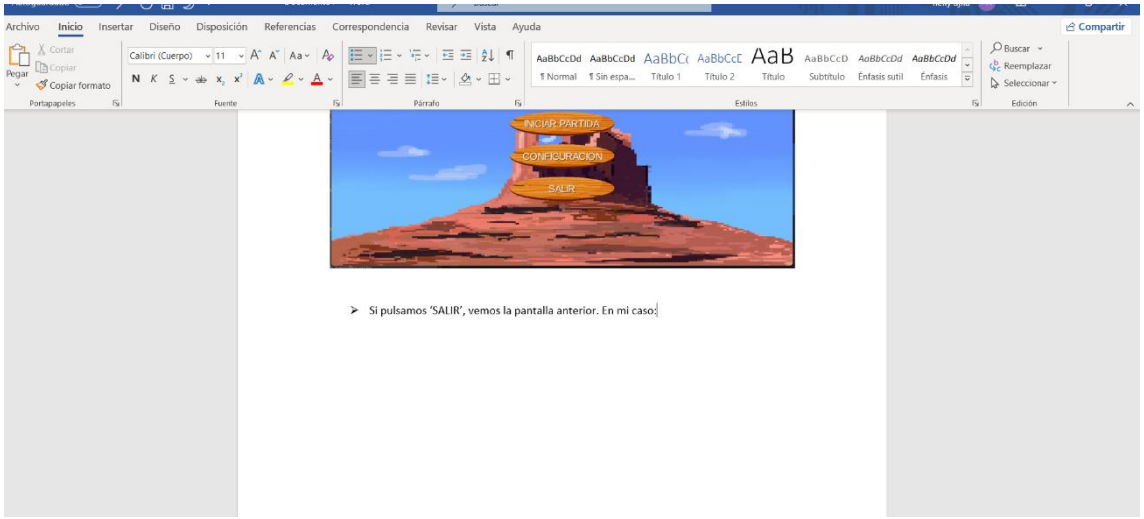
A continuación, voy a enseñar el pdf con el código del proyecto:

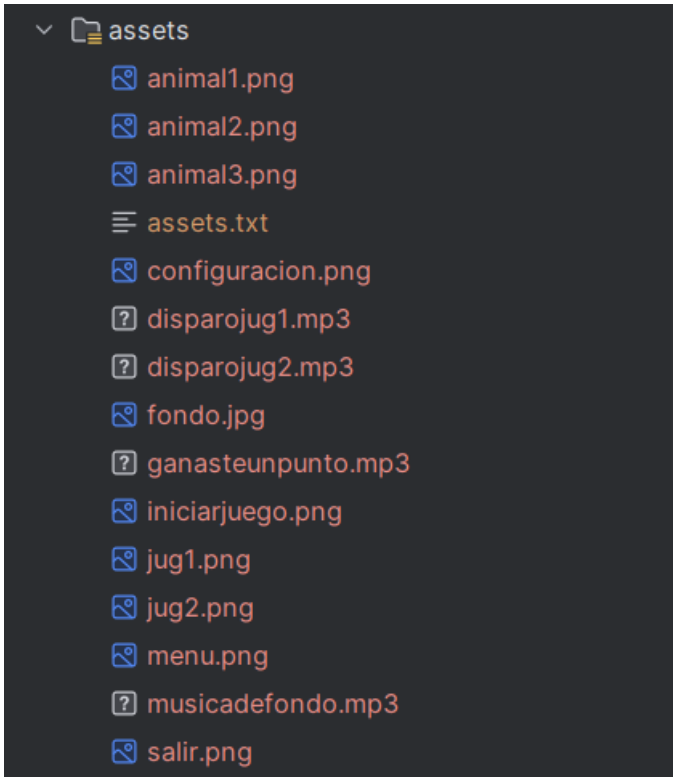
- Si pulsamos 'SALIR', vemos la pantalla anterior. En mi caso:



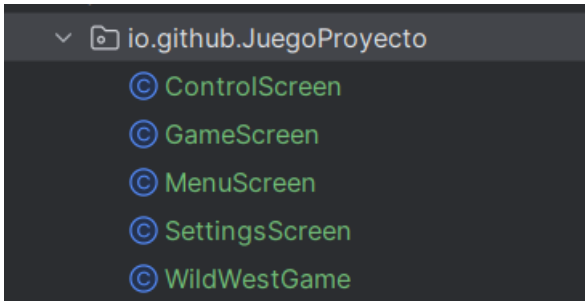
- Si pulsamos 'Iniciar partida', disfrutamos de la partida:

- Y si nos metemos en configuración:

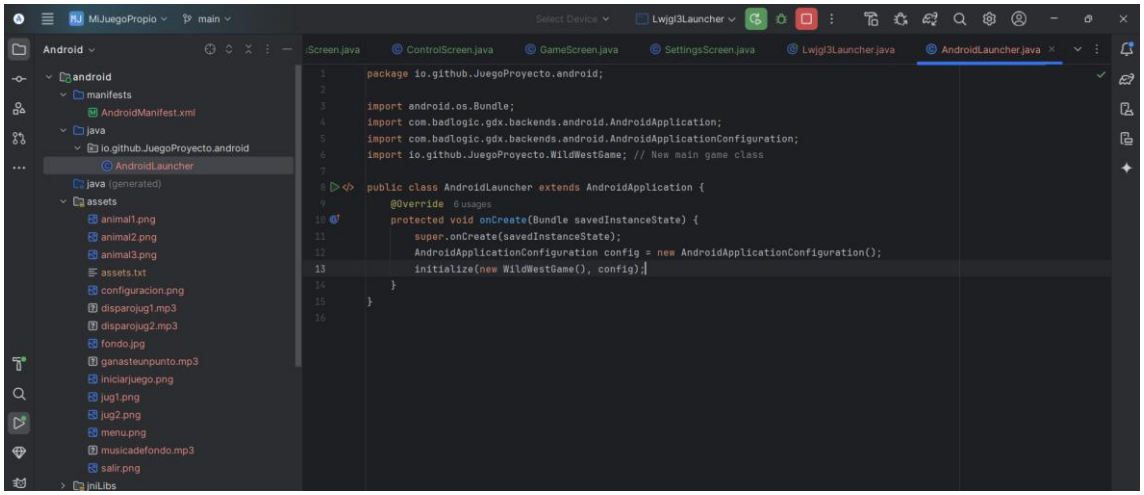
El proyecto ha sido creado tras guardar en los assets los siguientes recursos:

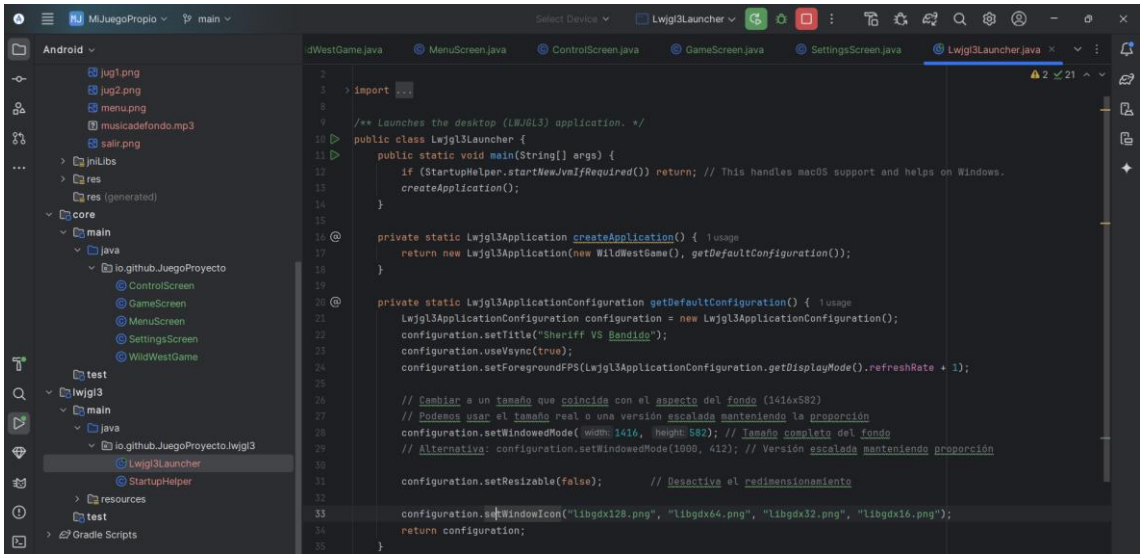


Y crear en el io.github.JuegoProyecto las siguientes clases:



Aunque antes modificamos esto:





WildWestGame, es el inicio:

```java
package io.github.JuegoProyecto;

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


/*
*
*La clase WildWestGame es la clase principal de tu juego que extiende
*la clase Game de LibGDX. Esta clase actúa como el punto de entrada y
*controlador principal de tu aplicación
*
* */
public class WildWestGame extends Game {
    private SpriteBatch batch;

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

        this.setScreen(new MenuScreen(this));
    }

    @Override
    public void render() {
        super.render();
    }

    @Override
    public void dispose() {
        batch.dispose();
        if (this.getScreen() != null) {
            this.getScreen().dispose();
        }
    }

    public SpriteBatch getBatch() {
        return batch;
```

```
    }
}
```

MenuScreen, que se encarga de las 3 opciones :

```
package io.github.JuegoProyecto;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.math.Rectangle;


public class MenuScreen implements Screen {

    private final WildWestGame game;

    // SpriteBatch compartido
    private SpriteBatch batch;

    // Recursos del menú
    private Texture menuBackground;
    private Texture playButton;
    private Texture settingsButton;
    private Texture exitButton;

    // Instancia de la pantalla de controles
    private ControlScreen controlsScreen;

    private Rectangle playBounds;
    private Rectangle settingsBounds;
    private Rectangle exitBounds;

    // Instancia de la pantalla de configuración
    private SettingsScreen settingsScreen;

    // Configuración del juego
    private float gameDuration = 1f;
    private boolean sfxOn   = true;   // Efectos activados por defecto
    private boolean musicOn = true;   // Música activada por defecto


    // Current screen being shown
    private Screen currentScreen;

    // Variable de clase
    private ShapeRenderer shapeRenderer;

    // Fuente para el texto de los botones
    private BitmapFont font;

    // Fuente para el título del juego
```

```
    private BitmapFont titleFont;


    // En el constructor
    public MenuScreen(WildWestGame game) {
        this.game = game;
        this.batch = game.getBatch(); // Usar el SpriteBatch
compartido
        this.currentScreen = this;
        this.shapeRenderer = new ShapeRenderer(); // Inicializar una
sola vez

        // Inicializar la fuente para los botones
        this.font = new BitmapFont();
        this.font.getData().setScale(2.0f); // Aumentar el tamaño de
la fuente
        this.font.setColor(Color.WHITE); // Establecer color blanco
para el texto

        // Inicializar la fuente para el título
        this.titleFont = new BitmapFont();
        this.titleFont.getData().setScale(3.5f); // Fuente más grande
para el título
        this.titleFont.setColor(Color.WHITE); // Establecer color
blanco para el título
    }

    @Override
    public void show() {
        try {
            // Mostrar información de depuración sobre directorios
            System.out.println("Directorio de trabajo: " +
Gdx.files.getLocalStoragePath());
            System.out.println("Intentando cargar recursos del
menú...");

            // Cargar texturas con manejo de errores y depuración
extendida
            try {
                // Verificar si el archivo existe antes de cargarlo
                if (Gdx.files.internal("menu.png").exists()) {
                    System.out.println("menu.png encontrado,
intentando cargar...");
                    menuBackground = new Texture("menu.png");
                    System.out.println("Menu background cargado
correctamente");
                } else {
                    System.out.println("menu.png NO EXISTE en la ruta
predeterminada");
                    // Intentar buscar en ubicaciones alternativas
                    String[] posiblesPaths = {"assets/menu.png",
"data/menu.png", "images/menu.png"};
                    boolean encontrado = false;

                    for (String path : posiblesPaths) {
                        System.out.println("Intentando cargar desde: "
+ path);
                        if (Gdx.files.internal(path).exists()) {
                            menuBackground = new
Texture(Gdx.files.internal(path));
                            System.out.println("Menu background
```

```
cargado desde " + path);
                            encontrado = true;
                            break;
                        }
                    }

                    if (!encontrado) {
                        System.out.println("No se encontró menu.png en
ninguna ruta, usando fallback");
                        // Crear una textura de fallback si falla la
carga
                        Pixmap pixmap = new
Pixmap(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(),
Pixmap.Format.RGBA8888);
                        pixmap.setColor(Color.BROWN);
                        pixmap.fill();
                        menuBackground = new Texture(pixmap);
                        pixmap.dispose();
                    }
                }
            } catch (Exception e) {
                System.err.println("Error al cargar menu.png: " +
e.getMessage());
                e.printStackTrace(); // Esto mostrará la traza
completa del error
                // Crear una textura de fallback si falla la carga
                Pixmap pixmap = new Pixmap(Gdx.graphics.getWidth(),
Gdx.graphics.getHeight(), Pixmap.Format.RGBA8888);
                pixmap.setColor(Color.BROWN);
                pixmap.fill();
                menuBackground = new Texture(pixmap);
                pixmap.dispose();
            }


            // Definir áreas de botones con posiciones que aseguren
visibilidad
            float centerX = Gdx.graphics.getWidth() / 2f;
            float centerY = Gdx.graphics.getHeight() / 2f;

            // Posicionar botones en el centro de la pantalla y
hacerlos más grandes
            playBounds = new Rectangle(centerX - 180, centerY + 100,
360, 90);
            settingsBounds = new Rectangle(centerX - 180, centerY,
360, 90);
            exitBounds = new Rectangle(centerX - 180, centerY - 100,
360, 90);

            // Similar proceso de diagnóstico para los botones
            try {
