ESCUELA INTERNACIONAL DE GERENCIA

Trabajo Final de Grado

APLICACIONES MULTIPLATAFORMA

TFG

Juan Ríos Jiménez

31/05/2021

Tabla de contenido

1. Objetivos

- Objetivos generales:

  Realizar un videojuego 2D musical en el que el usuario tiene que superar una serie de obstáculos para poder superar el nivel que va aumentando la dificultad conforme progresa.
  Explicado a fondo en la propuesta

  Obtener experiencia en el desarrollo de videojuegos, aprender a usar nuevas herramientas y mejorar en Android y Java.

  Mejorar habilidades artísticas.

  Vender el proyecto y obtener beneficios de este

- Objetivos específicos:

  Lograr salir de la interfaz y motor gráfico normal de Android y en vez de eso aplicarlo en realizar una simulación 2D física y en procesos necesarios para poder realizar el videojuego.

  Implementar imágenes en elementos del juego en Android ya que no vamos a usar los componentes normales de Android.

  Implementar objetos físicos.

  Detección de colisiones entre objetos y también detectar input del jugador.

  Generación de objetos en pantalla y tener una progresión de nivel.

  Sincronizar los objetos con el ritmo de la música.

  Creación de interfaz de juego. Desarrollar un modo de juego más apto para el usuario convencional, mostrar las vidas que tiene, indicador de donde está pulsando…

  Pulir funcionalidades básicas.

  Añadir sonidos y música.

2. Justificación

La motivación por la que desarrollo un videojuego para Android es por los siguiente:

Aprender a hacer videojuegos y entrar en el mundo ya que siempre ha sido algo que me ha llamado la atención.

Crear mi primer videojuego funcional.

Todos los videojuegos tienen un ciclo de vida, uno en el que el juego es muy conocido y lo juega mucha gente y después empieza un proceso de "envejecimiento" en el que los usuarios se cansan del producto y empiezan a buscar otros juegos. Depende mucho de la calidad del juego y de como se ha publicado que se prolongue el tiempo "vivo" del juego. Una estrategia muy común para mantenerlo vivo es actualizándolo y añadiéndole cosas con el transcurso del tiempo.

Entonces crear videojuegos nace de la necesidad de crear algo "nuevo" atractivo para los usuarios. Realmente no es crear algo nuevo, es de hacer sentir que es diferente . Por ejemplo como es el caso de los juegos de terror. El objetivo de todos esos juegos es de asustarte y con ello divertirte. Son muy parecidos todos y la funcionalidad es similar. Pero el problema es ese, que se acaban y nace ahí la necesidad de algo nuevo.

¿Qué pasa con mi juego?

Los juegos musicales están en decadencia, solo hay un par hoy día que son famosos y todavía se juegan aunque ya están en esa fase de envejecimiento. Como son el "Geometry Dash" y el "Osu!". No estoy diciendo que literalmente dejen de jugarse pero sí que han bajado mucho el número de usuarios.

Esto da puerta a que otros desarrolladores publiquen sus juegos. En mi caso aprovecho que este género está más "anticuado" para renovarlo añadiendo yo mi propio producto con un juego de móvil más dinámico, deslizar tu dedo por la pantalla es más interactivo que pulsar un botón y que salte un cuadrado, sientes más que eres tú el que realiza la acción e interactúa con el entorno. Buscaba este matiz porque no es algo que se vea mucho en juegos de móviles.

3. Análisis de la competencia

La competencia en el entorno que estoy trabajando no es muy alta ya que está abierto a otros muchos juegos dejando su hueco. Pero mi mayor competencia, móvil es Geometry Dash, un cuadrado que salta, para esquivar los obstáculos.

No es muy difícil de superar ya que nuestra aplicación es bastante diferente en cuanto a la manera de como superar esos obstáculos.

Mi juego cubre una manera diferente de jugar, innovar y ser intuitiva.

Con otras músicas y nuevos aspectos que ofrece.

No va a tener problema por ese lado.

Luego el juego se va a publicar por las redes sociales para conseguir las descargas que necesitamos.

Facebook , Twitter, Instagram, Telegram ,Whatsapp.

Luego para mantener enganchado al usuario es eso, darle la sensación de victoria por superar el nivel.

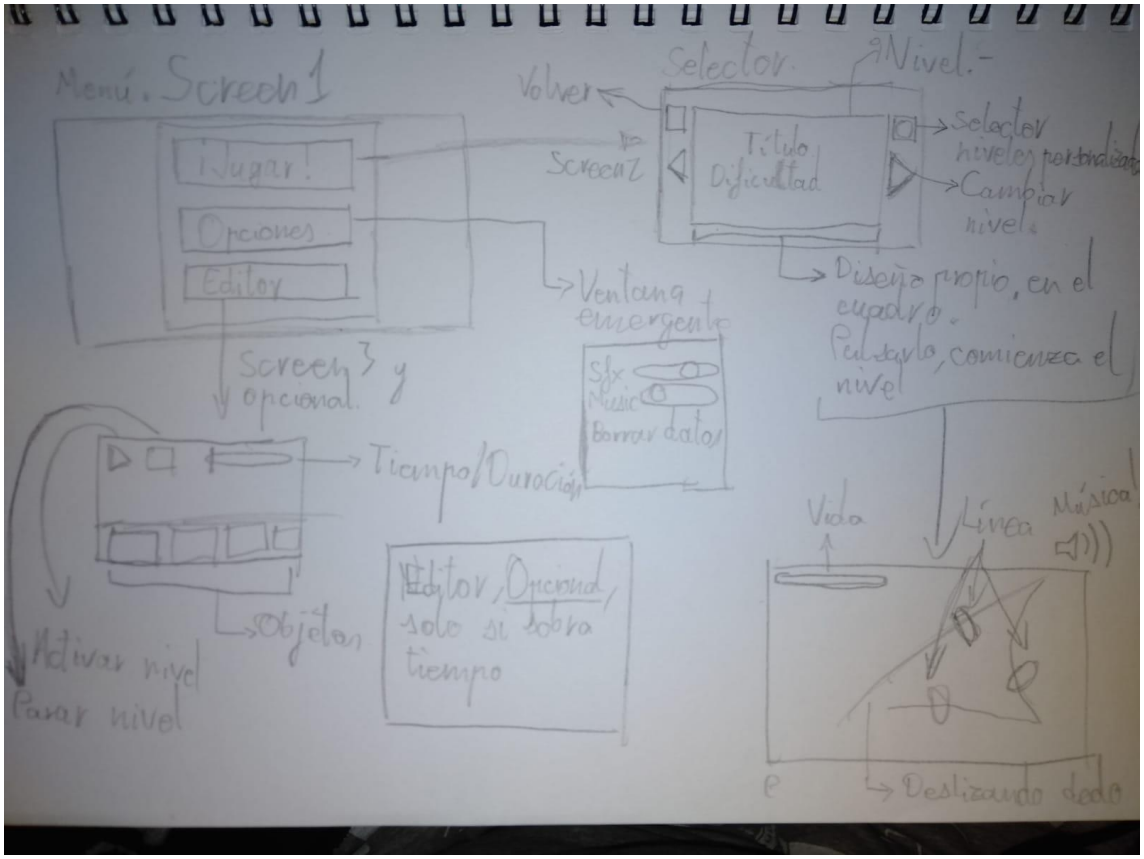
4. Propuesta detallada

Queremos hacer un videojuego 2D musical en el que el usuario se guía con el ritmo para poder progresar lo mejor posible. Para ello el usuario se le presentara una serie de obstáculos en pantalla.

