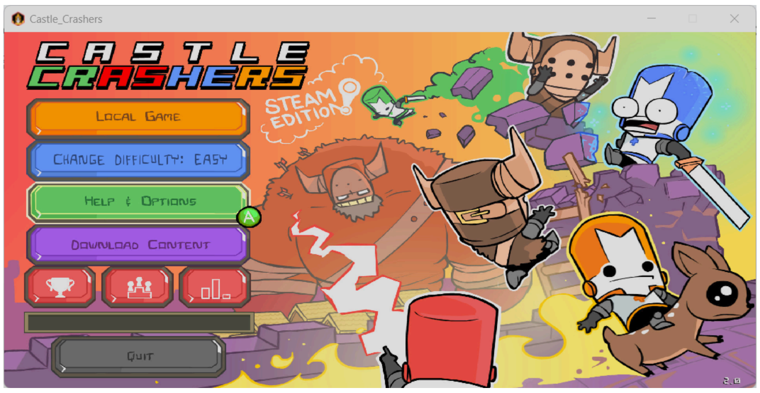
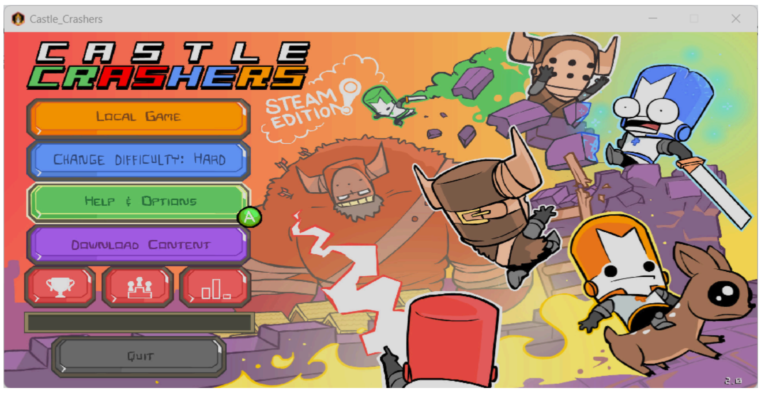
Entrega Individual Final - Crear un Juego en 2D en libGDX

Aplicación probada en el Lw3jglLauncher.java.
He creado un **menú** con tres opciones:
Local game: Inicia el nivel 1
Change difficulty: Alterna la dificultad de fácil a difícil, y cambia el texto del botón.
Quit: Cierra el juego

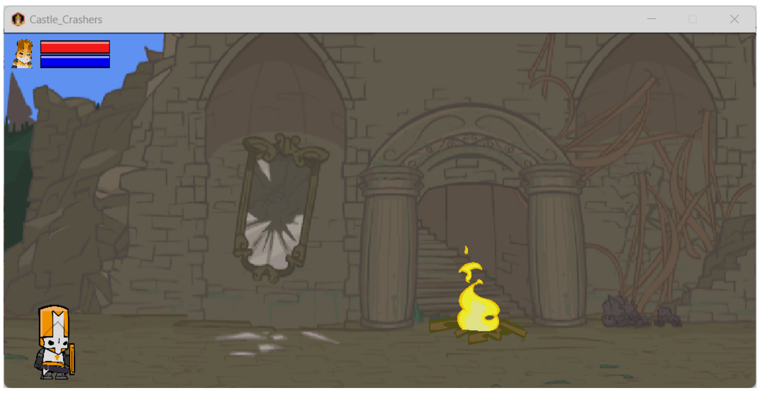




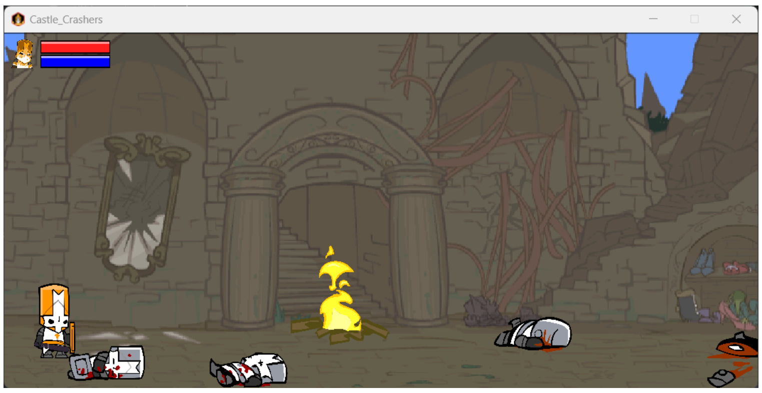
Pantallas de juego:
- Muevo el personaje en eje X e Y con "WASD / flechas"
- Pantalla móvil con uso de cámara.