                boolean todosCargados = true;

                // Verificar e intentar cargar cada botón
individualmente
                if (Gdx.files.internal("iniciarjuego.png").exists()) {
                    playButton = new Texture("iniciarjuego.png");
                    System.out.println("iniciarjuego.png cargado
correctamente");
                } else {
                    System.out.println("No se encontró
```

```
iniciarjuego.png");
                    todosCargados = false;
                }

                if (Gdx.files.internal("configuracion.png").exists())
{
                    settingsButton = new Texture("configuracion.png");
                    System.out.println("configuracion.png cargado
correctamente");
                } else {
                    System.out.println("No se encontró
configuracion.png");
                    todosCargados = false;
                }

                if (Gdx.files.internal("salir.png").exists()) {
                    exitButton = new Texture("salir.png");
                    System.out.println("salir.png cargado
correctamente");
                } else {
                    System.out.println("No se encontró salir.png");
                    todosCargados = false;
                }

                if (!todosCargados) {
                    System.out.println("Algunos botones no se
cargaron, usando fallback");
                    // Crear fallbacks con colores distintivos para
cada botón
                    Pixmap playPixmap = new Pixmap(360, 90,
Pixmap.Format.RGBA8888);
                    playPixmap.setColor(new Color(0.8f, 0.6f, 0.4f,
1)); // Color beige claro
                    playPixmap.fill();

                    Pixmap settingsPixmap = new Pixmap(360, 90,
Pixmap.Format.RGBA8888);
                    settingsPixmap.setColor(new Color(0.9f, 0.5f,
0.3f, 1)); // Color anaranjado
                    settingsPixmap.fill();

                    Pixmap exitPixmap = new Pixmap(360, 90,
Pixmap.Format.RGBA8888);
                    exitPixmap.setColor(new Color(0.8f, 0.4f, 0.2f,
1)); // Color marrón rojizo
                    exitPixmap.fill();

                    if (playButton == null) {
                        playButton = new Texture(playPixmap);
                    }
                    if (settingsButton == null) {
                        settingsButton = new Texture(settingsPixmap);
                    }
                    if (exitButton == null) {
                        exitButton = new Texture(exitPixmap);
                    }

                    playPixmap.dispose();
                    settingsPixmap.dispose();
                    exitPixmap.dispose();
                }
```

```
            } catch (Exception e) {
                System.err.println("Error al cargar botones: " +
e.getMessage());
                e.printStackTrace();

                // Crear texturas de fallback con colores distintivos
                Pixmap playPixmap = new Pixmap(360, 90,
Pixmap.Format.RGBA8888);
                playPixmap.setColor(new Color(0.8f, 0.6f, 0.4f, 1));
                playPixmap.fill();

                Pixmap settingsPixmap = new Pixmap(360, 90,
Pixmap.Format.RGBA8888);
                settingsPixmap.setColor(new Color(0.9f, 0.5f, 0.3f,
1));
                settingsPixmap.fill();

                Pixmap exitPixmap = new Pixmap(360, 90,
Pixmap.Format.RGBA8888);
                exitPixmap.setColor(new Color(0.8f, 0.4f, 0.2f, 1));
                exitPixmap.fill();

                playButton = new Texture(playPixmap);
                settingsButton = new Texture(settingsPixmap);
                exitButton = new Texture(exitPixmap);

                playPixmap.dispose();
                settingsPixmap.dispose();
                exitPixmap.dispose();
            }

            // Crear instancia de SettingsScreen
            settingsScreen = new SettingsScreen(this);

            System.out.println("MenuScreen show() completado
correctamente");
        } catch (Exception e) {
            System.err.println("Error general en show(): " +
e.getMessage());
            e.printStackTrace();
        }
    }

    @Override
    public void render(float delta) {
        if (currentScreen != this) {
            // Si estamos mostrando otra pantalla, delegar el render
            currentScreen.render(delta);

            // Comprobar escape para volver al menú desde settings o
controls
            if ((currentScreen == settingsScreen || currentScreen ==
controlsScreen)
                && Gdx.input.isKeyJustPressed(Input.Keys.ESCAPE)) {
                returnToMenu();
            }
            return;
        }

        // Limpiar la pantalla una sola vez aquí
        Gdx.gl.glClearColor(0.2f, 0.2f, 0.4f, 1); // Fondo azul oscuro
```

```
para distinguir
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        renderMenu();
    }

    private void renderMenu() {
        System.out.println("Dibujando menú...");

        // Mensajes de diagnóstico
        System.out.println("menuBackground es null? " +
(menuBackground == null));
        System.out.println("Tamaño de ventana: " +
Gdx.graphics.getWidth() + "x" + Gdx.graphics.getHeight());
        System.out.println("Posiciones de botones: Play(" +
playBounds.x + "," + playBounds.y + ")");

        // Dibujar texturas
        batch.begin();
        if (menuBackground != null) {
            batch.draw(menuBackground, 0, 0, Gdx.graphics.getWidth(),
Gdx.graphics.getHeight());
            System.out.println("Dibujando fondo del menú");

            // Para centrar el texto en los botones
            GlyphLayout layout = new GlyphLayout();

            // Dibujar el título "WildWestGame" encima del botón
"Iniciar Partida"
            layout.setText(titleFont, "WildWestGame");
            float titleX = Gdx.graphics.getWidth() / 2f - layout.width
/ 2;
            float titleY = playBounds.y + playBounds.height + 80; //
80 píxeles encima del botón superior
            titleFont.draw(batch, "WildWestGame", titleX, titleY);

            // Dibujar botones con texto
            if (playButton != null) {
                batch.draw(playButton, playBounds.x, playBounds.y,
playBounds.width, playBounds.height);
                System.out.println("Dibujando botón de juego");

                // Dibujar texto "INICIAR PARTIDA" dentro del botón
                layout.setText(font, "INICIAR PARTIDA");
                float textX = playBounds.x + (playBounds.width -
layout.width) / 2;
                float textY = playBounds.y + (playBounds.height +
layout.height) / 2;
                font.draw(batch, "INICIAR PARTIDA", textX, textY);
            }

            if (settingsButton != null) {
                batch.draw(settingsButton, settingsBounds.x,
settingsBounds.y, settingsBounds.width, settingsBounds.height);
                System.out.println("Dibujando botón de
