EBook Gratis

APRENDIZAJE libgdx

Free unaffiliated eBook created from
Stack Overflow contributors.

#libgdx

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: libgdx

It is an unofficial and free libgdx ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official libgdx.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con libgdx

Versiones

Versión	Fecha de lanzamiento
1.9.3	2016-05-16
1.9.5	2016-12-11

Examples

Instalación o configuración

LibGDX tiene una configuración bastante simple, con la ayuda de un programa Java simple. Puedes encontrar la descarga aquí . Cuando inicies la aplicación, se verá algo así:

Nota: esta captura de pantalla se ha tomado en Linux y muestra una ruta que difiere de una instalación de Windows. Sin embargo, el formulario es el mismo en cualquier sistema operativo compatible con esta aplicación de configuración

Cajas de entrada

En el cuadro de entrada "Nombre", está el nombre del juego para tu proyecto. El cuadro de entrada "Paquete" es el paquete para su clase principal. El cuadro de entrada "Clase de juego" es la clase principal a la que se llama cuando se ejecuta el juego. El cuadro de entrada "Destino" es el destino donde se generará el proyecto. El cuadro de entrada "Andriod SDK", la ruta a donde se encuentra su SDK de Android. Este cuadro de entrada es completamente opcional, por lo que si no desea implementar su aplicación en Android, no tiene que preocuparse por esto.

Sub Proyectos

Los subproyectos son solo las plataformas que se van a implementar. Esa parte es bastante autoexplicativa. Si no desea implementarlo en HTML, por ejemplo, simplemente desactive la casilla de verificación.

Extensiones

Las extensiones son las extensiones oficiales de LibGDX. Aquí hay una tabla que te dice lo que cada uno es:

Nombre de extensión	Descripción
Bala	Bullet es una biblioteca de detección de colisiones 3D y dinámica de cuerpos rígidos.
Freetype	Freetype le permite usar fuentes .ttf, en lugar de tener que usar fuentes de mapa de bits
Herramientas	Le permite implementar la salida de herramientas LibGDX.
Controladores	Le permite implementar controladores como los controladores XBox 360.
Box2d	Una biblioteca de física para juegos 2d.
Box2dlights	Permite una manera fácil de agregar luces dinámicas suaves a un juego de física.
Ashley	Un pequeño marco de entidad
Ai	Un marco de inteligencia artificial.

Puede agregar Extensiones de terceros, pero sus detalles o nombres no se mostrarán aquí.

Avanzado

En la sección Avanzado puede establecer varias configuraciones y generar archivos de proyecto adicionales para Eclipse e IDEA IDE.

Nombre de la configuración	Descripción
Maven Mirror URL	Reemplaza a Maven Central con la URL de Maven

Nombre de la configuración	Descripción
	proporcionada
IDEA	Genera archivos de proyecto IDEA Intellij
Eclipse	Genera archivos de proyecto de Eclipse.
Modo offline	No forzar dependencias de descarga

Generacion

Una vez que tenga todos los ajustes correctos, puede presionar el botón "Generar". Esto puede tardar un par de segundos, pero generará los archivos básicos, y Gradle es necesario para su proyecto. Una vez que haya terminado con eso, es hora de importar el proyecto a su IDE.

Hola mundo libGDX

Lo esencial

Los proyectos generados contienen una aplicación básica similar a Hello World ya implementada.

El proyecto principal es el proyecto principal, que contiene todo el código independiente de la plataforma. Esto es obligatorio, pero en función de la configuración de su generación, puede tener varios proyectos más para cada plataforma que seleccionó.