MainScreen:

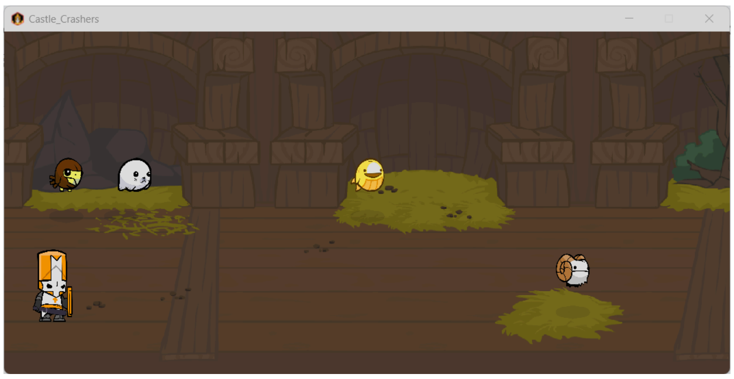
Modo fácil:

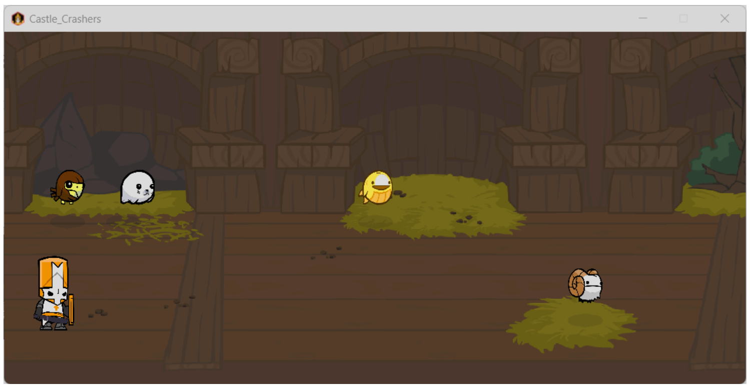


Modo difícil:



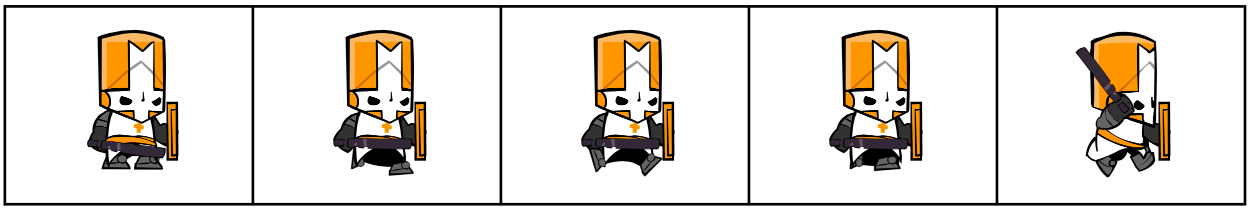
Hecho por Jorge Varela Zamora, de 2º del C.F.G.S. de D.A.M, curso 2024-2025, I.E.S. Vista Alegre animalArkScreen: dibuja animales flotantes(suben y bajan en un rango)



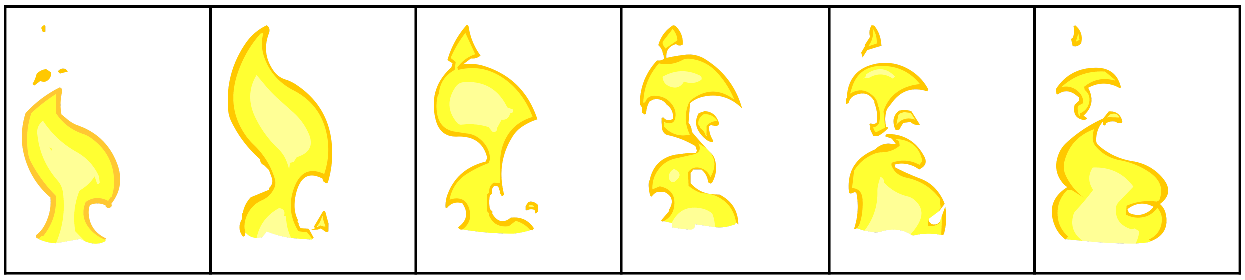


Características adicionales:

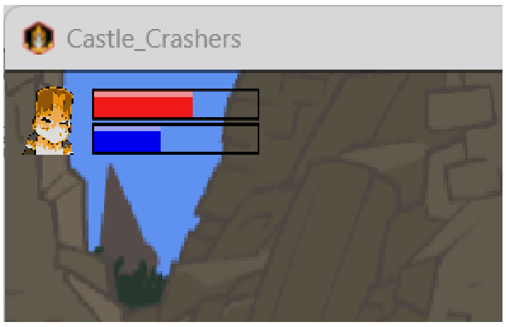
- **Animaciones**:
  He creado animaciones diferentes para el jugador quieto, movimiento y golpeando, juntando las extremidades en png mediante el uso de GIMP, creando los frames:



El fuego de la hoguera descargando los pngs:



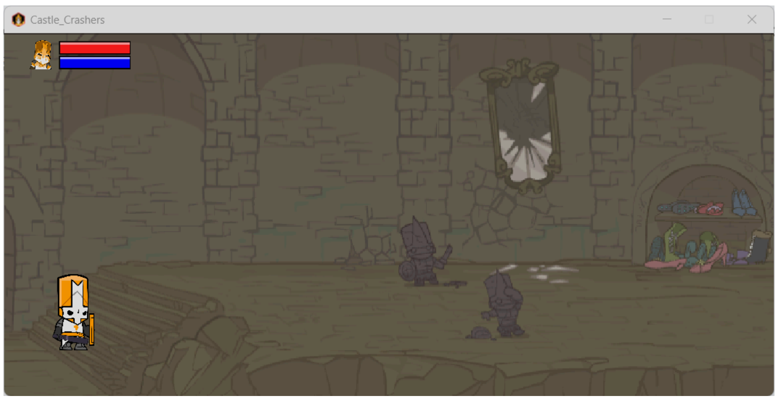
- **Colisiones**: se detecta colision entre jugador y fuego.
- **Sonido**:
  Música de fondo diferente en el menú, MainScreen y animalArkScreen (3).
- **Interfaz gráfica** / HUD:
  Sigue al jugador y muestra su vida y resistencia.