configuración");

                // Dibujar texto "CONFIGURACION" dentro del botón
                layout.setText(font, "CONFIGURACION");
                float textX = settingsBounds.x + (settingsBounds.width
- layout.width) / 2;
```

```
                float textY = settingsBounds.y +
(settingsBounds.height + layout.height) / 2;
                font.draw(batch, "CONFIGURACION", textX, textY);
            }

            if (exitButton != null) {
                batch.draw(exitButton, exitBounds.x, exitBounds.y,
exitBounds.width, exitBounds.height);
                System.out.println("Dibujando botón de salida");

                // Dibujar texto "SALIR" dentro del botón
                layout.setText(font, "SALIR");
                float textX = exitBounds.x + (exitBounds.width -
layout.width) / 2;
                float textY = exitBounds.y + (exitBounds.height +
layout.height) / 2;
                font.draw(batch, "SALIR", textX, textY);
            }
        } else {
            System.out.println("menuBackground es null, no se puede
dibujar el menú");
        }
        batch.end();

        // Comprobar clics
        if (Gdx.input.justTouched()) {
            float x = Gdx.input.getX();
            float y = Gdx.graphics.getHeight() - Gdx.input.getY(); //
Invertir Y
            System.out.println("Toque detectado en: (" + x + ", " + y
+ ")");

            if (playBounds.contains(x, y)) {
                System.out.println("Botón de juego presionado");
                startGame();
            } else if (settingsBounds.contains(x, y)) {
                try {
                    System.out.println("Botón de configuración
presionado");

                    if (settingsScreen == null) {
                        settingsScreen = new SettingsScreen(this);
                    }

                    currentScreen = settingsScreen;
                    settingsScreen.show();
                    System.out.println("Transición a configuración
completada");
                } catch (Exception e) {
                    System.err.println("Error en la transición a la
pantalla de configuración: " + e.getMessage());
                    e.printStackTrace();
                    currentScreen = this;
                }
            } else if (exitBounds.contains(x, y)) {
                System.out.println("Botón de salida presionado");
                Gdx.app.exit();
            }
        }
    }
```

```
    private void startGame() {
        // Crear e inicializar la pantalla de juego
        GameScreen gameScreen = new GameScreen(game);

        // Pasar la configuración al GameScreen
        gameScreen.setGameDuration(gameDuration);
        gameScreen.setMusicOn(musicOn);
        gameScreen.setSfxOn(sfxOn);

        System.out.println("Iniciando juego con duración: " +
gameDuration + " segundos, equivale a "
            + (gameDuration/60) + " minutos");
        System.out.println("Música: " + (musicOn ? "ON" : "OFF") + ",
Efectos: " + (sfxOn ? "ON" : "OFF"));

        game.setScreen(gameScreen);
    }

    @Override
    public void resize(int width, int height) {
        // Ajustar botones a la nueva resolución
        float centerX = width / 2f;
        playBounds.x = centerX - 180;
        settingsBounds.x = centerX - 180;
        exitBounds.x = centerX - 180;

        // Propagar resize a la pantalla actual
        if (currentScreen != this) {
            currentScreen.resize(width, height);
        }
    }

    @Override
    public void pause() {
        if (currentScreen != this) {
            currentScreen.pause();
        }
    }

    @Override
    public void resume() {
        if (currentScreen != this) {
            currentScreen.resume();
        }
    }

    @Override
    public void hide() {
        // No es necesario implementar
    }

    @Override
    public void dispose() {
        if (menuBackground != null) menuBackground.dispose();
        if (playButton != null) playButton.dispose();
        if (settingsButton != null) settingsButton.dispose();
        if (exitButton != null) exitButton.dispose();
        if (shapeRenderer != null) shapeRenderer.dispose();
        if (font != null) font.dispose();
        if (titleFont != null) titleFont.dispose();
```

```
        if (settingsScreen != null) {
            settingsScreen.dispose();
        }
        if (controlsScreen != null) {
            controlsScreen.dispose();
        }
    }

    // Método para configurar la duración del juego en minutos
    public void setGameDuration(float minutes) {
        this.gameDuration = minutes; // Almacenar en minutos como
entero
    }

    public float getGameDuration() {
        return gameDuration;
    }
    public void returnToMenu() {
        currentScreen = this;
    }

    // Para compartir el SpriteBatch
    public SpriteBatch getBatch() {
        return batch;
    }

    // Para obtener la referencia al juego principal
    public WildWestGame getGame() {
        return game;
    }

    public boolean isMusicOn() {
        return musicOn;
    }

    public void setMusicOn(boolean musicOn) {
        this.musicOn = musicOn;
        // Aquí podrías, por ejemplo, detener la música global o
reanudarla
        // si tu juego principal la maneja de forma centralizada.
    }

    public boolean isSfxOn() {
        return sfxOn;
    }

    public void setSfxOn(boolean sfxOn) {
        this.sfxOn = sfxOn;
        // Similarmente, aquí podrías establecer un volumen de 0
        // para los efectos o reactivarlos.
    }

    // --------------------------
    // PANTALLA DE CONTROLES
    // --------------------------
    /**
     * Muestra la pantalla de controles.
     * (Necesitarías implementar ControlsScreen y su lógica)
     */
    public void showControlsScreen() {
        controlsScreen = new ControlScreen(this);
```

```
        currentScreen = controlsScreen;
        controlsScreen.show();
    }

    /**
     * Obtiene la referencia a la pantalla de configuración.
     */
    public SettingsScreen getSettingsScreen() {
        return settingsScreen;
    }

    /**
     * Establece la pantalla actual.
     */
    public void setCurrentScreen(Screen screen) {
        this.currentScreen = screen;
    }
}
```

SettingsScreen y ControlScreen que se encarga de la parte de configuración:

```
package io.github.JuegoProyecto;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import java.awt.Rectangle;

public class ControlScreen implements Screen {

    private MenuScreen menuScreen;
    private SpriteBatch batch;

    private Texture background;
    private Texture buttonTexture;
    private BitmapFont font;
    private BitmapFont titleFont;

    // Botón para volver
    private com.badlogic.gdx.math.Rectangle backButton;

    /**
     * Constructor: recibe la referencia al MenuScreen para poder
     * volver al menú de configuración.