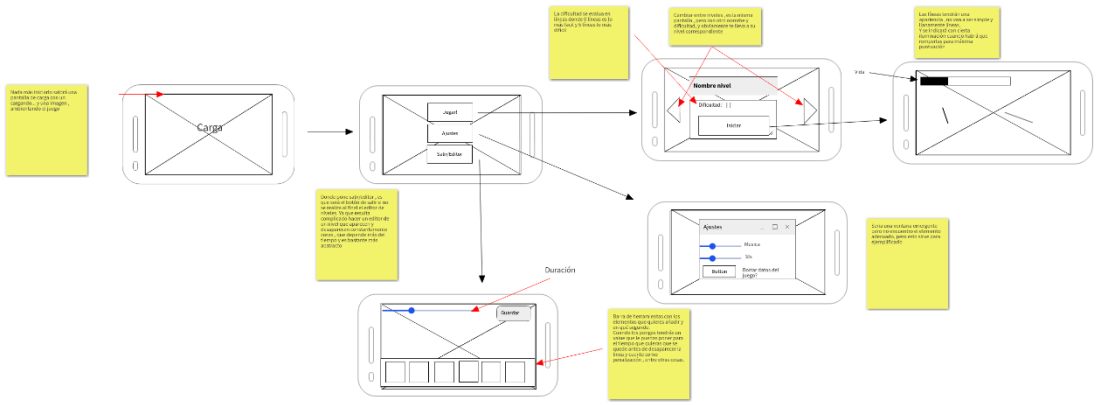
Los obstáculos de este juego son unas líneas que van apareciendo en la pantalla, siguen el ritmo de la música y el jugador tiene un tiempo para romperlas con el dedo antes de que desaparezcan y pierdan vidas. Las vidas se mostrarán en la parte superior izquierda de la pantalla indicándole la cantidad de oportunidades restantes que tiene el jugador antes de perder y tener que volver a empezar de cero.

Estará ambientado en un aspecto de pixel art. Las líneas van a tener aspecto de ladrillo. En este caso rompemos ladrillos.

4.1. Sketch

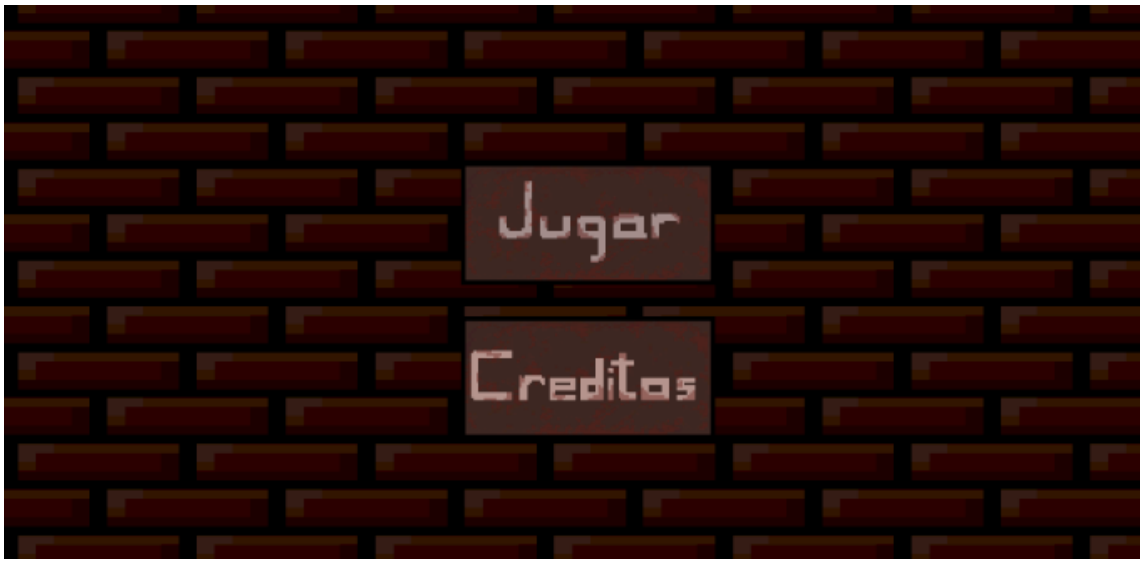


4.2. WireFrame

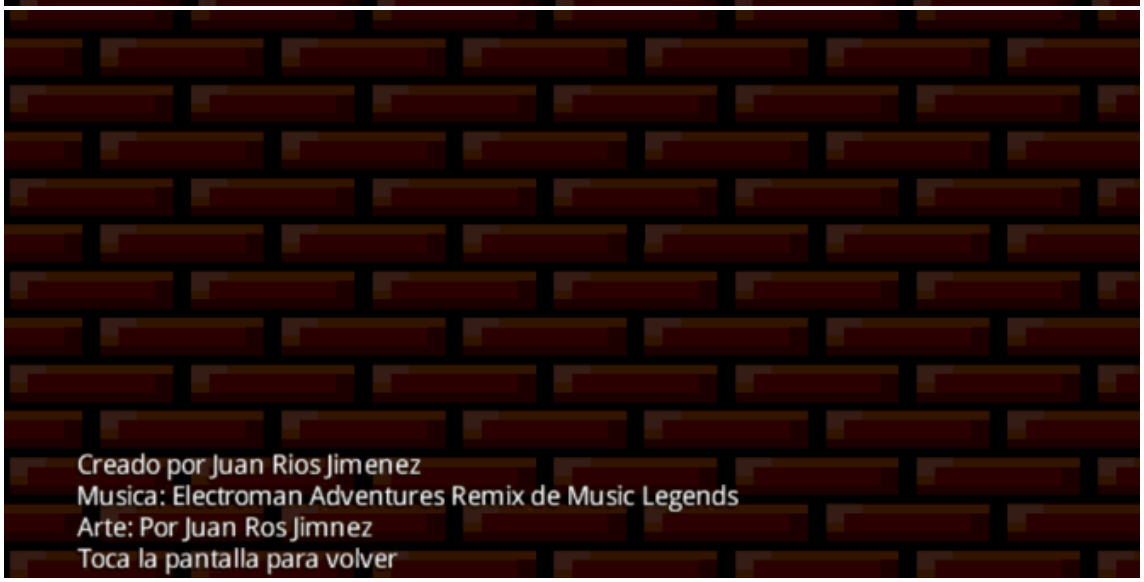


4.3. Mockup

El MockUp y el Prototipo son iguales, ya que hice la parte artística del juego antes, y después ya lo puse todo.


Jugar
Creditos


Vida: 1



.

4.4. Prototipo

No puedo poner un prototipo , es el juego en sí, no puedo hacer con un html o un powerpoint rompiendo líneas de forma interactiva.

5. Planificación Temporal

5.1. Especificación de los objetivos de las iteraciones

General:

- Crear la pantalla y renderizado.
- Arrancado de juego.

Crear la parte en Box2D del juego.(lógica y física)

- Creación de cuerpos.
- Creación de fixturas
- Generar cuerpos y que tipo de polígonos y ponerlos en pantalla.
- Detección de colisión de cuerpos
- Reacción del programa a la colisión de cuerpos
- Destrucción de cuerpos
- Generación por tiempo de cuerpos.

Crear la parte Scene2D del juego.(gráfica)

- Creación de actores.
- Creación de pantallas
- Carga de texturas e imágenes mediante el uso del AssetManager
- Creación de Atlas
- Carga de sonidos.
- Destrucción de imágenes
- Edición de imágenes.
- Renderizado de imágenes a tiempo real (para el swiper).

Unir ambas partes y crear las entidades en cuestión

- Arreglar el problema de Box2D y Scene2D con pasar metros a pixeles.
- Unir la imagen con el cuerpo
- Destrucción de ambas partes y creación de ambas partes.