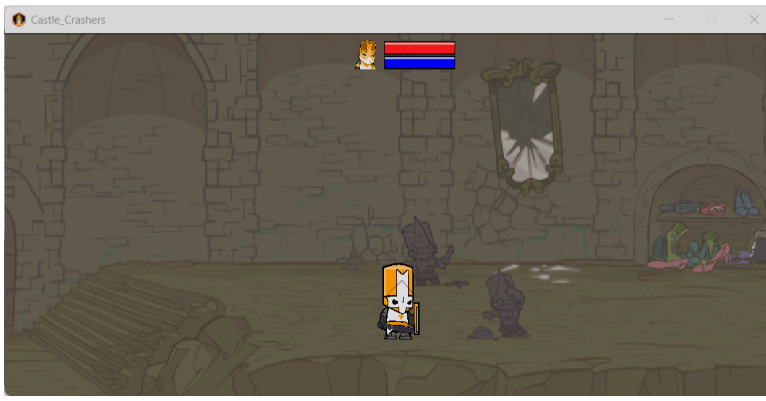


Estilo gráfico: 2d, gráficos png descargados de internet, algunos compuestos por mí con GIMP, y barras de vida del HUD diseñado en SVG y convertidos a png.

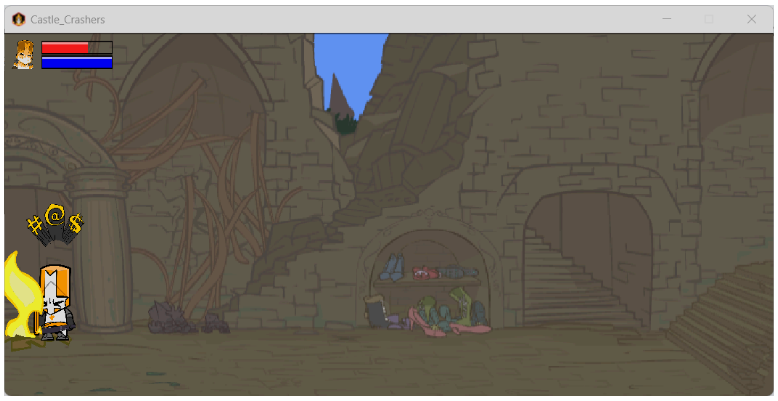
Mecánicas:

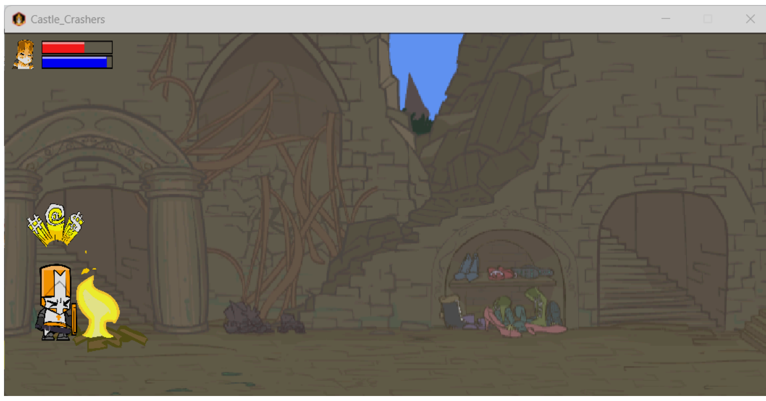
- Subir escaleras y suelo desnivelado.





- Correr con 'SHIFT_LEFT' baja la estamina (barra azul).
- Daño por fuego: Baja la vida y hace animación de daño al jugador. Si se acaba la vida, vuelve al menú principal.





Hecho por Jorge Varela Zamora, de 2º del C.F.G.S. de D.A.M, curso 2024-2025, I.E.S. Vista Alegre

FUTURAS ACTUALIZACIONES:

- Usar método resize() para hacer la app responsive.
- Añadir más frames a las animaciones.
- Añadir más animaciones, enemigos y niveles / screens.
- Añadir más botones al menú.