     */
    public ControlScreen(MenuScreen menuScreen) {
        this.menuScreen = menuScreen;
        this.batch = menuScreen.getBatch();
    }

    @Override
    public void show() {
        try {
            System.out.println("Mostrando pantalla de controles...");
```

```
            // Cargar texturas con manejo de errores
            try {
                background = new
Texture(Gdx.files.internal("fondo.jpg"));
                System.out.println("Fondo de controles cargado
correctamente");
            } catch (Exception e) {
                System.err.println("Error cargando fondo.jpg: " +
e.getMessage());
                // Crear una textura de fallback
                Pixmap fallbackPixmap = new Pixmap(1, 1,
Pixmap.Format.RGBA8888);
                fallbackPixmap.setColor(Color.BROWN);
                fallbackPixmap.fill();
                background = new Texture(fallbackPixmap);
                fallbackPixmap.dispose();
            }

            try {
                buttonTexture = new
Texture(Gdx.files.internal("configuracion.png"));
                System.out.println("Textura de botón cargada
correctamente");
            } catch (Exception e) {
                System.err.println("Error cargando configuracion.png:
" + e.getMessage());
                // Crear una textura de fallback
                Pixmap fallbackPixmap = new Pixmap(1, 1,
Pixmap.Format.RGBA8888);
                fallbackPixmap.setColor(Color.ORANGE);
                fallbackPixmap.fill();
                buttonTexture = new Texture(fallbackPixmap);
                fallbackPixmap.dispose();
            }

            // Inicializar fuentes
            font = new BitmapFont();
            font.setColor(Color.WHITE);
            font.getData().setScale(1.5f);

            titleFont = new BitmapFont();
            titleFont.setColor(Color.WHITE);
            titleFont.getData().setScale(2.5f);

            // Calcular posición central
            float centerX = Gdx.graphics.getWidth() / 2f;

            // Botón para volver
            backButton = new com.badlogic.gdx.math.Rectangle(centerX -
100, 50, 200, 50);

        } catch (Exception e) {
            System.err.println("Error en ControlsScreen.show(): " +
e.getMessage());
            e.printStackTrace();
        }
    }

    @Override
    public void render(float delta) {
```

```
        // Limpiar la pantalla
        Gdx.gl.glClearColor(0.1f, 0.1f, 0.1f, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        batch.begin();

        // Dibujar fondo
        if (background != null) {
            batch.draw(background, 0, 0, Gdx.graphics.getWidth(),
Gdx.graphics.getHeight());
        }

        // Título de la pantalla
        titleFont.draw(batch, "CONTROLES DEL JUEGO",
            Gdx.graphics.getWidth() / 2f - 150,
            Gdx.graphics.getHeight() - 60);

        // Información de controles
        float startY = Gdx.graphics.getHeight() - 120;
        float lineHeight = 40; // Espacio entre líneas
        float leftColumnX = 100;
        float rightColumnX = Gdx.graphics.getWidth() / 2f + 50;

        // Cabecera de jugadores
        font.draw(batch, "SHERIFF (JUGADOR 1)", leftColumnX, startY);
        font.draw(batch, "BANDIDO (JUGADOR 2)", rightColumnX, startY);

        // Controles de movimiento
        font.draw(batch, "Moverse izquierda: A", leftColumnX, startY -
lineHeight);
        font.draw(batch, "Moverse derecha: D", leftColumnX, startY -
lineHeight * 2);
        font.draw(batch, "Disparar: W", leftColumnX, startY -
lineHeight * 3);

        font.draw(batch, "Moverse izquierda: FLECHA IZQUIERDA",
rightColumnX, startY - lineHeight);
        font.draw(batch, "Moverse derecha: FLECHA DERECHA",
rightColumnX, startY - lineHeight * 2);
        font.draw(batch, "Disparar: FLECHA ARRIBA", rightColumnX,
startY - lineHeight * 3);

        // Información de puntuación
        float scoringY = startY - lineHeight * 5;
        titleFont.draw(batch, "PUNTUACIÓN", Gdx.graphics.getWidth() /
2f - 100, scoringY);

        font.draw(batch, "Águila: 3 puntos", leftColumnX, scoringY -
lineHeight);
        font.draw(batch, "Caballo: 2 puntos", leftColumnX, scoringY -
lineHeight * 2);
        font.draw(batch, "Buey: 1 punto", leftColumnX, scoringY -
lineHeight * 3);

        // Información adicional
        float infoY = scoringY - lineHeight * 5;
        titleFont.draw(batch, "INFORMACIÓN GENERAL",
Gdx.graphics.getWidth() / 2f - 150, infoY);

        font.draw(batch, "Volver al menú: ESC",
Gdx.graphics.getWidth() / 2f - 100, infoY - lineHeight);
```

```
        font.draw(batch, "Gana el jugador con más puntos al finalizar
el tiempo",
            Gdx.graphics.getWidth() / 2f - 250, infoY - lineHeight *
2);

        // Dibujar botón VOLVER
        if (buttonTexture != null) {
            batch.draw(buttonTexture, backButton.x, backButton.y,
backButton.width, backButton.height);
        }
        font.draw(batch, "VOLVER", backButton.x + 60, backButton.y +
35);

        batch.end();

        // Manejo de clicks
        if (Gdx.input.justTouched()) {
            float x = Gdx.input.getX();
            float y = Gdx.graphics.getHeight() - Gdx.input.getY();

            // Botón VOLVER
            if (backButton.contains(x, y)) {
                System.out.println("Volviendo a la pantalla de
configuración");
                // Cambiamos la pantalla actual en MenuScreen a
SettingsScreen
                if (menuScreen.getSettingsScreen() != null) {

menuScreen.setCurrentScreen(menuScreen.getSettingsScreen());
                } else {
                    // Si por alguna razón SettingsScreen es null,
volvemos al menú principal
                    menuScreen.returnToMenu();
                }
            }
        }

        // También permitir volver con ESC
        if (Gdx.input.isKeyJustPressed(Input.Keys.ESCAPE)) {
            System.out.println("Volviendo a la pantalla de
configuración (ESC)");
            // Mismo comportamiento que el botón VOLVER
            if (menuScreen.getSettingsScreen() != null) {

menuScreen.setCurrentScreen(menuScreen.getSettingsScreen());
            } else {
                menuScreen.returnToMenu();
            }
        }
    }

    @Override
    public void resize(int width, int height) {
        // Ajustar botón a la nueva resolución
        float centerX = width / 2f;
        backButton.x = centerX - 100;
    }

    @Override
    public void pause() {
        // Método requerido por la interfaz Screen
```

```
    }

    @Override
    public void resume() {
        // Método requerido por la interfaz Screen
    }

    @Override
    public void hide() {
        // Método requerido por la interfaz Screen
    }

    @Override
    public void dispose() {
        if (background != null) background.dispose();
        if (buttonTexture != null) buttonTexture.dispose();
        if (font != null) font.dispose();
        if (titleFont != null) titleFont.dispose();
    }
}
```

SettingsScreen:

```
package io.github.JuegoProyecto;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.math.Rectangle;

public class SettingsScreen implements Screen {

    private MenuScreen menuScreen;
    private SpriteBatch batch;

    private Texture background;
    private Texture buttonTexture;
    private BitmapFont font;

    // Botones para el tiempo de partida
    private Rectangle time30SecButton;
    private Rectangle time2MinButton;
    private Rectangle time3MinButton;
    private Rectangle time5MinButton;

    // Botones para volumen (música y efectos)
    private Rectangle musicToggleButton;
    private Rectangle sfxToggleButton;

    // Botón para abrir pantalla de Controles
    private Rectangle controlsButton;

    // Botón para volver al menú
    private Rectangle backButton;
```

```
    // Estados de configuración
    private float selectedDuration;
    private boolean musicOn;
    private boolean sfxOn;

    /**
     * Constructor: recibe la referencia al MenuScreen para poder
     * leer y modificar la configuración (duración, música, efectos,
etc.).
     */
    public SettingsScreen(MenuScreen menuScreen) {
        this.menuScreen = menuScreen;
        this.batch = menuScreen.getBatch();

        // Tomar valores actuales desde el menuScreen
        this.selectedDuration = menuScreen.getGameDuration(); // p.e.