General	Duración
Crear la pantalla y renderizado	1 día
Arrancado de juego	1 día

Scene2D	Duración
Creación de Actores	1 día
Creación de pantallas	4 días
Carga de texturas	2 día
Atlas	2 días
Carga de sonidos	1 día
Destrucción de imágenes	1 día
Edición de imágenes	1 día
Renderizado de imágenes	10 días

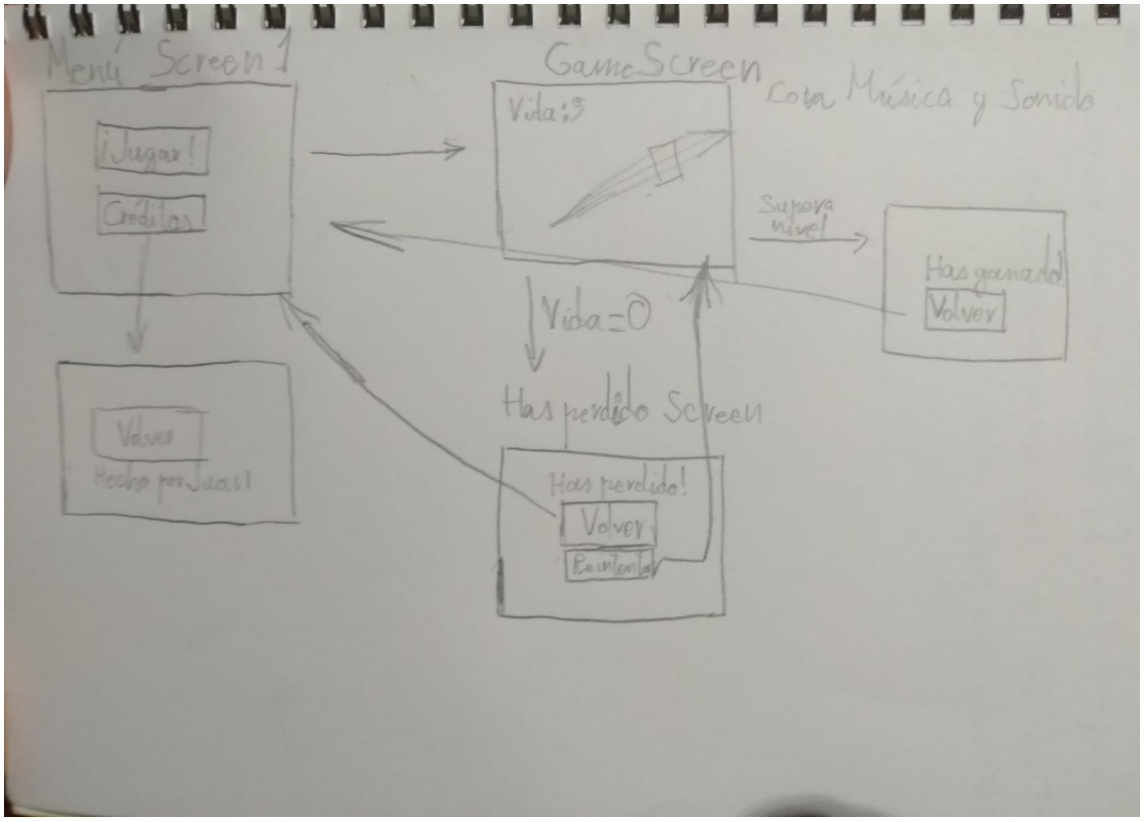
Box2D	Duración
Creación de cuerpos	1 día
Creación de fixturas	1 día
Generar cuerpos y tipo	2 días
Detección de colisiones	3 días
Reacción a colisión	3 días
Destrucción de cuerpos	4 días
Generación por tiempo de cuerpos	2 días

5.2. Prototipado en cada iteración

No ha habido cambios en el proyecto durante las iteraciones , hasta que no he alcanzado la parte gráfica + física que es conectar ya las diferentes cosas. Que es cuando ya hice el MockUp-

La mayor diferencia es la omisión de un montonazo de elementos que se proponían inicialmente en el Sketch y Wireframe, resultaron ser elementos del juego que no se podían implementar dentro de la fecha dada.

Lo que si tengo es un segundo sketch de lo que es la nueva interfaz ya que me hacia falta para poder reestructurarlo todo bien

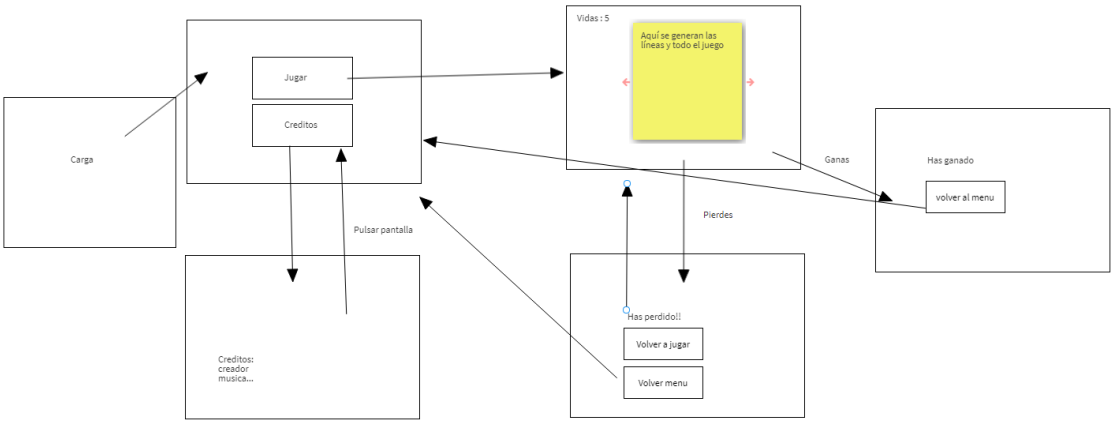


Como se puede observar se ha simplificado bastante, pero la funcionalidad esencial del juego se mantiene.

Ya la creación de más niveles es mucho más fácil puesto que el motor y todo lo demás está hecho.

6. Diseño de la aplicación

6.1. Esquema funcional



6.2. Diseño de estructuras de datos

Los objetos principales de este juego.

El primero de ellos obviamente es el objeto Game que es con el que instanciamos el juego.

Arrancamos el juego con MainGame.

```
public class MainGame extends Game {
    //Esta es la clase que prepara el juego
    //Con esto centralizamos la mayoria de los elementos que tenemos que cargar en el juego
    // Sounds, music, and images.
    private AssetManager manager;

    public BaseScreen gameScreen2, loadingScreen, MenuScreen, Gameover,win, Credits;

    public void create() {
        System.out.println("Cargando");

        //atlas = new TextureAtlas("textures.pack");  ---> Aquí estaba haciendo pruebas con un Atlas , pero
        // Realmente aquí no se utiliza., pero si lo cargamos para otras cosas.

        manager = new AssetManager();   //El AssetManager es el que carga imagenes y sonidos.
        //Para juegos mas grandes no es recomendable cargar uno por uno los elementos del juego.
        //Se recomienda uso de uno o varios Atlas (un unico archivo con varias imagenes), ya que
        //Reduce el codigo que hay que escribir y solo hay que cargar una imagen en vez de multiples.

        manager.load( fileName: "fondomenu.png", Texture.class);          //Fondo pantalla menu
        manager.load( fileName: "gradient.png", Texture.class);           //Parte grafica del cursor
        manager.load( fileName: "textures.pack", TextureAtlas.class);     //Atlas
        manager.load( fileName: "audio/menu.mp3", Music.class);           //Musica menu
        manager.load( fileName: "audio/electroman.mp3", Music.class);     //Musica nivel 1
        manager.load( fileName: "audio/breakLine.mp3", Sound.class);      //Sonido romper lineas
        manager.load( fileName: "audio/losehp.mp3", Sound.class);         //Sonido perder vida
        manager.load( fileName: "data/line.png", Texture.class);          //Textura de las lineas que no se consigue implementar bien
        manager.load( fileName: "empty.png", Texture.class);              //Textura de debuggeo es transparente.


        loadingScreen = new LoadingScreen( game: this); //Creamos la pantalla de carga
        /*
        * Hay que tener en cuenta un factor bastante interesante a la hora de cargar las cosas
        * Libgdx carga los elementos de forma asincrona y cuando realmente instanciamos lo que es la textura
        * para ponerla en lo que es la pantalla en sí, supone que si el assetmanager no ha terminado de cargar
        * la textura y la instanciamos da un error de memoria, porque no la encuentra y el juego
```

Como observamos tenemos el AssetManager que es el que carga todas las imágenes y audios.