Códigos del juego:

```
package io.github.Castle_Crashers

import com.badlogic.gdx.Application
import com.badlogic.gdx.Gdx
import com.badlogic.gdx.Screen
import com.badlogic.gdx.audio.Music
import com.badlogic.gdx.graphics.GL20
import com.badlogic.gdx.graphics.Texture
import com.badlogic.gdx.graphics.g2d.SpriteBatch
import com.badlogic.gdx.scenes.scene2d.InputEvent
import com.badlogic.gdx.scenes.scene2d.Stage
import com.badlogic.gdx.utils.viewport.ScreenViewport
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener

import com.badlogic.gdx.graphics.g2d.BitmapFont //Para el boton con texto
import com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator
import com.badlogic.gdx.scenes.scene2d.ui.TextButton
import com.badlogic.gdx.scenes.scene2d.ui.Skin
import com.badlogic.gdx.scenes.scene2d.ui.Label
import com.badlogic.gdx.scenes.scene2d.ui.TextButton.TextButtonStyle
import com.badlogic.gdx.graphics.Color

import com.badlogic.gdx.utils.Align


class MainGame(private val game: Main) : Screen {

   private lateinit var batch: SpriteBatch
   private lateinit var stage: Stage
   private lateinit var menuTexture: Texture
   private lateinit var btnTexture: Texture
   private lateinit var blueButtonBackground: Texture
   private lateinit var backgroundMusic: Music
   private val easy:String = "Easy"
   private val hard:String = "Hard"
   private var selectedDifficulty:String = easy
   private var isEasyMode:Boolean = true
```

```
override fun show() {

    batch = SpriteBatch()
    loadTextures()

    stage = Stage(ScreenViewport())
    Gdx.input.inputProcessor = stage

    //Musica de fondo
    backgroundMusic =
           Gdx.audio.newMusic(Gdx.files.internal("Music/Four_Brave_Champions.mp3
           "))
    backgroundMusic.isLooping = true
    backgroundMusic.volume = 1f
    backgroundMusic.play()

    val btnLocalGame = MenuButton(46f, 354f, 290f, 50f, btnTexture) {
        game.screen = GameScreen(game, isEasyMode)
    }

    val btnQuit = MenuButton(80f,20f,220f,50f,btnTexture){
        Gdx.app.exit()
    }

    // Cargar la fuente
    val font = loadFont(20)

    // Crear un estilo para el botón
    val textButtonStyle = TextButtonStyle()
    textButtonStyle.font = font // Asigna la fuente
    textButtonStyle.fontColor = Color(0 / 255f, 0 / 255f, 0 / 255f, 0.5f)


    // Crear el botón
    val miBoton = TextButton("Change difficulty: " + selectedDifficulty,
           textButtonStyle)
    miBoton.setPosition(70f, 302f) // Ubicación del botón
    miBoton.setSize(450f, 40f) // Tamaño del botón
    miBoton.label.setAlignment(Align.left)


    // Manejar el evento de clic
    miBoton.addListener(object : ClickListener() {
        override fun clicked(event: InputEvent?, x: Float, y: Float) {
            if(isEasyMode){
                selectedDifficulty = hard
                isEasyMode = false
            }else{
                selectedDifficulty = easy
                isEasyMode = true
            }
            miBoton.setText("Change difficulty: " + selectedDifficulty)
        }
    })

    stage.addActor(btnLocalGame)
    stage.addActor(btnQuit)
    stage.addActor(miBoton)

}

override fun render(delta: Float) {
    Gdx.gl.glClearColor(0f, 0f, 0f, 0f)
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT)

    batch.begin()
    batch.draw(menuTexture,
        0f, 0f,
        Gdx.graphics.width.toFloat(), Gdx.graphics.height.toFloat(),
        0, 0,
        menuTexture.width, menuTexture.height,
        false, false)
    batch.draw(blueButtonBackground, 54f,310f,274f,30f)
    batch.end()

    stage.act(delta)
    stage.draw()

}

private fun loadTextures(){
    menuTexture = Texture(Gdx.files.internal("mainMenu.jpg"))
    btnTexture = Texture(Gdx.files.internal("Menu/transparentButton.png"))
    blueButtonBackground =
           Texture(Gdx.files.internal("Menu/blueButtonBackground.png"))
}

private fun loadFont(size: Int): BitmapFont {
    val generator =
           FreeTypeFontGenerator(Gdx.files.internal("Fonts/CastleCrashers.ttf"))
    val parameter = FreeTypeFontGenerator.FreeTypeFontParameter()
    parameter.size = size
```

```
        parameter.characters =
                "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789.:,;<>
                +-*/=!'()"
        val font = generator.generateFont(parameter)
        generator.dispose() // Libera memoria
        return font
    }

    override fun resize(width: Int, height: Int) {
        stage.viewport.update(width, height, true)
    }

    override fun pause() {}
    override fun resume() {}
    override fun hide() {
        if (::backgroundMusic.isInitialized) backgroundMusic.stop()
    }

    fun create(){

    }

    override fun dispose() {
        batch.dispose()
        menuTexture.dispose()
        stage.dispose()
        if (::backgroundMusic.isInitialized)backgroundMusic.dispose()
    }
}
```

```
package io.github.Castle_Crashers

import com.badlogic.gdx.graphics.g2d.Animation
import com.badlogic.gdx.graphics.g2d.TextureRegion
import com.badlogic.gdx.Gdx
import com.badlogic.gdx.Input
import com.badlogic.gdx.Screen
import com.badlogic.gdx.graphics.GL20
import com.badlogic.gdx.graphics.OrthographicCamera
import com.badlogic.gdx.graphics.Texture
import com.badlogic.gdx.graphics.g2d.SpriteBatch
import com.badlogic.gdx.math.Rectangle

class GameScreen(private val game: Main, private var isEasyMode: Boolean) : Screen
              {

    private lateinit var batch: SpriteBatch