0.5f, 2f, 3f o 5f
        this.musicOn = menuScreen.isMusicOn();
        this.sfxOn   = menuScreen.isSfxOn();
    }

    @Override
    public void show() {
        try {
            // Log which files we're trying to load
            System.out.println("Intentando cargar textura de
fondo...");

            try {
                background = new
Texture(Gdx.files.internal("fondo.jpg"));
                System.out.println("Fondo cargado correctamente");
            } catch (Exception e) {
                System.err.println("Error cargando fondo: " +
e.getMessage());
                // Create a fallback texture
                Pixmap fallbackPixmap = new Pixmap(1, 1,
Pixmap.Format.RGBA8888);
                fallbackPixmap.setColor(Color.BROWN);
                fallbackPixmap.fill();
                background = new Texture(fallbackPixmap);
                fallbackPixmap.dispose();
            }

            System.out.println("Intentando cargar textura de
botón...");
            try {
                buttonTexture = new
Texture(Gdx.files.internal("configuracion.png"));
                System.out.println("Textura de botón cargada
correctamente");
            } catch (Exception e) {
                System.err.println("Error cargando configuracion.png:
" + e.getMessage());
                // Create a fallback texture
                Pixmap fallbackPixmap = new Pixmap(1, 1,
Pixmap.Format.RGBA8888);
                fallbackPixmap.setColor(Color.ORANGE);
                fallbackPixmap.fill();
                buttonTexture = new Texture(fallbackPixmap);
```

```
                fallbackPixmap.dispose();
            }

            // Fuente para texto con manejo de errores
            try {
                font = new BitmapFont();
                font.setColor(Color.WHITE);
                font.getData().setScale(2f);
                System.out.println("Fuente cargada correctamente");
            } catch (Exception e) {
                System.err.println("Error cargando fuente: " +
e.getMessage());
                font = new BitmapFont();
            }

            // Calcular posición central
            float centerX = Gdx.graphics.getWidth() / 2f;

            // Botones de tiempo de partida
            time30SecButton = new Rectangle(centerX - 260, 350, 120,
50);
            time2MinButton  = new Rectangle(centerX - 130, 350, 120,
50);
            time3MinButton  = new Rectangle(centerX,        350, 120,
50);
            time5MinButton  = new Rectangle(centerX + 130,  350, 120,
50);

            // Botones de música y efectos
            musicToggleButton = new Rectangle(centerX - 200, 250, 180,
50);
            sfxToggleButton   = new Rectangle(centerX +  20, 250, 180,
50);

            // Botón para Controles
            controlsButton = new Rectangle(centerX - 100, 150, 200,
50);

            // Botón para volver al menú principal
            backButton = new Rectangle(centerX - 100,  50, 200, 50);

            System.out.println("SettingsScreen mostrada. Duración: " +
selectedDuration
                + " | Música: " + musicOn + " | Efectos: " + sfxOn);
        } catch (Exception e) {
            System.err.println("Error general en
SettingsScreen.show(): " + e.getMessage());
            e.printStackTrace();
        }
    }
    @Override
    public void render(float delta) {
        // Limpiar la pantalla
        Gdx.gl.glClearColor(0.1f, 0.1f, 0.1f, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        batch.begin();

        // Dibujar fondo
        if (background != null) {
            batch.draw(background, 0, 0, Gdx.graphics.getWidth(),
```

```
Gdx.graphics.getHeight());
        }

        // Dibujar los botones (usando la misma textura para todos)
        if (buttonTexture != null) {
            // Botones de tiempo
            batch.draw(buttonTexture, time30SecButton.x,
time30SecButton.y,
                time30SecButton.width, time30SecButton.height);
            batch.draw(buttonTexture, time2MinButton.x,
time2MinButton.y,
                time2MinButton.width, time2MinButton.height);
            batch.draw(buttonTexture, time3MinButton.x,
time3MinButton.y,
                time3MinButton.width, time3MinButton.height);
            batch.draw(buttonTexture, time5MinButton.x,
time5MinButton.y,
                time5MinButton.width, time5MinButton.height);

            // Música y Efectos
            batch.draw(buttonTexture, musicToggleButton.x,
musicToggleButton.y,
                musicToggleButton.width, musicToggleButton.height);
            batch.draw(buttonTexture, sfxToggleButton.x,
sfxToggleButton.y,
                sfxToggleButton.width, sfxToggleButton.height);

            // Controles
            batch.draw(buttonTexture, controlsButton.x,
controlsButton.y,
                controlsButton.width, controlsButton.height);

            // Volver
            batch.draw(buttonTexture, backButton.x, backButton.y,
                backButton.width, backButton.height);
        }

        // Título de la pantalla
        font.draw(batch, "CONFIGURACIÓN",
            Gdx.graphics.getWidth() / 2f - 100,
            Gdx.graphics.getHeight() - 60);

        // Etiqueta "Tiempo de partida"
        font.draw(batch, "TIEMPO DE PARTIDA:",
            Gdx.graphics.getWidth()/2 - 200, 420);

        // Resaltar la duración seleccionada
        // 30 SEG
        if (Math.abs(selectedDuration - 0.5f) < 0.01f)
font.setColor(Color.YELLOW);
        else font.setColor(Color.WHITE);
        font.draw(batch, "30 SEG", time30SecButton.x + 10,
time30SecButton.y + 35);

        // 2 MIN
        if (Math.abs(selectedDuration - 2f) < 0.01f)
font.setColor(Color.YELLOW);
        else font.setColor(Color.WHITE);
        font.draw(batch, "2 MIN", time2MinButton.x + 15,
time2MinButton.y + 35);
```

```
        // 3 MIN
        if (Math.abs(selectedDuration - 3f) < 0.01f)
font.setColor(Color.YELLOW);
        else font.setColor(Color.WHITE);
        font.draw(batch, "3 MIN", time3MinButton.x + 15,
time3MinButton.y + 35);

        // 5 MIN
        if (Math.abs(selectedDuration - 5f) < 0.01f)
font.setColor(Color.YELLOW);
        else font.setColor(Color.WHITE);
        font.draw(batch, "5 MIN", time5MinButton.x + 15,
time5MinButton.y + 35);

        // Música y Efectos (resaltar si están ON)
        if (musicOn) font.setColor(Color.YELLOW); else
font.setColor(Color.WHITE);
        font.draw(batch, "MUSICA: " + (musicOn ? "ON" : "OFF"),
            musicToggleButton.x + 10, musicToggleButton.y + 35);

        if (sfxOn) font.setColor(Color.YELLOW); else
font.setColor(Color.WHITE);
        font.draw(batch, "EFECTOS: " + (sfxOn ? "ON" : "OFF"),
            sfxToggleButton.x + 10, sfxToggleButton.y + 35);

        // Controles
        font.setColor(Color.WHITE);
        font.draw(batch, "CONTROLES", controlsButton.x + 20,
controlsButton.y + 35);

        // Volver
        font.draw(batch, "VOLVER", backButton.x + 50, backButton.y +
35);

        batch.end();

        // Manejo de clicks
        if (Gdx.input.justTouched()) {
            float x = Gdx.input.getX();
            float y = Gdx.graphics.getHeight() - Gdx.input.getY();

            // Ajustar tiempo de partida
            if (time30SecButton.contains(x, y)) {
                selectedDuration = 0.5f;
                menuScreen.setGameDuration(0.5f);
                System.out.println("Duración configurada a 30 seg");
            } else if (time2MinButton.contains(x, y)) {
                selectedDuration = 2f;
                menuScreen.setGameDuration(2f);
                System.out.println("Duración configurada a 2
minutos");
            } else if (time3MinButton.contains(x, y)) {
                selectedDuration = 3f;
                menuScreen.setGameDuration(3f);
                System.out.println("Duración configurada a 3
minutos");
            } else if (time5MinButton.contains(x, y)) {
                selectedDuration = 5f;
                menuScreen.setGameDuration(5f);
                System.out.println("Duración configurada a 5
minutos");
```

```
            }
            // Música ON/OFF
            else if (musicToggleButton.contains(x, y)) {
                musicOn = !musicOn;
                menuScreen.setMusicOn(musicOn);
                System.out.println("Música ahora está: " + (musicOn ?
"ON" : "OFF"));
            }
            // Efectos ON/OFF
            else if (sfxToggleButton.contains(x, y)) {
                sfxOn = !sfxOn;
                menuScreen.setSfxOn(sfxOn);
                System.out.println("Efectos ahora están: " + (sfxOn ?
