteBatch que usaremos para dibujar  3 usages

    @Override
    public void create() {
        batch = new SpriteBatch();  // Inicializa el SpriteBatch

        // Establecer la pantalla inicial al menú (MenuScreen)
        this.setScreen(new MenuScreen( game: this));  // La referencia "this" pasa el objeto "Main" a MenuScreen
    }

    @Override
    public void render() {
        super.render();  // Llama al método render() de Game, que maneja el ciclo de vida de la pantalla activa
    }

    @Override
    public void dispose() {
        batch.dispose();  // Liberar los recursos del SpriteBatch
    }
}
```

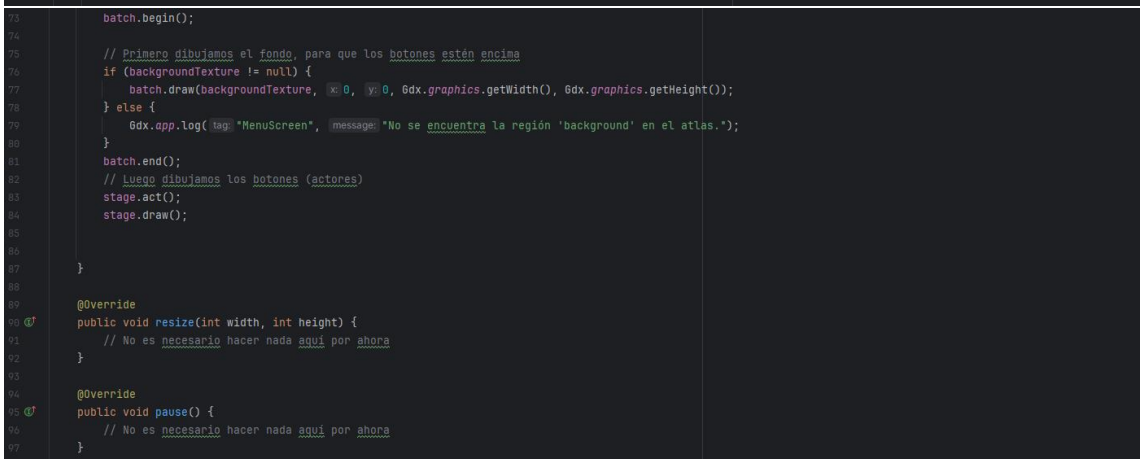
CODIGO DE LA CLASE MENUSCREEN



```java
package com.mario;

import ...

public class MenuScreen implements Screen {  1 usage
    private final Stage stage;  7 usages
    private final SpriteBatch batch;  4 usages
    private final TextureAtlas buttonAtlas;  5 usages
    private TextureRegion backgroundTexture;  3 usages
    private final Main game; // Referencia al juego principal  1 usage

    public MenuScreen(Main game) {  1 usage
        this.game = game; // Guardamos la referencia al juego principal
        this.batch = game.batch;
        stage = new Stage();

        // Cargar el atlas de botones
        buttonAtlas = new TextureAtlas(Gdx.files.internal( path: "ui/mario.atlas"));

        // Botón "Jugar"
        Image playButton = new Image(buttonAtlas.findRegion( name: "button_start"));
        playButton.setPosition( x: Gdx.graphics.getWidth() / 2f - 100,  y: Gdx.graphics.getHeight() / 2f + 50);

        // Acción al hacer clic en "Jugar"
        playButton.addListener(new ClickListener() {
            @Override
            public void clicked(InputEvent event, float x, float y) {
                // Cambiar a la pantalla principal del juego (GameplayScreen o lo que sea)
                game.setScreen(new MainScreen(game)); // Cambiar a la pantalla principal del juego
            }
        });
```



```java
        // Botón "Salir"
        Image exitButton = new Image(buttonAtlas.findRegion( name: "button_exit"));
        exitButton.setPosition( x: Gdx.graphics.getWidth() / 2f - 100,  y: Gdx.graphics.getHeight() / 2f - 50);

        // Acción al hacer clic en "Salir"
        exitButton.addListener(new ClickListener() {
            @Override
            public void clicked(InputEvent event, float x, float y) {
                Gdx.app.exit(); // Cerrar la aplicación
            }
        });

        // Cargar la región de fondo desde el atlas
        backgroundTexture = buttonAtlas.findRegion( name: "background"); // Asegúrate de que esta región esté en el atlas