```

```kotlin
    private lateinit var background: Texture
    private lateinit var OKStanding: Texture
    private lateinit var OKWalking1: Texture
    private lateinit var OKWalking2: Texture
    private lateinit var OKWalking3: Texture
    private lateinit var OKAtacking: Texture
    private lateinit var HUD: Texture
    private lateinit var HUDBackground:Texture
    private lateinit var HUDRedbar: Texture
    private lateinit var HUDBluebar: Texture

    private lateinit var fire1: Texture
    private lateinit var fire2: Texture
    private lateinit var fire3: Texture
    private lateinit var fire4: Texture
    private lateinit var fire5: Texture
    private lateinit var fire6: Texture
    private lateinit var firewood: Texture
    private lateinit var damage1: Texture
    private lateinit var damage2: Texture
    private lateinit var badeyes: Texture

    private lateinit var dead1: Texture
    private lateinit var dead2: Texture
    private lateinit var dead3: Texture
    private lateinit var dead4: Texture
    private lateinit var dead5: Texture


    private lateinit var OKPortrait: Texture
    private lateinit var walkingAnimation: Animation<TextureRegion>
    private lateinit var atackingAnimation: Animation<TextureRegion>
    private lateinit var standingAnimation: Animation<TextureRegion>
    private lateinit var currentAnimation: Animation<TextureRegion>
    private lateinit var fireAnimation: Animation<TextureRegion>
    private lateinit var damageAnimation: Animation<TextureRegion>
    private var stateTime = 0f
    private var globalStateTime = 0f
    private var speed = 300f // 200f on production
    private lateinit var camera: OrthographicCamera

    private var playerX = 100f
    private var playerY = 100f
    private var FTop = 160f
    private var FBottom = 80f

    private var  STop = 260f // FTop + 240f
    private var SBottom = 140f
```

```
private var stairsX = 1100f


private var isMoving:Boolean = false
private var isAtacking:Boolean = false
private var isLookingToRight = true
private var isRunning:Boolean = false
private var isGettingDamage:Boolean = false
private var canRun:Boolean = false
private var stamina = 100f
private var health = 100f


private fun handleInput(delta: Float) {
    Gdx.input.setOnscreenKeyboardVisible(false) // Para que no abra el teclado
           al pulsar teclas

    isMoving = false
    if (Gdx.input.isKeyPressed(Input.Keys.M)) {
        game.screen = MainGame(game) // Vuelve al menu principal
    }

    if (Gdx.input.isKeyPressed(Input.Keys.D) ||
           Gdx.input.isKeyPressed(Input.Keys.RIGHT)) {
        if (playerX <= stairsX) {
            playerX += speed * delta
        } else if (playerX >= stairsX && playerX < stairsX + 100) {
            playerX += speed * delta; playerY += speed * delta
        } else {
            playerX += speed * delta
        }

        if (playerX > 1800f) {
            game.screen = animalarkScreen(game)
        }

        currentAnimation = walkingAnimation // Use walking animation
        isMoving = true
        isLookingToRight = true
    }

    if (Gdx.input.isKeyPressed(Input.Keys.A) ||
           Gdx.input.isKeyPressed(Input.Keys.LEFT)) {
        if (playerX <= stairsX) {
            playerX -= speed * delta
        } else if (playerX >= stairsX && playerX < stairsX + 100 && playerY >=
           FBottom) {
            playerX -= speed * delta; playerY -= speed * delta
        } else {
            playerX -= speed * delta
        }

        currentAnimation = walkingAnimation // Use walking animation
        isMoving = true
        isLookingToRight = false
    }

    if (Gdx.input.isKeyPressed(Input.Keys.W) ||
           Gdx.input.isKeyPressed(Input.Keys.UP)) {
        if (playerX < stairsX) {
            if (playerY <= FTop) {
                playerY += speed * delta
            }
        }

        if (playerX >= stairsX + 100) {
            if (playerY <= STop) {
                playerY += speed * delta
            }
        }

        isMoving = true
    }

    if (Gdx.input.isKeyPressed(Input.Keys.S) ||
           Gdx.input.isKeyPressed(Input.Keys.DOWN)) {
        if (playerX < stairsX) {
            if (playerY >= FBottom) {
                playerY -= speed * delta
            }
        }

        if (playerX >= stairsX + 100) {
            if (playerY >= SBottom) {
                playerY -= speed * delta
            }
        }

        isMoving = true
    }

    if (Gdx.input.isKeyPressed(Input.Keys.Q)) {
        currentAnimation = atackingAnimation // Cambia a la animación de ataque
        isAtacking = true
        isMoving = false // Si estás atacando, no te mueves
    }else{
        isAtacking = false
    }

    if(Gdx.input.isKeyPressed(Input.Keys.SHIFT_LEFT) && canRun) {  //ALT_LEFT
           //SHIFT_RIGHT
        speed = 600f
        stamina -= delta * 140 // Se gasta estamina

        if (stamina <= 0) {
            stamina = 0f
            canRun = false // No puede correr hasta que la estamina recupere 20
        }

        isRunning = true
    } else {
        speed = 300f
        stamina += delta * 20 // Se regenera estamina

        if (stamina > 100f) stamina = 100f // Límite máximo de estamina

        // Si la estamina se regenera hasta 20, vuelve a permitir correr
        if (stamina >= 20f) {
            canRun = true
        }

        isRunning = false
    }


    if (!isMoving && !isAtacking) {