"ON" : "OFF"));
            }
            // Botón CONTROLES
            else if (controlsButton.contains(x, y)) {
                System.out.println("Abriendo pantalla de Controles");
                menuScreen.showControlsScreen();
            }
            // Botón VOLVER
            else if (backButton.contains(x, y)) {
                System.out.println("Volviendo al menú principal");
                menuScreen.returnToMenu();
            }
        }
    }

    @Override
    public void resize(int width, int height) {
        float centerX = width / 2f;

        // Reposicionar cada botón al cambiar la resolución
        time30SecButton.x = centerX - 260;
        time2MinButton.x  = centerX - 130;
        time3MinButton.x  = centerX;
        time5MinButton.x  = centerX + 130;

        musicToggleButton.x = centerX - 200;
        sfxToggleButton.x   = centerX + 20;

        controlsButton.x = centerX - 100;
        backButton.x     = centerX - 100;
    }

    @Override
    public void pause() { }

    @Override
    public void resume() { }

    @Override
    public void hide() { }

    /** Libera recursos gráficos */
    @Override
    public void dispose() {
        if (background != null) background.dispose();
        if (buttonTexture != null) buttonTexture.dispose();
        if (font != null) font.dispose();
```

```
    }
}
```

La opción de iniciar partida:

```
package io.github.JuegoProyecto;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.Pixmap;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.GlyphLayout;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.math.Rectangle;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.utils.ScreenUtils;
import com.badlogic.gdx.utils.viewport.FitViewport;
import com.badlogic.gdx.utils.viewport.Viewport;
import com.badlogic.gdx.graphics.OrthographicCamera;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Random;
import com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator;
import
com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator.FreeTypeF
ontParameter;

public class GameScreen implements Screen {
    private WildWestGame game;
    private SpriteBatch batch;
    private Texture image;
    private Texture jugador1, jugador2;
    private Texture aguilaTexture, caballoTexture, bueyTexture;
    private Viewport viewport;
    private OrthographicCamera camera;
    private BitmapFont fuenteOeste;
    private GlyphLayout layout;
    private Texture hudFondo;
    private boolean mostrarMensaje = false;
    private String mensajeGanador = "";
    private float tiempoMensaje = 0;
    private final float DURACION_MENSAJE = 2f;
    private float gameDuration = 180;//valor predeterminado que luego
suscribira menuscreen
    private float gameTimer = 0;
    private boolean gameTimeOver = false;
    private int puntosJugador1 = 0;
    private int puntosJugador2 = 0;
    private Music musicadefondopartida;
    private Sound disparoSonido1, disparoSonido2, sonidoPunto;
    private static final float VIRTUAL_WIDTH = 1416;
    private static final float VIRTUAL_HEIGHT = 582;

    private float nivelSuelo = 95;
    private static final float SUELO_Y = 95;
```

```
    private Vector2 posJugador1, posJugador2;
    private float velocidadJugador = 300f;

    // Clase Animal sin cambios...
    private class Animal {
        Rectangle bounds;
        float frameTime = 0;
        int currentFrame = 0;
        float velocidad;
        int totalFrames;
        boolean visible = true;

        public Animal(float x, float y, float width, float height,
float velocidad, int totalFrames) {
            this.bounds = new Rectangle(x, y, width, height);
            this.velocidad = velocidad;
            this.totalFrames = totalFrames;
        }

        public void update(float delta) {
            // Solo actualizar si el juego no ha terminado
            if (!gameTimeOver) {
                bounds.x -= velocidad * delta;
                frameTime += delta;
                if (frameTime >= 0.1f) {
                    frameTime = 0;
                    currentFrame = (currentFrame + 1) % totalFrames;
                }
                if (bounds.x < -bounds.width) {
                    visible = false;
                }
            }
        }

        public int getFrame() {
            return currentFrame;
        }
    }

    private ArrayList<Animal> aguilas;
    private ArrayList<Animal> caballos;
    private ArrayList<Animal> bueyes;

    private float tiempoDesdeUltimaAguila = 0;
    private float tiempoDesdeUltimoCaballo = 0;
    private float tiempoDesdeUltimobuey = 0;

    private float intervaloAparicionAguila = 4f;
    private float intervaloAparicionCaballo = 6f;
    private float intervaloAparicionBuey = 8f;

    private static final int AGUILA_FRAMES = 2;
    private static final boolean AGUILA_FRAMES_VERTICALES = true;
    private static final int CABALLO_FRAMES = 8;
    private static final int BUEY_FRAMES = 7;

    public GameScreen(WildWestGame game) {
        this.game = game;
        this.batch = game.getBatch();
        this.layout = new GlyphLayout();
```

```
    }

    @Override
    public void show() {
        try {
            // Cargar texturas con manejo de errores
            try {
                image = new Texture("fondo.jpg");
                System.out.println("Fondo cargado correctamente");
            } catch (Exception e) {
                System.err.println("Error al cargar fondo.jpg: " +
e.getMessage());
            }

            try {
                jugador1 = new Texture("jug1.png");
                jugador2 = new Texture("jug2.png");
                System.out.println("Jugadores cargados
correctamente");
            } catch (Exception e) {
                System.err.println("Error al cargar jugadores: " +
e.getMessage());
            }

            try {
                aguilaTexture = new Texture("animal1.png");
                caballoTexture = new Texture("animal2.png");
                bueyTexture = new Texture("animal3.png");
                System.out.println("Animales cargados correctamente");
            } catch (Exception e) {
                System.err.println("Error al cargar animales: " +
e.getMessage());
            }

            try {
                disparoSonido1 =
Gdx.audio.newSound(Gdx.files.internal("disparojug1.mp3"));
                disparoSonido2 =
Gdx.audio.newSound(Gdx.files.internal("disparojug2.mp3"));
                sonidoPunto =
Gdx.audio.newSound(Gdx.files.internal("ganasteunpunto.mp3"));
                System.out.println("Sonidos cargados correctamente");
            } catch (Exception e) {
                System.err.println("Error al cargar sonidos: " +
e.getMessage());
            }

            try {
                hudFondo = new Texture("hudfondo.png");
            } catch (Exception e) {
                System.err.println("No se pudo cargar hudfondo.png: "
+ e.getMessage());
                // Crear una textura de 1x1 píxel blanco como fallback
                Pixmap pixmap = new Pixmap(1, 1,
Pixmap.Format.RGBA8888);
                pixmap.setColor(Color.WHITE);
                pixmap.fill();
                hudFondo = new Texture(pixmap);
                pixmap.dispose();
            }
```

```
            try {
                musicadefondopartida =
Gdx.audio.newMusic(Gdx.files.internal("musicadefondo.mp3"));
                musicadefondopartida.setLooping(true); // Que suene en
bucle

                // Solo reproducir si musicOn está activado
                if (musicOn) {
                    musicadefondopartida.play();
                    System.out.println("Música de fondo cargada y
reproduciéndose");
                } else {
                    System.out.println("Música de fondo cargada pero
en silencio (OFF)");
                }
            } catch (Exception e) {
                System.err.println("Error al cargar la música de
fondo: " + e.getMessage());
            }


            camera = new OrthographicCamera();
            viewport = new FitViewport(VIRTUAL_WIDTH, VIRTUAL_HEIGHT,
camera);
            camera.position.set(VIRTUAL_WIDTH / 2f, VIRTUAL_HEIGHT /
2f, 0);
            camera.update();

            // Colocar jugadores al nivel del suelo
            posJugador1 = new Vector2(50, nivelSuelo);
            posJugador2 = new Vector2(60, nivelSuelo);

            // Inicializar las listas de animales
            aguilas = new ArrayList<>();
            caballos = new ArrayList<>();
            bueyes = new ArrayList<>();

            // Inicializar la fuente
            inicializarFuenteEstilizada();

            // Reiniciar marcadores
            puntosJugador1 = 0;
            puntosJugador2 = 0;
            gameTimer = 0;
            gameTimeOver = false;

            System.out.println("GameScreen show() completado
correctamente");
        } catch (Exception e) {
            System.err.println("Error general en show(): " +
e.getMessage());
            e.printStackTrace();
        }
    }

    // Resto de métodos de generación y animación de animales sin
cambios...