Y luego tenemos las Screens que es lo segundo más importante de este proyecto, con ellas pasamos entre pantallas y en ellas se muestra todo el contenido

Usamos una clase abstracta del que heredan todas las pantallas el cual hemos llamado BaseScreen , que tiene los métodos necesarios para funcionar correctamente.

Son métodos de libgdx y se llaman solos cuando llegamos a instanciar las screens.

Hacemos @Overrides en las clases heredadas porque obviamente le vamos a poner contenido en algunos de los métodos en las otras screens si no no funciona.

```
public abstract class BaseScreen implements Screen {

    protected MainGame game;

    public BaseScreen(MainGame game) { this.game = game; }

    @Override
    public void show() {

    }

    @Override
    public void render(float delta) {


    }

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

    }

    @Override
    public void pause() {

    }

    @Override
    public void resume() {

    }

    @Override
    public void hide() {
```

Después de las Screens tenemos las cámaras , sin ellas no podemos ver el juego apropiadamente, en la pantalla de carga no ponemos.

La screen necesita también de un par de variables más para poder mostrar el entorno.

Stage , sobre todo para la parte gráfica y World , que es para la parte simulada o física.

Por ejemplo en MenuScreen solo tenemos Stage.

```
public class MenuScreen extends BaseScreen {

    private Stage stage;

    private Skin skin,skin2;

    private Image background;

    private TextureAtlas textures;

    private TextButton credits;

    private ImageButton boton1,boton2;

    private Music menuMusic;

    public MenuScreen(final MainGame game) {
        super(game);
        stage = new Stage(new FitViewport( worldWidth: 720, worldHeight: 370));

        textures = new TextureAtlas( internalPackFile: "textures.pack");

        skin = new Skin(Gdx.files.internal( path: "skin/uiskin.json"));

        skin2 = new Skin(game.getManager().get( fileName: "textures.pack",TextureAtlas.class));

        credits = new TextButton( text: "Creditos", skin);

        background = new Image(game.getManager().get( fileName: "fondomenu.png", Texture.class));

        boton1 = new ImageButton(skin2.getDrawable( name: "botonjugar"));
        boton2 = new ImageButton(skin2.getDrawable( name: "botoncreditos"));
```

Pero en el GameScreen también tenemos world.

Variables que no se entienden bien aquí son el TextureAtlas y Skin el resto se explican por si solas, skin se refiere al tipo de textura que va a usar un botón o cualquier componente. El json, es una fuente de letras que he descargado para escribir texto.

Y hace falta instanciarlo con una skin para que se pueda usar.

TextureAtlas es un atlas de imágenes , son varias imágenes en una sola imagen y hay que instanciar la skin para que lo puedan recoger los ImageButton , si no, no funciona, porque te pide obligatoriamente ese parámetro.

Después nos encontramos con variables menos generales como son las de las entidades. Estos ya son los objetos en si con los que creamos los cuerpos en la simulación.

En la versión final no se encuentran porque no funcionan correctamente. Solo están los cuerpos pero la cosa con las entidades es que están divididas en 2 partes, ya que funcionan con 2 herramientas diferentes Box2D y Scene2D. La parte de Box2D es la parte física, que a su vez se divide en otras 2 partes , el cuerpo que se instancia y la fixtura, que es la estructura abstracta de un cuerpo que vamos a instanciar, es decir, cómo es el cuerpo previamente a ser creado.

Luego tenemos su parte gráfica de Scene2D, que se compone por Actores, estos son más simples y solo son para imágenes, el problema es luego unir ambas partes pero eso se explicará después.

En la GameScreen que es la del juego en sí.

Tenemos unos cuantos long para tener en cuenta el tiempo que ha pasado en el juego , startTime, tiene en cuenta el tiempo del sistema, y step, que va aumentando su valor conforme avanza la partida, este es muy importante porque cada step en el que llega a cierto valor, es el que hace que se generen las líneas en el juego.

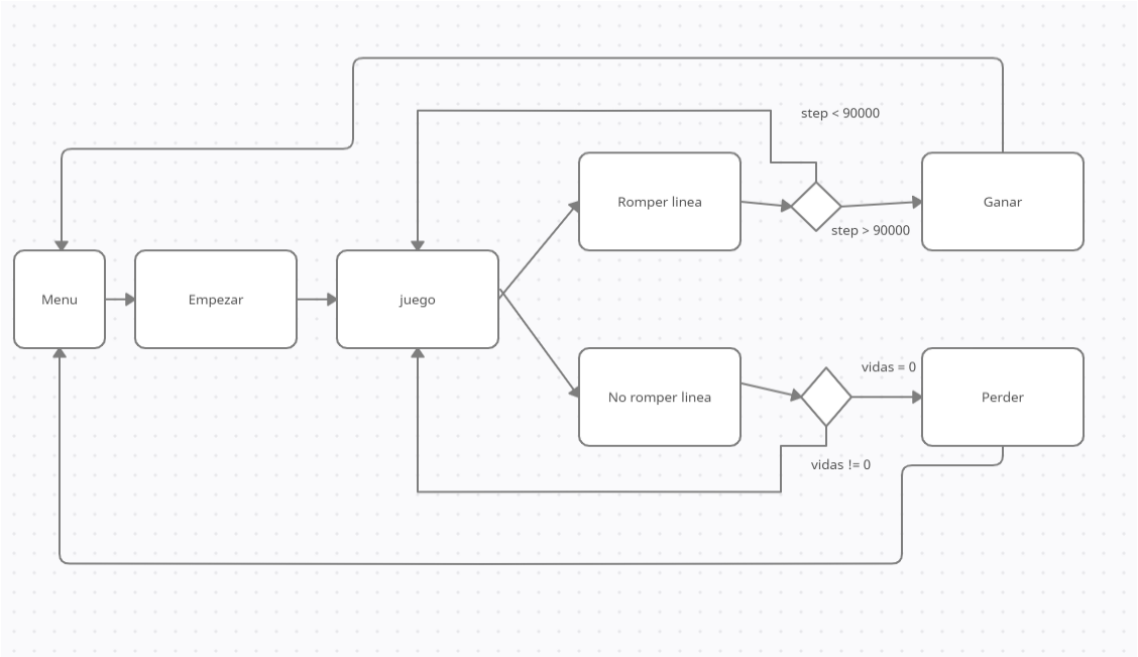
```
private final Vector2 mouseInWorld2D = new Vector2();
private final Vector3 mouseInWorld3D = new Vector3();

private Random r;

long startTime;

long step;


public GameScreen2(MainGame game){

    super(game);

    //Creacion de la pantalla
    stage = new Stage(new FitViewport( worldWidth: 2000, worldHeight: 1000));
    position = new Vector3(stage.getCamera().position);
    world = new World(new Vector2( x: 0, y: 0),  doSleep: true);
    background = new Image(game.getManager().get( fileName: "fondomenu.png", Texture.class));

    skin = new Skin(Gdx.files.internal( path: "skin/uiskin.json"));
    hpUI = new Label( text: "",skin);


    //Sonidos
    destroySound = game.getManager().get("audio/breakLine.mp3");
    backgroundMusic = game.getManager().get("audio/electroman.mp3");
    hploss = game.getManager().get("audio/losehp.mp3");

    //Camaras
    rendererBox2D = new Box2DDebugRenderer();        //Debug
    cam = new OrthographicCamera( viewportWidth: 16, viewportHeight: 9);             //Camara

    //Miscelaneo
    r = new Random();

    hpUI.setFontScale(5f);

    stage.addActor(hpUI);
```

La cámara es importante para centrarlo con la simulación

Recomiendo encarecidamente que se lean los comentarios del código ya que explican también algunas cosas

6.3. Estudio de la interacción del usuario con la aplicación

Juego:



6.4. Guía de estilo

Paleta cromática, son marrones y algunos rojos, sus códigos:

a38d8b

805652

4e201b

432520

4d231c

Las herramientas de pixel art provoca un degradado de colores que hace que sea una paleta muy grande como para hacer una escala que quepa aquí, pero todos giran en torno a 3 colores de base y de aquí se va difuminando según el gusto. Los pondré aquí abajo:

3e2723

4f211a

a38d8b

Luego a la hora de hacer los rectángulos siempre de anchura era x y altura x/2.