        currentAnimation = standingAnimation
    }

}


private fun loadAnimations() {
    try {
        background = Texture(Gdx.files.internal("Backgrounds/1Level.png"))
        HUD = Texture(Gdx.files.internal("HUD/HUD.png"))
        HUDBackground = Texture(Gdx.files.internal("HUD/HUDBackground.png"))
        HUDRedbar = Texture(Gdx.files.internal("HUD/HUDRedbar.png"))
        HUDBluebar = Texture(Gdx.files.internal("HUD/HUDBluebar.png"))
        OKPortrait = Texture(Gdx.files.internal("HUD/OKPortrait.png"))
        badeyes = Texture(Gdx.files.internal("HUD/badeyes.png"))

        OKStanding = Texture(Gdx.files.internal("Characters/OKStanding.png"))
        OKAtacking = Texture(Gdx.files.internal("Characters/OKAtacking.png"))

        // Cargar las tres texturas para la animación de caminar
        OKWalking1 = Texture(Gdx.files.internal("Characters/OKWalking3.png"))
        OKWalking2 = Texture(Gdx.files.internal("Characters/OKWalking2.png"))
        OKWalking3 = Texture(Gdx.files.internal("Characters/OKWalking1.png"))

        // Crear un array con las texturas y dividirlas en regiones para
           caminar
        val walkFrames = arrayOf(
            TextureRegion(OKWalking1),
            TextureRegion(OKWalking2),
            TextureRegion(OKWalking3)
        )

        // Crear la animación con un tiempo entre fotogramas de 0.1 segundos
        walkingAnimation = Animation(0.1f, *walkFrames)

        // Crear un array con las texturas y dividirlas en regiones para atacar
        val attackFrames = arrayOf(
            TextureRegion(OKAtacking),
        )

        // Crear la animación de ataque con un tiempo entre fotogramas de 0.2
           segundos
        atackingAnimation = Animation(0.2f, *attackFrames)

        val standFrames = arrayOf(
            TextureRegion(OKStanding),
        )

        // Crea animacion de parado con tiempo entre fotogramas de 0.1 segundos
        standingAnimation = Animation(0.1f, *standFrames)


        fire1 = Texture(Gdx.files.internal("Effects/Fire/fire1.png"))
        fire2 = Texture(Gdx.files.internal("Effects/Fire/fire2.png"))
        fire3 = Texture(Gdx.files.internal("Effects/Fire/fire3.png"))
        fire4 = Texture(Gdx.files.internal("Effects/Fire/fire4.png"))
        fire5 = Texture(Gdx.files.internal("Effects/Fire/fire5.png"))
        fire6 = Texture(Gdx.files.internal("Effects/Fire/fire6.png"))

        val fireFrames = arrayOf(
            TextureRegion(fire1),
            TextureRegion(fire2),
            TextureRegion(fire3),
            TextureRegion(fire4),
            TextureRegion(fire5),
            TextureRegion(fire6)
        )

        fireAnimation = Animation(0.1f, *fireFrames)

        firewood = Texture(Gdx.files.internal("Props/firewood.png"))


        damage1 = Texture(Gdx.files.internal("HUD/damage1.png"))
        damage2 = Texture(Gdx.files.internal("HUD/damage2.png"))
        val damageFrames = arrayOf(TextureRegion(damage1),
           TextureRegion(damage2))
        damageAnimation = Animation(0.1f, *damageFrames)

        if(!isEasyMode){ //Solo los carga en hard mode
            dead1 = Texture(Gdx.files.internal("Props/dead1.png"))
            dead2 = Texture(Gdx.files.internal("Props/dead2.png"))
            dead3 = Texture(Gdx.files.internal("Props/dead3.png"))
            dead4 = Texture(Gdx.files.internal("Props/dead4.png"))
            dead5 = Texture(Gdx.files.internal("Props/dead5.png"))
        }




    } catch (e: Exception) {
        Gdx.app.error("GameScreen", "Error cargando la imagen: ${e.message}")
    }

    currentAnimation = standingAnimation // Default to walking animation when
           idle
}

override fun show() {

    Gdx.input.setOnscreenKeyboardVisible(false)


    batch = SpriteBatch()
    loadAnimations()

    camera = OrthographicCamera()
    camera.setToOrtho(false, Gdx.graphics.width.toFloat(),
           Gdx.graphics.height.toFloat())
}

override fun render(delta: Float) {
    handleInput(delta)
```

```
Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT)

println("playerX = $playerX")  // Verificación en la consola
if (playerX < 1120f) {
    camera.position.set(playerX + Gdx.graphics.width / 2f,
       Gdx.graphics.height / 2f, 0f)
}
camera.update()

val backgroundScrollSpeed = 0.5f
val backgroundX = -camera.position.x * backgroundScrollSpeed

batch.projectionMatrix = camera.combined

batch.begin()
batch.draw(background, backgroundX, 0f, 3000f, 500f)

//Background props
batch.draw(firewood, backgroundX + 1014f, 60f, 140f,40f)


// Si está caminando o atacando, usamos la animación correspondiente
if (isMoving) {
    stateTime += delta // Actualizar el tiempo de la animación
    currentAnimation = walkingAnimation // We set the walking animation to
       current
}

batch.draw(HUDRedbar, playerX+50, 450f, 100f * health / 100f, 40f)
batch.draw(HUDBluebar, playerX+50, 450f, 100f * stamina / 100f, 40f)
       //Cambia el ancho segun la estamina que tenga
batch.draw(HUDBackground, playerX+50, 450f, 100f, 40f)


batch.draw(OKPortrait, playerX+10, 450f, 30f, 40f)


// Obtener el fotograma de la animación
val OKframe = currentAnimation.getKeyFrame(stateTime, true)
globalStateTime += delta
val fireFrame = fireAnimation.getKeyFrame(globalStateTime, true)

// Dibujar el personaje con la animación seleccionada
if (isLookingToRight) {
    batch.draw(OKframe, playerX, playerY - 100, 140f, 120f)
} else {