    private void generarAguila() {
        // No generar nuevos animales si el juego ha terminado
        if (gameTimeOver) return;
```

```
        Random rand = new Random();
        float y = SUELO_Y + rand.nextFloat() * 40 + 10;
        float width = 100;
        float height = 100;

        Animal aguila = new Animal(VIRTUAL_WIDTH, y, width, height,
150f, AGUILA_FRAMES);
        aguilas.add(aguila);
        System.out.println("Águila generada en posición (" +
VIRTUAL_WIDTH + ", " + y + ")");
    }

    private void generarCaballo() {
        // No generar nuevos animales si el juego ha terminado
        if (gameTimeOver) return;

        float posicionYCaballo = SUELO_Y - 20;
        Animal caballo = new Animal(VIRTUAL_WIDTH, posicionYCaballo,
150, 100, 100f, CABALLO_FRAMES);
        caballos.add(caballo);
        System.out.println("Caballo generado en posición (" +
VIRTUAL_WIDTH + ", " + posicionYCaballo + ")");
    }

    private void generarBuey() {
        // No generar nuevos animales si el juego ha terminado
        if (gameTimeOver) return;

        Animal buey = new Animal(VIRTUAL_WIDTH, SUELO_Y, 150, 100,
80f, BUEY_FRAMES);
        bueyes.add(buey);
        System.out.println("Buey generado en posición (" +
VIRTUAL_WIDTH + ", " + SUELO_Y + ")");
    }

    private void dibujarAguilas() {
        if (aguilaTexture != null) {
            int frameWidth, frameHeight;

            if (AGUILA_FRAMES_VERTICALES) {
                frameWidth = aguilaTexture.getWidth();
                frameHeight = aguilaTexture.getHeight() /
AGUILA_FRAMES;

                for (Animal aguila : aguilas) {
                    if (aguila.visible) {
                        int frameY = aguila.getFrame() * frameHeight;
                        batch.draw(aguilaTexture,
                            aguila.bounds.x, aguila.bounds.y,
                            aguila.bounds.width, aguila.bounds.height,
                            0, frameY,
                            frameWidth, frameHeight,
                            false, false);
                    }
                }
            } else {
                frameWidth = aguilaTexture.getWidth() / AGUILA_FRAMES;
                frameHeight = aguilaTexture.getHeight();

                for (Animal aguila : aguilas) {
                    if (aguila.visible) {
```

```
                        int frameX = aguila.getFrame() * frameWidth;
                        batch.draw(aguilaTexture,
                            aguila.bounds.x, aguila.bounds.y,
                            aguila.bounds.width, aguila.bounds.height,
                            frameX, 0,
                            frameWidth, frameHeight,
                            false, false);
                    }
                }
            }
        }
    }

    private void inicializarFuenteEstilizada() {
        try {
            FreeTypeFontGenerator generator = new
FreeTypeFontGenerator(Gdx.files.internal("western_font.ttf"));
            FreeTypeFontParameter parameter = new
FreeTypeFontParameter();
            parameter.size = 24;
            parameter.color = new Color(0.9f, 0.8f, 0.2f, 1);
            parameter.borderWidth = 2;
            parameter.borderColor = new Color(0.3f, 0.1f, 0, 1);
            parameter.shadowOffsetX = 2;
            parameter.shadowOffsetY = 2;
            parameter.shadowColor = new Color(0, 0, 0, 0.75f);
            fuenteOeste = generator.generateFont(parameter);
            generator.dispose();
        } catch (Exception e) {
            fuenteOeste = new BitmapFont();
            fuenteOeste.setColor(new Color(0.9f, 0.8f, 0.2f, 1));
            fuenteOeste.getData().setScale(1.5f);
        }
    }

    private void dibujarHUDMejorado() {
        // Dibujar fondos para las puntuaciones (opcional)
        if (hudFondo != null) {
            batch.draw(hudFondo, 10, VIRTUAL_HEIGHT - 50, 200, 40); //
Fondo para jugador 1
            batch.draw(hudFondo, VIRTUAL_WIDTH - 210, VIRTUAL_HEIGHT -
50, 200, 40); // Fondo para jugador 2
        }

        // Puntuación del jugador 1 (esquina izquierda)
        String textoJugador1 = "SHERIFF: " + puntosJugador1;
        fuenteOeste.draw(batch, textoJugador1, 20, VIRTUAL_HEIGHT -
20);

        // Puntuación del jugador 2 (esquina derecha)
        String textoJugador2 = "BANDIDO: " + puntosJugador2;
        layout.setText(fuenteOeste, textoJugador2);
        fuenteOeste.draw(batch, textoJugador2, VIRTUAL_WIDTH -
layout.width - 20, VIRTUAL_HEIGHT - 20);

        // Mostrar mensaje cuando un jugador alcanza cierta puntuación
o gana
        if (mostrarMensaje) {
            layout.setText(fuenteOeste, mensajeGanador);
            float x = (VIRTUAL_WIDTH - layout.width) / 2;
            float y = VIRTUAL_HEIGHT / 2;
```

```
            // Dibujar un fondo para el mensaje (opcional)
            batch.setColor(0.3f, 0.1f, 0, 0.7f); // Color marrón
semitransparente
            batch.draw(hudFondo, x - 20, y - layout.height - 10,
layout.width + 40, layout.height + 20);
            batch.setColor(Color.WHITE); // Restaurar color

            fuenteOeste.draw(batch, mensajeGanador, x, y);
        }

        // Si el tiempo se ha agotado, mostrar un mensaje permanente
        if (gameTimeOver) {
            // Crear un mensaje más destacado para el final del juego
            String mensajeFinal;
            if (puntosJugador1 > puntosJugador2) {
                mensajeFinal = "¡TIEMPO AGOTADO! ¡EL SHERIFF GANA!";
            } else if (puntosJugador2 > puntosJugador1) {
                mensajeFinal = "¡TIEMPO AGOTADO! ¡EL BANDIDO GANA!";
            } else {
                mensajeFinal = "¡TIEMPO AGOTADO! ¡EMPATE!";
            }

            layout.setText(fuenteOeste, mensajeFinal);
            float x = (VIRTUAL_WIDTH - layout.width) / 2;
            float y = VIRTUAL_HEIGHT / 2 + 50;

            // Fondo más grande y destacado para el mensaje final
            batch.setColor(0.3f, 0.1f, 0, 0.8f);
            batch.draw(hudFondo, x - 30, y - layout.height - 20,
layout.width + 60, layout.height + 40);
            batch.setColor(Color.WHITE);

            fuenteOeste.draw(batch, mensajeFinal, x, y);

            // Instrucciones para volver al menú
            String volverTexto = "Presiona ESC para volver al menú";
            layout.setText(fuenteOeste, volverTexto);
            float volverX = (VIRTUAL_WIDTH - layout.width) / 2;
            float volverY = y - 40;
            fuenteOeste.draw(batch, volverTexto, volverX, volverY);
        }
    }

    // Método para mostrar un mensaje temporal cuando ocurre algo
importante
    public void mostrarMensajeTemporal(String mensaje) {
        mensajeGanador = mensaje;
        mostrarMensaje = true;
        tiempoMensaje = 0;
    }

    @Override
    public void render(float delta) {
        try {
            // Actualizar temporizador de juego solo si no ha
terminado
            if (!gameTimeOver) {
                gameTimer += delta;
                if (gameTimer >= gameDuration) {
                    gameTimeOver = true;
```

```
                    mostrarMensaje = false; // Asegurarnos de que no
se muestren mensajes temporales

                    // Parar la música de fondo
                    if (musicadefondopartida != null &&
musicadefondopartida.isPlaying()) {
                        musicadefondopartida.stop();
                    }

                    System.out.println("¡Tiempo de juego agotado! La
pantalla se ha congelado.");
                }
            }

            // Color de fondo para que coincida con tu escenario del
viejo oeste
            ScreenUtils.clear(0.7f, 0.5f, 0.3f, 1f);

            viewport.apply();
            batch.setProjectionMatrix(camera.combined);

            // Solo actualizar la lógica del juego si no ha terminado
            if (!gameTimeOver) {
                // Actualizar temporizadores para la generación de
animales
                tiempoDesdeUltimaAguila += delta;
                tiempoDesdeUltimoCaballo += delta;
                tiempoDesdeUltimobuey += delta;

                // Generar nuevos animales si ha pasado suficiente
tiempo
                if (tiempoDesdeUltimaAguila >=
intervaloAparicionAguila) {
                    generarAguila();
                    tiempoDesdeUltimaAguila = 0;
                }

                if (tiempoDesdeUltimoCaballo >=
intervaloAparicionCaballo) {
                    generarCaballo();
                    tiempoDesdeUltimoCaballo = 0;
                }

                if (tiempoDesdeUltimobuey >= intervaloAparicionBuey) {
                    generarBuey();
                    tiempoDesdeUltimobuey = 0;
                }

                // Actualizar la posición y animación de todos los
animales
                actualizarAnimales(delta);

                // Mover jugadores solo si el juego no ha terminado
                moverJugadores();

                // Actualizar el tiempo del mensaje
                if (mostrarMensaje) {
                    tiempoMensaje += delta;
                    if (tiempoMensaje > DURACION_MENSAJE) {
                        mostrarMensaje = false;
                    }
```

```
                }

                // Comprobar si algún jugador ha ganado (alcanzado
cierta puntuación)
                if (puntosJugador1 >= 20) {
                    gameTimeOver = true; // Congelar la pantalla
cuando alguien gana
                    // No llamamos a mostrarMensajeTemporal para
evitar duplicación
                } else if (puntosJugador2 >= 20) {
                    gameTimeOver = true; // Congelar la pantalla
cuando alguien gana
                    // No llamamos a mostrarMensajeTemporal para
evitar duplicación
                }
            }

            batch.begin();

            // Dibuja el fondo
            if (image != null) {
                batch.draw(image, 0, 0, VIRTUAL_WIDTH,
VIRTUAL_HEIGHT);
            }

            // Dibuja primero los jugadores para que estén detrás de
los animales
            if (jugador1 != null && jugador2 != null) {
                batch.draw(jugador1, posJugador1.x, posJugador1.y,
100, 100);
                batch.draw(jugador2, posJugador2.x, posJugador2.y,
100, 100);
            }