El fondo se rige por la misma regla.

En general son colores asociados con ladrillos.

6.5. Video demostrativo

https://www.youtube.com/watch?v=AWqMPGaaIXw&ab_channel=JuanRios

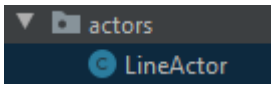
7. Codificación

Los módulos no me van a caber en un esquema ya que son unas 25 clases de las que hablar. Voy a hacer un resumen de ellos.

Bueno según las carpetas que tenemos en mi proyecto tenemos:

La carpeta Actors.

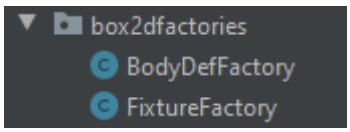
Aquí residen todos los actores, es decir todo lo que se pinta a la parte gráfica. Solo esta la línea aunque en versiones más antiguas también estaba el swiper pero como no se llegó a usar pues era residual puesto que se renderiza a parte con las clases del swiperresolver.



Ahí está el actor

Ahora vamos a hablar de las factories…

Como dice el nombre las factories son fabricas, tengo varias, y las uso para tener el código organizado, si no es imposible de llevar correctamente.

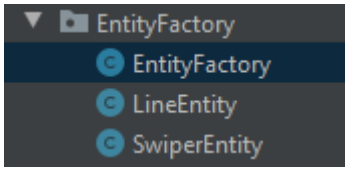


Ahí están las factories de box2d , la BodyDef crea los cuerpos en sí , y la FixtureFactory crea el modelo abstracto del que se tiene que basar el body.

Recuerdo que está explicado más detalladamente en el propio código.

El problema es que esta clase es sobre todo para debugeo, como tal no se utiliza ya, porque ahora la próxima factory de la que voy a hablar es de la entityfactory que es la que tiene las partes de box2d y scene2d. La anterior carpeta de Actors era la de scene2d

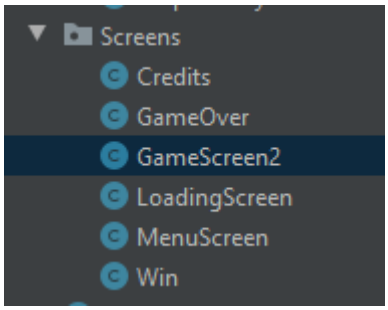
La entityfactory es la que se encarga de construirnos ya nuestros cuerpos con sus imágenes enteros tenemos 3 clases aunque la de swiper al final no se usa.

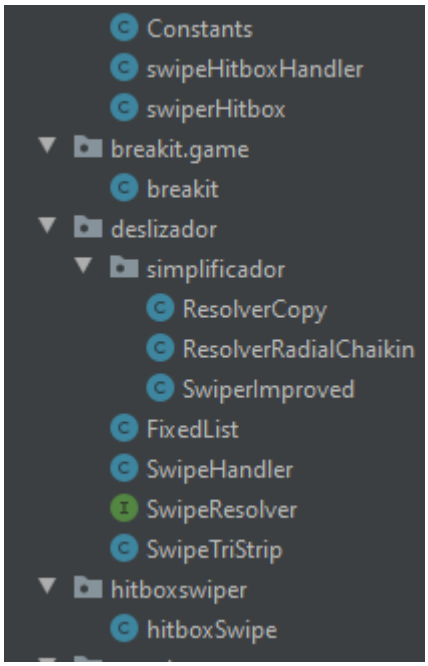


A la hora de crear nuestras entidades EntityFactory tiene un método createLine que crea la línea basándose en la clase LineEntity que es la constructora por así decirlo.

Este es el esquema general de fabricación bastante organizado y limpio.

Luego tenemos las pantallas , que son las diferentes partes donde va a viajar el usuario





Constants es para arreglar la proporción de Box2D y Scene2D.

SwipeHitboxHandler se ha usado para depuración y obtención de coordenadas para evaluar si funcionaba bien el swiper.

SwiperHitbox es un cuerpo que sigue el cursor o el dedo pulsado, el input listener se pone en el render del gamescreen.

ResolverCopy , es parte del proceso de simplificación de puntos del array del swiper.

ResolverRadialChaikin, es para suavizar las curvas del rastro que deja el swiper.

SwiperImproved es el motor principal del swiper, es también un modo depuración, para evaluar el comportamiento del swiper.

Fixed List es una modificación de ArrayList para que tenga un tamaño máximo.

SwipeHandler lleva todos los listener del swiper.

SwipeResolver es como el ResolverCopy.

SwipeTriStrip es el que lleva el array perpendicular , el color y el aspecto que tiene de rombo alargado el swiper.

7.1. Lenguajes, herramientas y metodologías empleadas

En la creación de esta aplicación sólo he utilizado Java como lenguaje de programación.

En cuanto a herramientas he utilizado bastantes más como:

1.- Android Studio, para realizar la aplicación multiplataforma.

2.- LibGDX, que es un framework para hacer videojuegos, del cual he usado Box2D basado en OpenGL, que es para la generación de una simulación física y de colisiones para los elementos necesarios en nuestro juego.

3.- LibGDX he utilizado Scene2D que es una herramienta para programar sobre todo la parte gráfica del juego, consiste en las imágenes, sonidos, animaciones y partículas que se pueden usar.

4.- Audacity para la edición de sonidos.

5.- Pixilart para la creación del arte del juego.

7.2. Aspectos destacables de la implementación

La verdad este proyecto ha pasado por muchas metas que había que superar y ha sido un aprendizaje continuo , en cuanto a Android, Java , desarrollo de videojuegos y LibGDX.

Por tanto, ha habido muchísimos aspectos destacables a la hora de desarrollar este proyecto.

1.- la creación del deslizador en el juego:

El deslizador es un elemento 2D el cuál sirve para mostrar al usuario que el juego está detectando su dedo en el juego y aparte siendo bastante estético para el tipo de juego que se ha realizado.

Ha supuesto una de las partes más difíciles porque había que tener en cuenta varias cosas.

- Que la imagen siguiera el dedo y dejará un rastro
- Tener en cuenta la velocidad a la que se traslada el dedo para hacer el rastro más grande o más pequeño
- Problemas a la hora de renderizar la imagen debido a cómo funciona el plano en 2D del juego.
- Problemas a la hora de implementar esto en el juego en sí.
- Que LibGDX no tiene tanto soporte en Android, como el que hay en la versión de Escritorio.

Bueno LibGDX cuenta con un método para detectar el input dentro de la pantalla.