El ejemplo

Abra `com.mygdx.game.MyGdxGame.java` en el proyecto `core` . Verás el siguiente código:

```
public class MyGdxGame extends ApplicationAdapter {
    SpriteBatch batch;
    Texture img;

    @Override
    public void create () {
        batch = new SpriteBatch();
        img = new Texture("badlogic.jpg");
    }

    @Override
    public void render () {
        Gdx.gl.glClearColor(1, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
        batch.begin();
        batch.draw(img, 0, 0);
        batch.end();
    }

    @Override
    public void dispose () {
```

```
        batch.dispose();
        img.dispose();
    }
}
```

Aunque este es su proyecto principal, no lo ejecutará directamente, siempre tiene que ejecutar el Lanzador específico de la plataforma, para el escritorio se llama `com.mygdx.game.desktop.DesktopLauncher.java` en el proyecto de `desktop` .

```
public class DesktopLauncher {
    public static void main (String[] arg) {
        LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
        new LwjglApplication(new MyGdxGame(), config);
    }
}
```

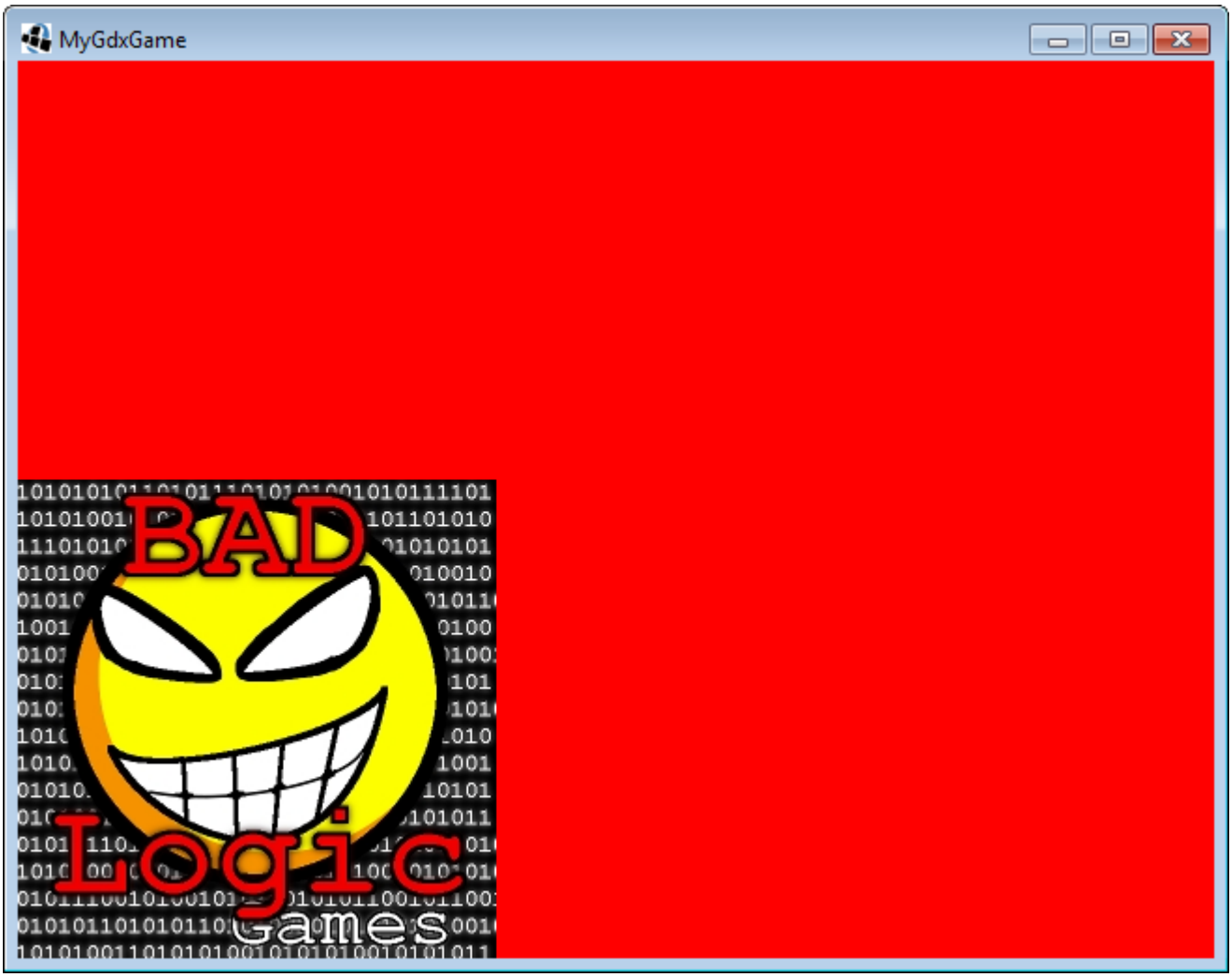
En esta clase puede establecer configuraciones específicas de la plataforma.