        // Establecer el procesador de entrada para la escena
        Gdx.input.setInputProcessor(stage);
        stage.addActor(playButton);
        stage.addActor(exitButton);
    }

    @Override
    public void show() {
        // Este método se llama cuando la pantalla es mostrada.
    }

    @Override
    public void render(float delta) {
        // Limpiar la pantalla (fondos, efectos gráficos)
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
```



```java
        batch.begin();

        // Primero dibujamos el fondo, para que los botones estén encima
        if (backgroundTexture != null) {
            batch.draw(backgroundTexture,  x: 0,  y: 0, Gdx.graphics.getWidth(), Gdx.graphics.getHeight());
        } else {
            Gdx.app.log( tag: "MenuScreen",  message: "No se encuentra la región 'background' en el atlas.");
        }
        batch.end();
        // Luego dibujamos los botones (actores)
        stage.act();
        stage.draw();


    }

    @Override
    public void resize(int width, int height) {
        // No es necesario hacer nada aquí por ahora
    }

    @Override
    public void pause() {
        // No es necesario hacer nada aquí por ahora
    }
```

```java
    @Override
    public void resume() {
        // No es necesario hacer nada aquí por ahora
    }

    @Override
    public void hide() {
        // No es necesario hacer nada aquí por ahora
    }

    @Override
    public void dispose() {
        // Limpiar los recursos cuando ya no se usen
        stage.dispose();
        buttonAtlas.dispose();
    }
}
```

CODIGO DE LA CLASE MAINSCREEN

```java
package com.mario;

import ...

public class MainScreen implements Screen {  1 usage
    public SpriteBatch batch;  5 usages
    private TextureAtlas marioAtlas;  6 usages
    private Animation<TextureRegion> marioAnimation;  no usages
    private float stateTime;  6 usages
    private Vector2 marioPosition;  13 usages
    private boolean moving = false;  6 usages
    static boolean isJumping = false;  4 usages
    private float verticalSpeed = 0f;  // Velocidad vertical  4 usages
    private final float groundLevel = 100f;  // Nivel del suelo (piso)  4 usages
    private TextureRegion marioIdle;  3 usages
    private TextureAtlas marioAtlas1;  1 usage
    private Array<TextureRegion> littleMarioFrames;  // Aquí guardamos los frames de little_mario  3 usages
    private Array<TextureRegion> jumpframes;  3 usages
    private Animation<TextureRegion> littleMarioAnimation;  2 usages
    private Animation<TextureRegion> jump;  2 usages
    private float speed = 200f;  3 usages
    private boolean facingRight = true;  4 usages
    private TextureRegion background;  no usages


    public MainScreen(Main game) {  1 usage
        this.batch = game.batch;  // Referencia al SpriteBatch del juego principal
        marioAtlas = new TextureAtlas(Gdx.files.internal( path: "ui/marioandenemies.atlas"));
        marioAtlas1 = new TextureAtlas(Gdx.files.internal( path: "ui/mario.atlas"));

        // Cargar animación de Mario
        Array<TextureRegion> allFrames = new Array<>();
        allFrames.add(marioAtlas.findRegion( name: "little_mario"));
        littleMarioFrames = new Array<TextureRegion>();
        jumpframes = new Array<TextureRegion>();
        // Agregamos los 14 frames de little_mario (14 sprites en total, cada uno de 16x16 px)
        int spriteWidth = 15; // ancho
        int spriteHeight = 16; // alto
```

```java
        // Usamos las coordenadas y tamaños proporcionados en el atlas para crear los frames
        for (int i = 0; i < 4; i++) {
            littleMarioFrames.add(new TextureRegion(marioAtlas.findRegion( name: "little_mario"),
                x: 1 + i * spriteWidth,  y: 11, spriteWidth, spriteHeight));
        }
        for (int i = 5; i < 6; i++) {
            jumpframes.add(new TextureRegion(marioAtlas.findRegion( name: "little_mario"),
                x: 6 + i * spriteWidth,  y: 11, spriteWidth, spriteHeight));
        }