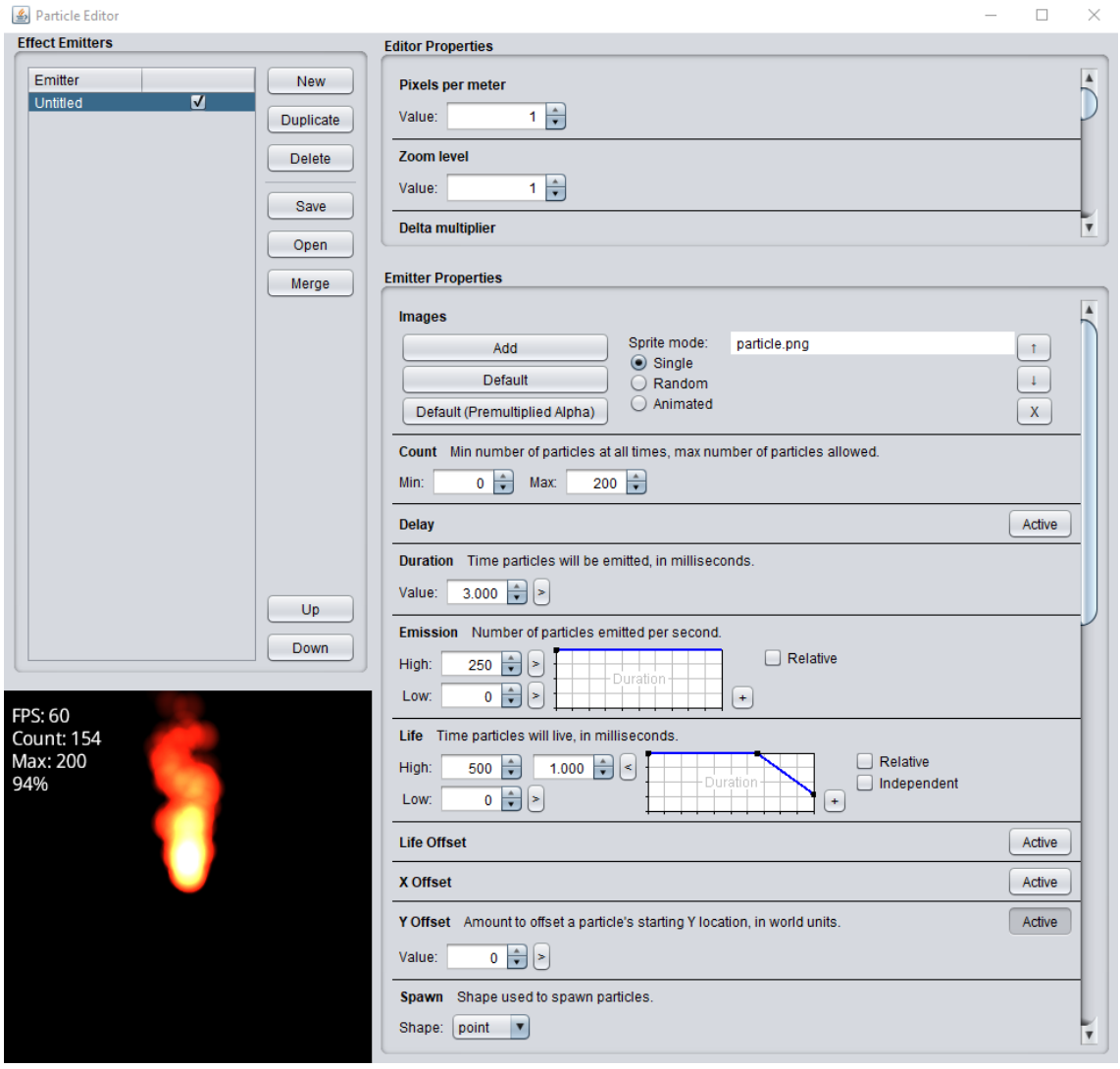
Que es LibGDX gdx.input.setinputprocessor . Que es la que se encarga de detectar el input del usuario, que se puede configurar.

A partir de ahí es solo hacer un get de la posición donde se ha realizado el input en la pantalla no?
Pero no es tan simple, hay que realizar un Array de puntos que hay que tener en cuenta y almacenarlos y en estos puntos hay que poner la imagen que queremos.

Esto de primeras daba resultados muy raros, el swiper parecía un juego de la serpiente sin serlo(Estaba basado en eso para realizarlo). Las diagonales las llevaba bastante mal a la hora de renderizarse, no hacia las diagonales.

Entonces se busco otras alternativas para solucionar este problema.

La primera opción fue la de usar partículas. Imágenes que aparecen durante un tiempo en la pantalla y luego desaparecen. Aquí una foto del editor:



Después de estar probando una herramienta para crear partículas de Libgdx. Y de intentar implementarlas sin resultado alguno.

Vemos que LibGDX para Android no tiene soporte para partículas, es decir, no se puede implementar en móviles.

Por tanto ¿Cómo iba a conseguir hacer el swiper? Es bastante difícil encontrar otras fuentes en LibGDX ya que hay una comunidad extremadamente pequeña como para encontrar soluciones a problemas de otros, es muy difícil encontrar las cosas en libgdx comparado con javascript o java puro por no decir que me he leído media documentación de libgdx para decirme que algunas cosas al final no están implementadas en Android. Libgdx esta limitandome

Buscando en otros lenguajes, vi una solución al swiper.

Consiste en un Array de posiciones que las va simplificando y haciendo un trazo entre las posiciones para pintar las imágenes.

También, se tiene en cuenta el número de iteraciones, es decir, la cantidad de puntos que se han dado en qué tiempo.

Esto es para determinar el tamaño del swiper (hay un limite de todas formas), cuantas menos iteraciones y más puntos se den, más grande el swiper.

Una cosa que no tenía puesta era la desaparición de este una vez se deja de pulsar, lo implemente yo.

La implementación del swiper fue relativamente difícil , porque primero había que comprender el código para luego no tener problemas, y lo más difícil las fórmulas matemáticas que utiliza.
Al principio supuso un bug visual en el que no se pintaban el resto de cosas en pantalla excepto el swiper y entonces no podía ver el comportamiento del juego en cuanto a colisiones o lo que mostrara.

La solución fue tener en cuenta el orden de pintado de los elementos y tener en cuenta que se pinta.

El primer elemento que se pinte ira por detrás de los siguientes que se pinten en pantalla.

Si no se le pone el método draw a los elementos que se van a pintar , no se van a pintar en pantalla.

Este bug luego me sirvió para solucionar otro similar a la hora de pintar las líneas con la colisión. Pero ya lo explicaré en su apartado adecuado

Problemas con la parte física del swiper:

Una cosa a tener en cuenta es que el swiper , lo que es la colisión para que siguiera el input de donde pulsaras en la pantalla. Lo cambiaba de posición todo el rato, es decir, lo teletransportaba a la posición del input. Y cuando se dejaba de pulsar iba a una posición de fuera de pantalla a x:-100 , y:-100.

Según la documentación de LibGDX realizar un teletransporte es mala práctica y es malo en general.
Se recomienda usar el comportamiento físico para mover las cosas, mediante impulsos.

Pero yo creo que en mi caso no hay otra manera …

Para tener en cuenta donde realiza el próximo input el jugador mediante impulsos físicos habría que calcular la distancia, dar el impulso exacto para que llegue y pararlo en cuanto llegue, con lo que le quede de velocidad lineal… Pero si se da un nuevo input sin llegar al primero ya daría problemas, que también se puede solucionar pero puede seguir dando un comportamiento impredecible a causa de que se mueve por pantalla por impulsos.

No puedo hacer “buenas prácticas” ya que seria ridículamente difícil lograr lo que pide la documentación y seguramente acabe siendo un caos no funcional.

Así que al final teletransporte, más sencillo y no me ha dado problemas.

Otros problemas de implementación fue a la hora de realizar las colisiones.