Corriendo tu juego

Eclipse e Intellij utilizan dos métodos diferentes para ejecutar su proyecto. Encuentra el IDE que estás usando a continuación.

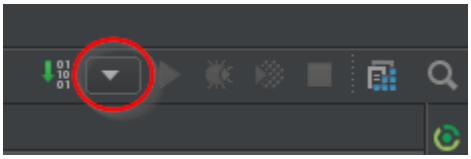
Ejecutando usando Eclipse

Usando eclipse, puede ejecutar su aplicación ejecutando esta clase como una aplicación Java (haga clic con el botón derecho en el proyecto -> Ejecutar como -> aplicación Java). Verás la siguiente ventana:

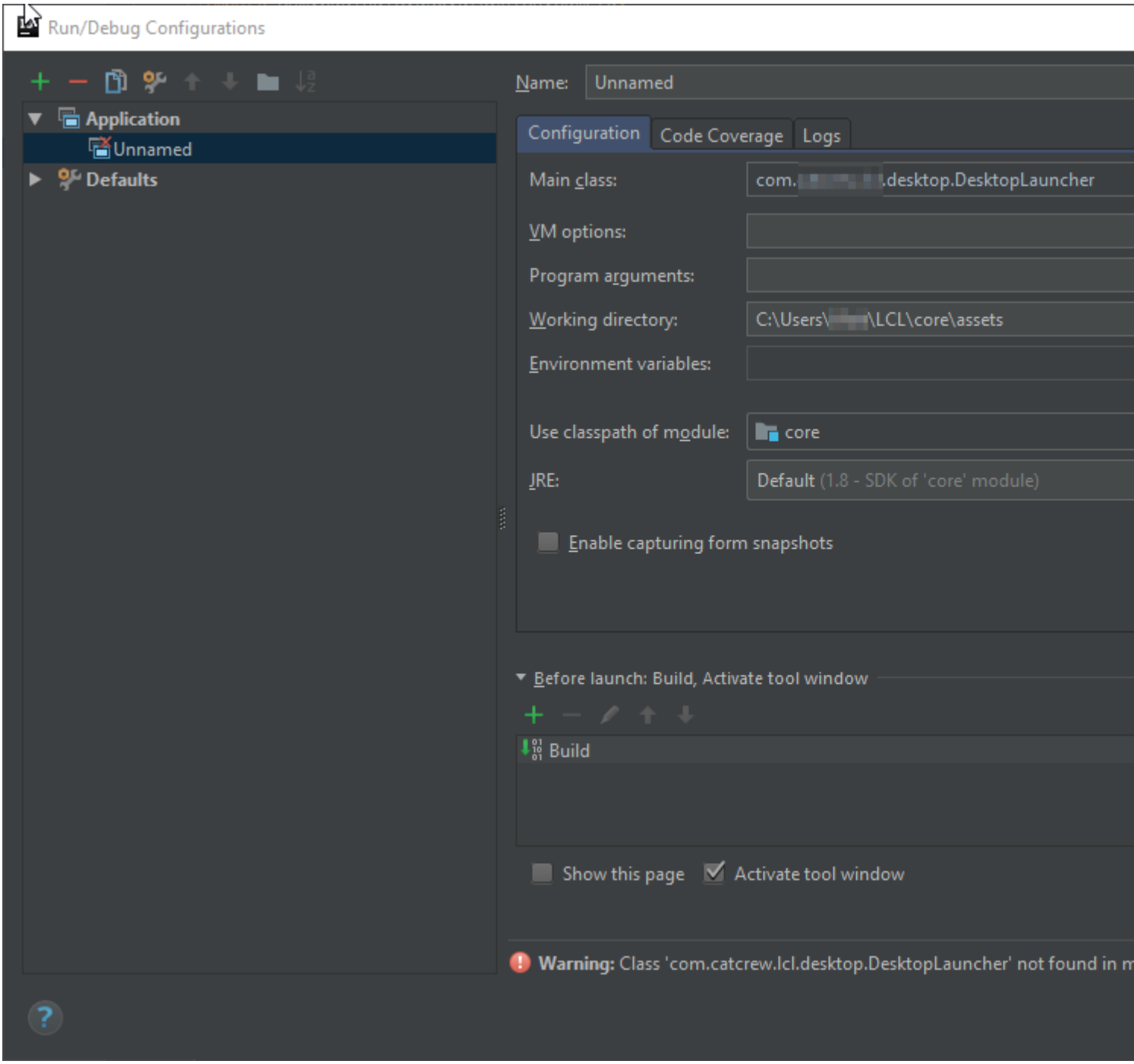


Correr con la idea de Intellij

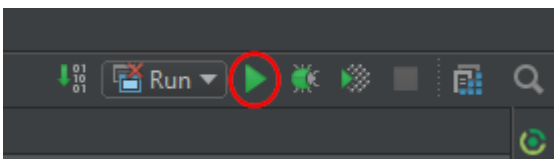
En Intellij, necesitarás hacer una configuración de ejecución. Para hacer esto, debes hacer clic en el botón en la esquina superior derecha que parece una zanahoria hacia abajo:



Luego haga clic en el botón "Editar configuraciones ...". Aparecerá una pantalla con todas las configuraciones de ejecución actuales. Haga clic en el "+" en la parte superior izquierda de esa ventana y seleccione la opción "Aplicación". Desde allí, seleccione el archivo "DesktopLauncher" para la opción "Clase principal", configure "Usar classpath del módulo" a la opción principal y configure "Directorio de trabajo" en la carpeta de activos en su carpeta principal. El producto final debe verse algo como esto:



Una vez que haya hecho eso, puede seleccionar un nombre para su configuración de ejecución y hacer clic en "Aplicar", luego "Aceptar". Una vez que haya terminado con eso, puede hacer clic en el icono de ejecución verde en la parte superior derecha:



¿Qué está pasando en la clase MyGdxGame?

Primero se llama al método de `create` , que inicializará el lote que se dibuja en la pantalla. Luego

el método carga el archivo badlogic.jpg en la memoria.

Después de esto, se llama repetidamente al método de `render` hasta que se detiene la aplicación. Este método restablecerá el color de fondo a rojo y dibujará la imagen en la pantalla. Como puede ver, siempre tiene que comenzar y finalizar el dibujo por lotes. Por último, cuando la aplicación está a punto de detener el método de `dispose` , se liberará el espacio de memoria utilizado por la textura y el lote.

(Es bueno saber que la eliminación puede ocurrir en tiempo de ejecución también en otras plataformas, por ejemplo, Android, ya que Android puede liberar espacio de memoria cuando las aplicaciones están en segundo plano, pero este es un tema más avanzado)

Observación: ¡ Si obtiene un error al ejecutar como se muestra a continuación, verifique esta pregunta para obtener respuesta!

> Archivo no encontrado: badlogic.jpg (interno)

Descripción general de LibGDX

LibGDX es una biblioteca de desarrollo de juegos de código abierto y gratuita desarrollada en Java. Sus objetivos son permitir a los usuarios desarrollar juegos multiplataforma que se ejecutan en navegadores de escritorio, Android, iOS y web. Escribir código una vez, desplegarlo en cualquiera de las plataformas principales.