        // Crear la animación para little_mario
        littleMarioAnimation = new Animation<>( frameDuration: 0.1f, littleMarioFrames); // Cada sprite se muestra 0.1 segundos
        jump = new Animation<>( frameDuration: 0.1f, jumpframes); // Cada sprite se muestra 0.1 segundos
        // Inicializamos el tiempo del estado de la animación
        stateTime = 0f;
        marioIdle = marioAtlas.findRegion( name: "little_mario");

        int count = 4;
        for (int i = 0; i < count; i++) {
            allFrames.add(new TextureRegion(marioIdle,  x: i * 15,  y: 0,  width: 16, marioIdle.getRegionHeight()));

            // Posición inicial de Mario
            marioPosition = new Vector2( x: 100, groundLevel);
            // Inicializar estado de animación
            stateTime = 0f;
            moving = false;
        }
    }
```

```java
@Override
public void render(float delta) {
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);  // Limpiar la pantalla

    handleInput(delta);  // Detecta las entradas del jugador

    batch.begin();

    // Si Mario se está moviendo, avanzamos en la animación
    if (moving) {
        stateTime += Gdx.graphics.getDeltaTime();
    } else {
        stateTime = 0; // Si no se mueve, reiniciamos la animación para que se quede en el primer frame
    }
    // Seleccionar el frame correcto según el estado (saltando o no)
    TextureRegion currentFrame;
    if (isJumping) {
        currentFrame = jump.getKeyFrame(stateTime, looping: false);
    } else {
        currentFrame = littleMarioAnimation.getKeyFrame(stateTime, looping: true);
    }

    // Verificar si hay que voltear el sprite
    if (!facingRight && !currentFrame.isFlipX()) {
        currentFrame.flip( x: true, y: false);
    } else if (facingRight && currentFrame.isFlipX()) {
        currentFrame.flip( x: true, y: false);
    }

    // Dibujar a Mario con el frame actual respetando la dirección
    batch.draw(currentFrame, marioPosition.x, marioPosition.y,
        width: currentFrame.getRegionWidth() * 2f, height: currentFrame.getRegionHeight() * 2f);
```

```java
        // Aplicar gravedad
        if (marioPosition.y > groundLevel) {
            float gravity = -500f;
            verticalSpeed += gravity * Gdx.graphics.getDeltaTime();
            marioPosition.y += verticalSpeed * Gdx.graphics.getDeltaTime();
        } else {
            marioPosition.y = groundLevel;
            verticalSpeed = 0;
            isJumping = false;
        }

        batch.end();
    }

    @Override
    public void resize(int width, int height) {

    }

    @Override
    public void pause() {

    }

    @Override
    public void resume() {

    }

    @Override
    public void hide() {

    }
```

```java
private void handleInput(float deltaTime) {  1 usage
    moving = false;

    if (Gdx.input.isKeyPressed(Input.Keys.A)) {
        marioPosition.x -= speed * deltaTime; // Mover a la izquierda
        moving = true;
        facingRight = false; // Ahora mira a la izquierda
    }
    if (Gdx.input.isKeyPressed(Input.Keys.D)) {
        marioPosition.x += speed * deltaTime; // Mover a la derecha
        moving = true;
        facingRight = true; // Ahora mira a la derecha
    }

    // Control de salto (tecla de espacio)
    if (Gdx.input.isKeyPressed(Input.Keys.SPACE)) {
        if (!isJumping) {  // Solo permitir saltar si Mario no está saltando
            marioPosition.y += speed * deltaTime;
            verticalSpeed = 260f;  // Iniciar el salto con velocidad hacia arriba
            isJumping = true;  // Cambiar el estado a que está saltando
            moving = true;
        }
    }

    // Evitar que Mario salga de los límites de la pantalla
    marioPosition.x = Math.max(0, Math.min(marioPosition.x, Gdx.graphics.getWidth() - 64));
    marioPosition.y = Math.max(groundLevel, marioPosition.y); // Asegurar que Mario no pase del suelo
}

@Override
public void dispose() {
    batch.dispose();
    marioAtlas.dispose();
}

@Override
public void show() {
    // Este método se implementa pero no se hace nada, ya que no es necesario.
}
```

CAPUTRAS DE PANTALLA DE LA APP AL INICIARSE

LE DAMOS A START PARA INICIAR EL JUEGO O EXIT PARA CERRAR LA APP

CUANDO SE PRESIONA LA TECLA ESPACIO Y SE ANDA CON LAS TECLAS A / D

CUANDO CAMINA CON LAS TECLAS A / D