Inicialmente cuando chocaba el swiper (colisión) con la línea(colisión), se detectaban entre ellos pero tenían un comportamiento físico y no destruía la línea. Con comportamiento físico quiero decir que cuando se tocaban la línea se movía con momento lineal por un momento y después paraba. Cosa que no queremos, no queremos un comportamiento físico real, aunque libgdx , box2d está especialmente preparado para esas cosas.

Pero el problema no era ese, si no ya al intentar colisiones y que se provocara la destrucción como tal del cuerpo, hacia que el juego entero se rompiera y se cerrara solo.(A veces hasta el emulador dejó de funcionar)

A la hora de generar las líneas LibGDX recomendaba la práctica de hacer una lista de objetos que se vayan creando y cuando termine con ellos lo destruye.

Inicialmente yo lo tome con esta trayectoria pero vi que era bastante complicado para un juego tan simple. Incluso creo que no tenía nivel de programación para realizarlo como lo pide.

Ya que había que tener una entidad con su parte de Box2D y Scene2D unida , con un ID y todo y decidí optar por otra solución porque esto para crearlas , destruirlas es horrible tanto para código como a nivel de optimización.

Entonces la solución consistía en que la línea realmente no se destruye, si no que cambia a posiciones negativas cada vez que la tocas.

Daba apariencia de que desaparecía pero no es así.

Simplemente un boolean de destruido y cuando el step llegará a lo que es el siguiente paso vuelve a true y se le da una nueva posición y ángulo para volver a ser destruida y así en un ciclo.

La verdad no me imaginaba ingeniármelas así para la generación de líneas pensaba que iba a ser bastante más complicado.

Otro punto importante es el de como uní la colisión del swiper con lo que es la imagen.

Realmente están separadas , nunca se llegan a unir como tal en una entidad.

Pero como ambas siempre van a la posición del dedo basándose en un input , pues no hacia falta unirlas , porque ya por posición parecen unidas.

Por último, algo que no termino de entender como lo desarrollaron así los de LibGDX es las medidas en Scene2D y Box2D.

Box2D usa “metros” para medir las distancias en la simulación

Y Scene2D usa pixeles para lo mismo.

Hice una clase Constants, que realmente es una variable final para hacer la conversión teórica de metros a pixeles. Pero esto no funciona en todos los teléfonos por tanto con libgdx **no se puede hacer una aplicación apta a todos los formatos**.

También es importante que para que la hitbox del swiper siga al cursor la cámara tenemos que quitarle la proyección momentáneamente para reevaluar las coordenadas a unas absolutas al móvil y no relativas a la cámara porque si no, no se posiciona bien

7.3. Uso de estándares y normas de accesibilidad

No tiene uso de estándares .

8. Despliegue de la aplicación

La aplicación para que funcione correctamente se debe utilizar el formato estándar de tamaño de móvil que respete un aspect ratio de 16:9.

Pero a día de hoy eso no funciona bien, además de que LibGDX lleva muy mal este aspecto ya que gráficamente en mi caso empezó a fallar mucho el juego. Y también por la falta de soporte que tiene en Android. A veces hace comportamiento impredecible. Que explicaré más adelante y que no pude solucionar. Pero fuera de eso el juego es perfectamente funcional , quizá lo que ocurra es que se generen líneas fuera de sitio cuando probéis el juego o bien las imágenes estén descolocadas.

He hecho todo lo posible para ponerlo bien, desde coger la anchura y altura del móvil como referencia , a hacer yo mi propio altura y anchura , mantener el aspecto ratio , quitándolo… no encontraba forma pero se porque falla.

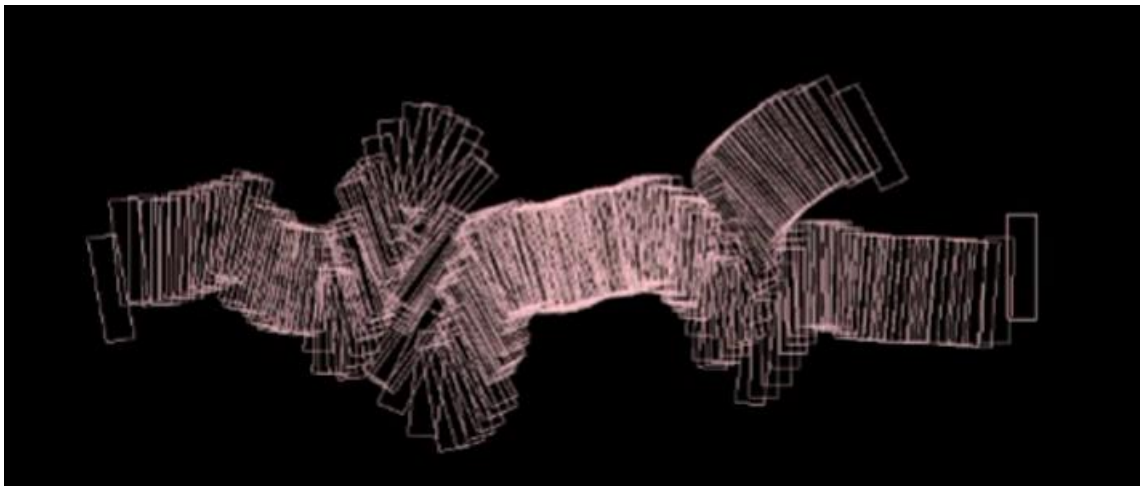
9. Evaluación y pruebas

Problemas presentados en el proyecto

- Colisiones y destrucción de otro elemento debido a que LibGDX no esta sincronizado .

Tenemos un método que es de world que es el world.step. Básicamente es el equivalente de pasar por un frame en una simulación.

Cuando intentamos destruir un objeto por colisión el step no ha llegado a pasar y LibGDX no controla el solo la destrucción de objetos, no tiene una cola de objetos a destruir cuando haga falta, se la tienes que hacer tú , esto lo encontré al final en la documentación después de **días** buscando una solución también me daba "comportamientos impredecibles" muy curiosos como:



Se multiplicaba la línea en vez de borrarse infinitamente hasta que moría el móvil…

Esto es que no puse bien la destrucción del objeto y no llegaba nunca a borrar.

La solución consiste en poner la destrucción del body después del world.step y no antes porque el juego deja de funcionar.

Para ello ponemos un boolean de destroyed = false, y cuando ocurre la colisión lo ponemos en true, así cuando llegue ya al world.step hay un if que si es true, ahí ya destruye la línea, aunque realmente no se destruye como tal.

- Crasheo de aplicación por colisión

La aplicación dejaba de funcionar cuando colisionaban, no solo rompes la línea ,también rompes el juego de ahí nació el nombre del juego , lo he llamado BrokeIt porque no paraba de crashear y buggearse desde el principio.

Se solucionó con el apartado anterior

- No se ve nada de lo pintado

No se pintaba nada en pantalla y tampoco pillaba las colisiones pero si se mostraba el cursor.

Después de días investigando resulta que cambié el valor de vista de la cámara. Es decir tenía la cámara en otro sitio a donde debería estar y no podía ver la simulación de ninguna manera.

- Parte gráfica

LibGDX no solo le tienes que decir cuando destruir las cosas, también le tienes que decir cuando las vas a cargar y no tocar nada hasta que termine de cargar porque si no te da un error de memoria. Porque LibGDX es completamente asíncrono y si le pones varias cosas a hacer , te las va a mandar desordenadas según las vaya terminando es decir si muestra antes la imagen y no la tiene cargada, error de memoria.

De ahí viene una de las cosas que más me siento orgulloso.

La pantalla de carga. Es simple y me salva de que deje de funcionar la aplicación,

Su función es obvia , cargar las cosas antes de mostrarlas para evitar ese error de memoria.

La pantalla de carga solo tiene un label, con texto que tiene en cuenta como va progresando la carga y el fondo negro es el que le he puesto estándar a libgdx. Una vez termina cambia a la pantalla del Menú y no da más problemas.