Agregando soporte para código específico de la plataforma

LibGDX está diseñado de manera que pueda escribir el mismo código y desplegarlo en varias plataformas diferentes. Sin embargo, hay ocasiones en las que desea obtener acceso a un código específico de la plataforma. Por ejemplo, si tiene tablas de clasificación y logros en su juego, es posible que desee utilizar herramientas específicas de la plataforma (como Google Play Games) además de almacenarlas localmente. O quieres usar una base de datos, o algo completamente diferente.

No puede agregar este tipo de código en el módulo principal. Entonces el primer paso es crear una interfaz. Crealo en el módulo central. Este primero es una utilidad para gestionar otras interfaces:

```
public interface PlatformWrapper{
    //here you can also add other things you need that are platform specific.
    //If you want to create your own file saver for an instance, a JSON based database,
    //achievements, leaderboards, in app purchases and anything else you need platform
specific code for.
    SQLWrapper getSQLSaver();//This one will be used in this example
    AchievementWrapper getAchievementHandler();//this line is here as an example
}
```

Entonces, necesitamos crear una segunda interfaz, el SQLWrapper. Éste también va en el

módulo central.

```
public interface SQLWrapper{
    void init(String DATABASE);
    void saveSerializable(int id, Object o);
    Object loadSerializable(int id, Object o);
    void saveString(int id, String s);
    //.... and other methods you need here. This is something that varies
    //with usage. You may not need the serializable methods, but really need a custom method
for saving
    //an entire game world. This part is entirely up to you to find what methods you need

    //With these three methods, always assume it is the active database in question. Unless
    //otherwise specified, it most likely is
    String getActiveDatabase();
    void deleteDatabase();
    void deleteTable(String table);//delete the active table
}
```

Ahora, debe ir a cada proyecto y crear una clase para implementar PlatformWrapper y una para implementar SQLWrapper. En cada proyecto se agrega el código necesario, como instancias, constructores, etc.

Una vez que haya anulado todas las interfaces que creó, asegúrese de que todas tengan una instancia en la clase que implemente PlatformWrapper (y que haya un captador). Finalmente, cambias el constructor en la clase principal. La clase principal es la clase a la que hace referencia desde el iniciador. Extiende ApplicationAdapter, implementa ApplicationListener o extiende el juego. Edite el **constructor** y agregue PlatformWrapper como un argumento. Dentro de la envoltura de la plataforma tiene algunas utilidades (si agregó alguna) además de todas las otras envolturas (sql, logros, o cualquier otra cosa que haya agregado).

Ahora, si todo está configurado correctamente, puede hacer una llamada a PlatformWrapper y obtener cualquiera de las interfaces multiplataforma. Llame a cualquier método y (suponiendo que el código ejecutado sea correcto) verá en cualquier plataforma, hace lo que se supone que debe hacer con el código específico de la plataforma

Gestor de activos

El AssetManager es una clase que le ayuda a administrar sus activos.

En primer lugar, necesitas crear una instancia:

```
AssetManager am = new AssetManager();
```

Después de que esto se haya inicializado, y antes de renderizar algo, desea obtener los recursos:

```
am.load("badlogic.jpg", Texture.class);//Texture.class is the class this asset is of. If it is
a
//sound asset, it doesn't go under Texture. if it is a 3D model, it doesn't go under
Texture.class
//Which class added depends on the asset you load
```

```
//... other loading here ...//

//when finished, call finishLoading:
am.finishLoading();
```

Ahora, donde sea que quieras tener `badlogic.jpg` :

```
Texture texture = am.get("badlogic.jpg");
//Ready to render! The rendering itself is in the normal way
```

El uso de AssetManager le permite cargarlos una vez en la memoria del AssetManager y luego obtenerlos tantas veces como desee.