    batch.draw(OKframe, playerX + 140, playerY - 100, -140f, 120f)
}
```

```
if(isGettingDamage){
    batch.draw(damageAnimation.getKeyFrame(globalStateTime, true),
       playerX+30, playerY +40, 80f,60f)

    if(isLookingToRight){
        batch.draw(badeyes, playerX+58, playerY-44, 32f, 14f)
    }else{
        batch.draw(badeyes, playerX+82, playerY-44, -32f, 14f)
    }

}




batch.draw(fireFrame, backgroundX + 1054f, 80f, 60f,120f)

if(!isEasyMode){  //Llena el mapa de cadaveres
    batch.draw(dead1, backgroundX + 700f, 10f, 110f,50f)

    batch.draw(dead2, backgroundX + 900f, 0f, 110f,50f)

    batch.draw(dead3, backgroundX + 1300f, 50f, 110f,50f)

    batch.draw(dead4, backgroundX + 1600f, 30f, 110f,50f)

    batch.draw(dead5, backgroundX + 1600f, 0f, 80f,40f)
}



batch.end()

if(playerX > backgroundX + 950f && playerX < backgroundX + 1070f && playerY
       > 100f && playerY < 200f){
    health -= delta * 20
    isGettingDamage = true
}else{
    isGettingDamage = false
}


if(health <= 0){
    game.screen = MainGame(game)
}
```

```kotlin
   }

   override fun resize(width: Int, height: Int) {}

   override fun pause() {}

   override fun resume() {}

   override fun hide() {}

   override fun dispose() {
       batch.dispose()
       if (::background.isInitialized) background.dispose()

       // Liberar las texturas asociadas a las animaciones
       OKStanding.dispose()
       OKWalking1.dispose()
       OKWalking2.dispose()
       OKWalking3.dispose()
       OKAtacking.dispose()

   }

}
```

```kotlin
package io.github.Castle_Crashers

import com.badlogic.gdx.Screen
import com.badlogic.gdx.Gdx
import com.badlogic.gdx.Input
import com.badlogic.gdx.audio.Music
import com.badlogic.gdx.graphics.GL20
import com.badlogic.gdx.graphics.OrthographicCamera
import com.badlogic.gdx.graphics.Texture
import com.badlogic.gdx.graphics.g2d.Animation
import com.badlogic.gdx.graphics.g2d.SpriteBatch
import com.badlogic.gdx.graphics.g2d.TextureRegion
import com.badlogic.gdx.math.Rectangle

class animalarkScreen(private val game: Main) : Screen {

   private lateinit var batch: SpriteBatch
   private lateinit var background: Texture
   private lateinit var OKStanding: Texture
   private lateinit var OKWalking1: Texture
```

```
    private lateinit var OKWalking2: Texture
    private lateinit var OKWalking3: Texture
    private lateinit var OKAtacking: Texture
    private lateinit var walkingAnimation: Animation<TextureRegion>
    private lateinit var atackingAnimation: Animation<TextureRegion>
    private lateinit var standingAnimation: Animation<TextureRegion>
    private lateinit var currentAnimation: Animation<TextureRegion>
    private lateinit var backgroundMusic: Music

    //Animal orbs
    private lateinit var AOPelter: Texture
    private lateinit var AOHawkster: Texture
    private lateinit var AOGoldenWhale: Texture
    private lateinit var AORammy: Texture
    private lateinit var shadow:Texture


    private var stateTime = 0f
    private val speed = 300f //200f on production
    private lateinit var camera: OrthographicCamera

    private var playerX = -174f
    private var playerY = 100f
    private var YTop = 300f
    private var YBottom = 80f

    private var isMoving:Boolean = false
    private var isAtacking:Boolean = false
    private var isLookingToRight = true

    private fun handleInput(delta: Float) {
        Gdx.input.setOnscreenKeyboardVisible(false) // Para que no abra el teclado
               al pulsar teclas

        isMoving = false
        if (Gdx.input.isKeyPressed(Input.Keys.M)) {
            game.screen = MainGame(game) //Vuelve al menu principal
        }

        if (Gdx.input.isKeyPressed(Input.Keys.D) ||
               Gdx.input.isKeyPressed(Input.Keys.RIGHT)){


            playerX += speed * delta

            currentAnimation = walkingAnimation
            isMoving = true
            isLookingToRight = true
```

```java
        }

        if (Gdx.input.isKeyPressed(Input.Keys.A) ||
               Gdx.input.isKeyPressed(Input.Keys.LEFT)) {

            if(playerX> -174f){
                playerX -= speed * delta
            }

            if(playerX< -174f){
                playerX = -174f
            }




            currentAnimation = walkingAnimation
            isMoving = true
            isLookingToRight = false

        }

        if(Gdx.input.isKeyPressed(Input.Keys.W) ||
               Gdx.input.isKeyPressed(Input.Keys.UP)) {


                if(playerY <= YTop){
                    playerY += speed * delta
                }
            currentAnimation = walkingAnimation
            isMoving = true
        }

        if(Gdx.input.isKeyPressed(Input.Keys.S) ||
               Gdx.input.isKeyPressed(Input.Keys.DOWN)) {

            if(playerY >= YBottom){
                playerY -= speed * delta
            }
            currentAnimation = walkingAnimation
            isMoving = true
        }

        if(Gdx.input.isKeyPressed(Input.Keys.Q)) {
            currentAnimation = atackingAnimation
            isMoving = true
            isAtacking = true
        }
```

```
    if(!isMoving){
        currentAnimation = standingAnimation
    }

}

private fun loadAnimations() {
    try {
        //Estas solo en esta clase
        background = Texture(Gdx.files.internal("Backgrounds/animalArk.png"))
        AOPelter = Texture(Gdx.files.internal("Animals/26_Pelter.png"))