- Error por ImageButtons

Los imagebutton no son como los de Android, son mucho más tediosos porque te fuerzan el uso de un atlas para funcionar. Ignorante de mí intenté poner texturas normales y no funcionaba hasta que vi en la documentación que se les pone un Atlas a partir de una skin instanciada.

Que como ya hemos explicado antes un Atlas es una imagen que tiene varias imágenes dentro.

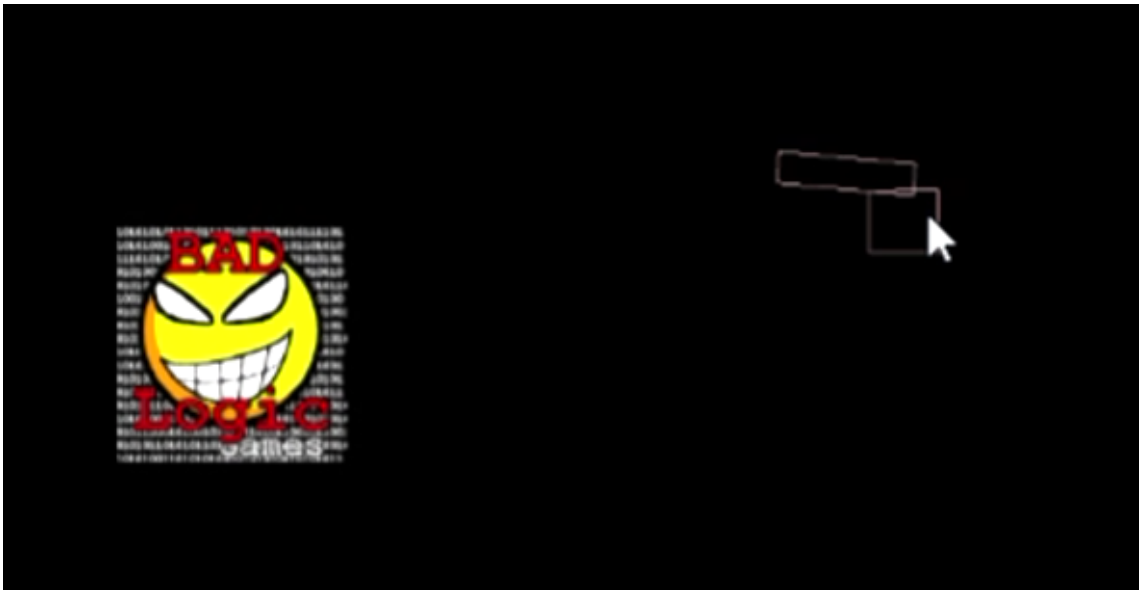
Se pasa a skin porque el skin sirve para referenciar ciertas partes de una textura o fuente o similares.

Como el Atlas son varias partes tu lo utilizas para referenciar solo la parte que te interesa mostrar en cierto componente.

Para crear el atlas tuve que usar una herramienta de libgdx que es el atlasgenerator.

Y es muy sencillo de usar.

Errores sin solucionar:

1. Recordando que hay problemas a la hora de las distancias entre Box2D y Scene2D , viene un error al hacer las entidades con las 2 partes unidas.
   Por muchas proporciones que hagas , por mucho que respetes el ratio se te va a descolocar siempre.
   Porque la proporción por pixeles tiene un problema y es que no todos los teléfonos tienen la misma cantidad de pixeles, algunos son más largos que otros y otros más anchos.
   Esto supone el problema de que nunca la imagen va a ser fiel a la simulación.
   Por tanto le das al ladrillo que quieres romper y no se rompe porque la colisión está en otro lado.
   La solución no la he encontrado pero si el problema, la cosa es que se puede solucionar de 2 maneras, 1 no usando Box2D y usar Ashley (otro motor) porque no esta bien soportado en Android, y 2 la parte de imagen de las líneas eliminarla y usar el debugrenderer (una cámara especial para ver las colisiones) como cámara estándar para los niveles.
   Esto resulta poco estético porque también se ve la hitbox del cursor y la verdad no me gusta nada. Pero al menos es visible lo que estás jugando y se ve la funcionalidad y como funciona el proyecto.
   Aquí un ejemplo de imagen que debería estar encima de la línea se encuentra bastante lejos de donde debería estar, cuando mueves la línea se descoloca más, este comportamiento muestra que no mantiene una posición exacta aunque devuelvas el cuerpo a donde estaba antes, no es proporcional, la proporción no funciona.
   Entonces al no funcionar la entidad , ya que la parte de Scene2D no coincide con la de Box2D. Estaba usando cuerpos más que entidades.
   

2. Nunca cierres la virtualización del móvil con mi juego abierto. Mi ordenador dejó de funcionar y se quedó congelado las 2 veces que lo hice y pensé que había perdido el proyecto.
3. El swiper ahora es de color negro, cuando en los parámetros está de color violeta.
4. La vida, cuando tocas en pantalla se mueve hacia arriba, esto no pasa si no tiene fondo, pero si le pones fondo detrás se mueve … comportamiento impredecible , creo que es

porque libgdx no lleva bien Android en general porque no tiene sentido que sea por tocar y teniendo fondo se mueve hacia arriba un poco, y si no hay fondo no pasa.

10. Manual de usuario

10.1. Descripción de la aplicación

Tenemos ante nosotros un juego 2D pixelado cuyo objetivo es romper unas líneas en pantalla para ganar el juego.

10.2. Funcionalidad y características

- Para iniciar el juego dale al botón de jugar.
- Aparecerán líneas en pantalla conforme suena la música, rómpelas para no perder vidas.
- Si pierdes todas tus vidas tendrás que empezar de 0.
- Si ocurre solo dale a volver a jugar o volver al menú según lo que prefieras.
- Dale al botón de créditos si quieres saber quién hizo el juego.

11. Conclusiones

El LibGDX aunque se diga que es un framework para principiantes… A mi no me lo ha parecido. Es un framework complicado, en el que hay que escribir mucho código porque haces muchas cosas de cero. Comparándolo con Unity este ya te las da hechas. Es también muy delicado y estricto.

Un proyecto de este calibre con Unity hubiera tardado una semana, sin embargo, con LibGDX he tardado mes y medio.

Teniendo en cuenta, que previamente estuve aprendiendo a usar el framework con un tutorial haciendo una copia de Geometry Dash.

Este framework no lo recomiendo sin conocimientos intermedios de Java. A mi me ha costado bastante realizar este proyecto.

Es muy importante estar habituado a mirar la documentación de lo que estás usando y saber dónde buscar ya que no es una práctica que tuviera aprendida o desarrollada de antes porque en clase es bastante más fácil. Pero cuando estás solo con un framework que no usa nadie…

Te toca aprender por tu cuenta , mirar la documentación investigar y probarlo todo.

Android prácticamente no lo he usado y no podía echar mano de su documentación o de problemas que ha tenido gente similares a los míos.

Ha sido, sin lugar a dudas, una aventura demasiado temeraria y difícil pensando en que este año de pandemia ha influido en mi aprendizaje.

He llegado a dudar en algún momento, poder finalizar este proyecto a tiempo. Pero he conseguido terminarlo en el tiempo establecido y las metas cumplidas.

Lo principal ha sido aprender en este camino, lo cual me enorgullece.

Incluso me he metido en la comunidad de LibGDX de discord para ver como hacían algunas cosas ya que no hay mucho de donde tirar al ser poco usado este framework.

Una nota a tener en cuenta es que LibGDX haces tu las cosas que Unity ya te da por echas, es decir que al final lo que he aprendido es como funcionan también por detrás algunos motores gráficos (más o menos porque no es igual).

En conclusión no voy a volver a programar con LibGDX en móvil.

No tiene buen soporte para móvil.