¿Por qué usar AssetManager? (de la wiki):

AssetManager (código) le ayuda a cargar y administrar sus activos. Es la forma recomendada de cargar sus activos, debido a los siguientes comportamientos agradables:

- La carga de la mayoría de los recursos se realiza de forma asíncrona, por lo que puede mostrar una pantalla de carga reactiva mientras se cargan las cosas
- Los activos son referencia contados. Si los dos activos A y B dependen de otro activo C, C no se eliminará hasta que A y B se eliminen. Esto también significa que si carga un activo varias veces, en realidad se compartirá y solo ocupará memoria una vez.
- Un solo lugar para almacenar todos sus activos.
- Permite implementar de manera transparente cosas como cachés (ver FileHandleResolver más abajo)

El gráfico de escena 2D

Cuando comienzas con Java o Android, aprendes rápidamente que (0,0) está en la esquina superior izquierda. En LibGDX, sin embargo, (0,0) está por defecto en la esquina inferior izquierda.

Usando una cámara ortográfica, puede obtener (0, 0) estar en la esquina superior izquierda. Aunque por defecto, (0, 0) está en la esquina inferior izquierda. Esto es algo que es importante saber, ya que también cambia qué esquina de las texturas tienen las coordenadas X e Y.

Lea Empezando con libgdx en línea: https://riptutorial.com/es/libgdx/topic/1368/empezando-con-libgdx

Capítulo 2: Apoyo a múltiples resoluciones

Examples

Miradores

Para admitir múltiples resoluciones y relaciones de aspecto, Libgdx usa los llamados `Viewports` .
Hay algunos tipos de `Viewports` que utilizan diferentes estrategias para manejar múltiples resoluciones y relaciones de aspecto.
Una `Viewport` utiliza una `Camera` debajo del capó y administra su `viewportHeight` de `viewportHeight` de `viewportHeight` y `viewportWidth` . Opcionalmente, puede darle a la `Viewport` una `Camera` en su constructor, de lo contrario usará una `Camera` `OrthographicCamera` por defecto:

```
private Viewport viewport;
private Camera camera;

public void create() {
   camera = new PerspectiveCamera();
   viewport = new FitViewport(8f, 4.8f, camera);
}
```

Además, debe especificar `worldWidth` y `worldHeight` para el constructor de la ventana `worldHeight` . Este espacio representará el sistema de coordenadas virtual que utilizará para especificar la posición de los objetos que se dibujarán y los tamaños. La transformación de la ventana gráfica, que se puede aplicar a un SpriteBatch, por ejemplo, se encargará automáticamente de transformar las coordenadas lógicas en coordenadas reales de la pantalla, de una manera que se ajuste al tipo real de ventana gráfica que esté utilizando. Por ejemplo, al usar una proyección ortográfica (`OrthographicCamera` , que es la predeterminada): si su tamaño del mundo es 12.8 por 7.8 metros y la pantalla de su dispositivo es de 2560x1560 píxeles, entonces su mundo se proyectará dinámicamente con 200 píxeles por metro.

Cuando cambia el tamaño de la `Viewport` (por ejemplo, si la orientación de la pantalla del teléfono inteligente está cambiando), debe informar a la `Viewport` sobre ese cambio. luego actualizará automáticamente la `viewportHeight` la `Camera` la `viewportHeight` y la `viewportWidth` `viewportHeight` `viewportWidth` :

```
public void resize(int width, int height) {
    viewport.update(width, height);
}
```

Los `Viewport` también administran el `OpenGL` `Viewport` y modifican el área de dibujo según sea necesario.