        AOHawkster = Texture(Gdx.files.internal("Animals/16_Hawkster.png"))
        AOGoldenWhale =
           Texture(Gdx.files.internal("Animals/29_Golden_Whale.png"))
        AORammy = Texture(Gdx.files.internal("Animals/3_Rammy.png"))
        shadow = Texture(Gdx.files.internal("Animals/shadow.png"))

        //estas en todas las clases
        OKStanding = Texture(Gdx.files.internal("Characters/OKStanding.png"))
        OKAtacking = Texture(Gdx.files.internal("Characters/OKAtacking.png"))

        // Cargar las tres texturas para la animación de caminar
        OKWalking1 = Texture(Gdx.files.internal("Characters/OKWalking3.png"))
        OKWalking2 = Texture(Gdx.files.internal("Characters/OKWalking2.png"))
        OKWalking3 = Texture(Gdx.files.internal("Characters/OKWalking1.png"))

        // Crear un array con las texturas y dividirlas en regiones para
           caminar
        val walkFrames = arrayOf(
            TextureRegion(OKWalking1),
            TextureRegion(OKWalking2),
            TextureRegion(OKWalking3)
        )

        // Crear la animación con un tiempo entre fotogramas de 0.2 segundos
        walkingAnimation = Animation(0.1f, *walkFrames)

        // Crear un array con las texturas y dividirlas en regiones para atacar
        val attackFrames = arrayOf(
            TextureRegion(OKAtacking),
        )

        // Crear la animación de ataque con un tiempo entre fotogramas de 0.2
           segundos
        atackingAnimation = Animation(0.2f, *attackFrames)
        val standFrames = arrayOf(
            TextureRegion(OKStanding),
        )

        // Crear la animación de ataque con un tiempo entre fotogramas de 0.2
           segundos
        standingAnimation = Animation(0.2f, *standFrames)




    } catch (e: Exception) {
        Gdx.app.error("GameScreen", "Error cargando la imagen: ${e.message}")
    }

    currentAnimation = standingAnimation
}

override fun show() {
    Gdx.input.setOnscreenKeyboardVisible(false)

    backgroundMusic =
           Gdx.audio.newMusic(Gdx.files.internal("Music/Spanish_Waltz.mp3"))
    Gdx.app.log("Music", "Música cargada correctamente")

    backgroundMusic.isLooping = true
    backgroundMusic.volume = 1f
    Gdx.app.postRunnable {
        Gdx.app.log("Music", "Intentando reproducir música...")
        Thread.sleep(1000)
        backgroundMusic.play()
        Gdx.app.log("Music", "IsPlaying después de play():
           ${backgroundMusic.isPlaying}")
    }

    batch = SpriteBatch()
    loadAnimations()
    currentAnimation = standingAnimation

    camera = OrthographicCamera()
    camera.setToOrtho(false, Gdx.graphics.width.toFloat(),
           Gdx.graphics.height.toFloat())
}

override fun render(delta: Float) {
    handleInput(delta)

    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT)
```

```kotlin
        stateTime += delta

        if(playerX<=180f) {
            camera.position.set(playerX + Gdx.graphics.width / 2f,
               Gdx.graphics.height / 2f, 0f)
        }
        camera.update()

        val backgroundScrollSpeed = 0.5f
        val backgroundX = -camera.position.x * backgroundScrollSpeed

        batch.projectionMatrix = camera.combined

        batch.begin()
        batch.draw(background, backgroundX, 0f, 1600f, 600f) //background

        val frame = currentAnimation.getKeyFrame(stateTime, true)
        if (isLookingToRight) {
            batch.draw(frame, playerX, playerY - 100, 140f, 120f)
        } else {
            batch.draw(frame, playerX + 140f, playerY - 100, -140f, 120f)
        }

        batch.draw(AOPelter, backgroundX +200, animalGravity(260f,stateTime), 50f,
               50f)
        batch.draw(shadow, backgroundX +180, 220f, 50f, 20f)

        batch.draw(AOHawkster, backgroundX +100, animalGravity(260f,stateTime),
               50f, 50f)
        batch.draw(shadow, backgroundX +100, 220f, 50f, 20f)

        batch.draw(AOGoldenWhale, backgroundX +540, animalGravity(260f,stateTime),
               50f, 50f)
        batch.draw(shadow, backgroundX +540, 220f, 50f, 20f)

        batch.draw(AORammy, backgroundX +840, animalGravity(120f, stateTime), 50f,
               50f)
        batch.draw(shadow, backgroundX +840, 80f, 50f, 20f)


        batch.end()

    }

    private fun animalGravity(animalY: Float, time: Float): Float {
        val amplitude = 6f  // Altura máxima del movimiento
        val frequency = 2f   // Frecuencia de oscilación
        return animalY + amplitude * Math.sin((time *
                frequency).toDouble()).toFloat()
    }

    override fun resize(width: Int, height: Int) {}

    override fun pause() {}

    override fun resume() {}

    override fun hide() {
        if (::backgroundMusic.isInitialized) backgroundMusic.stop()
    }

    override fun dispose() {
        batch.dispose()
        if (::background.isInitialized) background.dispose()
        if (::backgroundMusic.isInitialized) backgroundMusic.dispose()
        OKStanding.dispose()
        OKWalking1.dispose()
        OKWalking2.dispose()
        OKWalking3.dispose()
        OKAtacking.dispose()
    }
}
```

```
package io.github.Castle_Crashers
import com.badlogic.gdx.Game

class Main : Game() {
    override fun create(){
        setScreen(MainGame(this))  //cambiar MainGame cuando se termine de modo dev
                para que aparezca menu
    }

    override fun render(){
        super.render()
    }

    override fun dispose(){
        screen?.dispose()
    }
}
```