            dibujarAguilas();

            // Dibuja caballos con frames de animación
            if (caballoTexture != null) {
                int frameWidth = caballoTexture.getWidth() /
CABALLO_FRAMES;
                int frameHeight = caballoTexture.getHeight();

                for (Animal caballo : caballos) {
                    if (caballo.visible) {
                        int frameX = caballo.getFrame() * frameWidth;
                        batch.draw(caballoTexture,
                            caballo.bounds.x, caballo.bounds.y,
                            caballo.bounds.width,
caballo.bounds.height,
                            frameX, 0,
                            frameWidth, frameHeight,
                            false, false);
                    }
                }
            }

            // Dibuja bueyes con frames de animación
            if (bueyTexture != null) {
                int frameWidth = bueyTexture.getWidth() / BUEY_FRAMES;
                int frameHeight = bueyTexture.getHeight();
```

```
                for (Animal buey : bueyes) {
                    if (buey.visible) {
                        int frameX = buey.getFrame() * frameWidth;
                        batch.draw(bueyTexture,
                            buey.bounds.x, buey.bounds.y,
                            buey.bounds.width, buey.bounds.height,
                            frameX, 0,
                            frameWidth, frameHeight,
                            false, false);
                    }
                }
            }

            // Dibujar HUD
            dibujarHUDMejorado();

            // Dibujar temporizador
            if (!gameTimeOver) {
                int remainingSeconds = (int)(gameDuration -
gameTimer);
                int minutes = remainingSeconds / 60;
                int seconds = remainingSeconds % 60;
                String timeStr = String.format("%02d:%02d", minutes,
seconds);

                // Usar tu fuente existente
                fuenteOeste.draw(batch, "TIEMPO: " + timeStr,
VIRTUAL_WIDTH/2 - 100, VIRTUAL_HEIGHT - 20);
            } else {
                // Mostrar tiempo agotado
                fuenteOeste.draw(batch, "TIEMPO: 00:00",
VIRTUAL_WIDTH/2 - 100, VIRTUAL_HEIGHT - 20);
            }

            batch.end();

            // Limpiar los animales que ya no son visibles (salieron
de la pantalla)
            limpiarAnimales();

            // Permitir volver al menú principal con la tecla Escape
            if (Gdx.input.isKeyJustPressed(Input.Keys.ESCAPE)) {
                game.setScreen(new MenuScreen(game));
                dispose();
            }

        } catch (Exception e) {
            System.err.println("Error en render(): " +
e.getMessage());
            e.printStackTrace();
        }
    }

    private void actualizarAnimales(float delta) {
        // No actualizar animales si el juego ha terminado
        if (gameTimeOver) return;

        // Actualizar todas las águilas
        for (Animal aguila : aguilas) {
            aguila.update(delta);
        }
```

```
        // Actualizar todos los caballos
        for (Animal caballo : caballos) {
            caballo.update(delta);
        }

        // Actualizar todos los bueyes
        for (Animal buey : bueyes) {
            buey.update(delta);
        }
    }

    private void limpiarAnimales() {
        // Eliminar animales que ya no son visibles
        Iterator<Animal> iter = aguilas.iterator();
        while (iter.hasNext()) {
            Animal animal = iter.next();
            if (!animal.visible) {
                iter.remove();
            }
        }

        iter = caballos.iterator();
        while (iter.hasNext()) {
            Animal animal = iter.next();
            if (!animal.visible) {
                iter.remove();
            }
        }

        iter = bueyes.iterator();
        while (iter.hasNext()) {
            Animal animal = iter.next();
            if (!animal.visible) {
                iter.remove();
            }
        }
    }

    private void moverJugadores() {
        // No permitir movimiento si el juego ha terminado
        if (gameTimeOver) return;

        float delta = Gdx.graphics.getDeltaTime();

        // Movimiento horizontal jugador 1
        if (Gdx.input.isKeyPressed(Input.Keys.A)) {
            posJugador1.x -= velocidadJugador * delta;
        }
        if (Gdx.input.isKeyPressed(Input.Keys.D)) {
            posJugador1.x += velocidadJugador * delta;
        }

        // Movimiento horizontal jugador 2
        if (Gdx.input.isKeyPressed(Input.Keys.LEFT)) {
            posJugador2.x -= velocidadJugador * delta;
        }
        if (Gdx.input.isKeyPressed(Input.Keys.RIGHT)) {
            posJugador2.x += velocidadJugador * delta;
        }
```

```
        // Límites de pantalla para jugador 1
        if (posJugador1.x < 0) posJugador1.x = 0;
        if (posJugador1.x > VIRTUAL_WIDTH - 100) posJugador1.x =
VIRTUAL_WIDTH - 100;

        // Límites de pantalla para jugador 2
        if (posJugador2.x < 0) posJugador2.x = 0;
        if (posJugador2.x > VIRTUAL_WIDTH - 100) posJugador2.x =
VIRTUAL_WIDTH - 100;

        // Disparos
        if (Gdx.input.isKeyJustPressed(Input.Keys.W)) {
            if (disparoSonido1 != null && sfxOn) {
                disparoSonido1.play();
            }
            verificarImpacto(posJugador1, true);
        }
        if (Gdx.input.isKeyJustPressed(Input.Keys.UP)) {
            if (disparoSonido2 != null && sfxOn) {
                disparoSonido2.play();
            }
            verificarImpacto(posJugador2, false);
        }
    }

    private void verificarImpacto(Vector2 posJugador, boolean
esJugador1) {
        // No verificar impactos si el juego ha terminado
        if (gameTimeOver) return;

        Rectangle jugadorRect = new Rectangle(posJugador.x,
posJugador.y, 100, 100);

        // Verificar impacto con águilas
        Iterator<Animal> iter = aguilas.iterator();
        while (iter.hasNext()) {
            Animal animal = iter.next();
            if (animal.visible && animal.bounds.overlaps(jugadorRect))
{
                iter.remove();
                if (sonidoPunto != null && sfxOn) {
                    sonidoPunto.play();
                }
                // Incrementar puntos según jugador
                if (esJugador1) {
                    puntosJugador1 += 3; // Águilas valen 3 puntos
                } else {
                    puntosJugador2 += 3;
                }
            }
        }



        // Verificar impacto con caballos
        iter = caballos.iterator();
        while (iter.hasNext()) {
            Animal animal = iter.next();
            if (animal.visible && animal.bounds.overlaps(jugadorRect))
{
                iter.remove();
```

```
                if (sonidoPunto != null && sfxOn) {
                    sonidoPunto.play();
                }
                // Incrementar puntos según jugador
                if (esJugador1) {
                    puntosJugador1 += 2; // Caballos valen 2 puntos
                } else {
                    puntosJugador2 += 2;
                }
            }
        }

        // Verificar impacto con bueyes
        iter = bueyes.iterator();
        while (iter.hasNext()) {
            Animal animal = iter.next();
            if (animal.visible && animal.bounds.overlaps(jugadorRect))
{
                iter.remove();
                if (sonidoPunto != null && sfxOn) {
                    sonidoPunto.play();
                }
                // Incrementar puntos según jugador
                if (esJugador1) {
                    puntosJugador1 += 1; // Bueyes valen 1 punto
                } else {
                    puntosJugador2 += 1;
                }
            }
        }
    }

    @Override
    public void resize(int width, int height) {
        viewport.update(width, height, true);
        camera.update();
        System.out.println("Ventana redimensionada: " + width + " x "
+ height);
    }

    @Override
    public void pause() {
        // Método requerido por la interfaz Screen
    }

    @Override
    public void resume() {
        // Método requerido por la interfaz Screen
    }

    @Override
    public void hide() {
        // Método requerido por la interfaz Screen
    }

    private boolean musicOn = true;
    private boolean sfxOn = true;
    @Override
    public void dispose() {
        try {
            if (image != null) image.dispose();
```

```
            if (jugador1 != null) jugador1.dispose();
            if (jugador2 != null) jugador2.dispose();
            if (disparoSonido1 != null) disparoSonido1.dispose();
            if (disparoSonido2 != null) disparoSonido2.dispose();
            if (sonidoPunto != null) sonidoPunto.dispose();
            if (aguilaTexture != null) aguilaTexture.dispose();
            if (caballoTexture != null) caballoTexture.dispose();
            if (bueyTexture != null) bueyTexture.dispose();
            if (fuenteOeste != null) fuenteOeste.dispose();
            if (hudFondo != null) hudFondo.dispose();
            if (musicadefondopartida != null) {
                musicadefondopartida.dispose();
            }
            System.out.println("Recursos de GameScreen liberados
correctamente");
        } catch (Exception e) {
            System.err.println("Error en dispose(): " +
e.getMessage());
        }
    }
    public void setMusicOn(boolean musicOn) {
        this.musicOn = musicOn;
        // Aplicar el cambio inmediatamente
        if (musicadefondopartida != null) {
            if (musicOn) {
                if (!musicadefondopartida.isPlaying()) {
                    musicadefondopartida.play();
                }
            } else {
                if (musicadefondopartida.isPlaying()) {
                    musicadefondopartida.stop();
                }
            }
        }
    }

    public void setSfxOn(boolean sfxOn) {
        this.sfxOn = sfxOn;
    }

    // Método para configurar la duración del juego en minutos
    public void setGameDuration(float minutes) {
        this.gameDuration = minutes * 60; // Convertir a segundos
        this.gameTimer = 0;
        this.gameTimeOver = false;
    }
}
```