Los `Viewport` también se encargan de convertir las coordenadas de la pantalla en coordenadas del juego, lo que se necesita especialmente para la selección:

```
private Vector2 worldPos = new Vector2();
```

```
public boolean touchDown (int x, int y, int pointer, int button) {
    worldPos.set(x, y);
    viewport.unproject(worldPos);
    processPicking(worldPos);
}
```

Viewports integrados

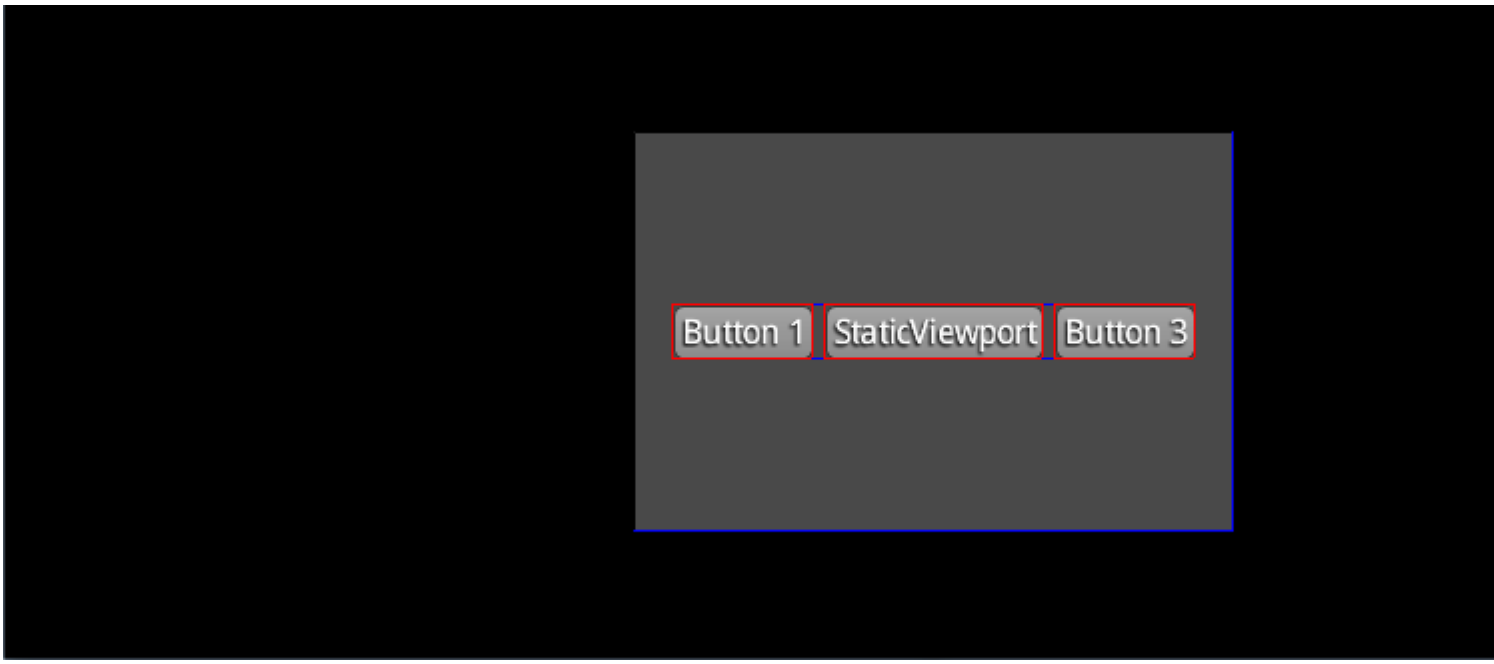
Hay un par de viewports incorporados, que cada uno hace cosas diferentes. Aquí hay una lista de los nombres y sus descripciones:

Nombre de la ventana gráfica	Descripción
StretchViewport	Estirará la pantalla. No hay barras negras, pero la relación de aspecto puede estar desactivada.
FitViewport	Maximizará su tamaño basado en la relación de aspecto. Podría tener barras negras.
FillViewport	Es lo mismo que un FitVieport, pero siempre llena la pantalla completa.
ScreenViewport	Siempre llena la pantalla completa, pero no cambia el tamaño de ningún niño.
ExtendedViewport	Mantiene la relación de aspecto del mundo sin barras negras al extender el mundo en una dirección
CustomViewport	Una vista programada personalizada. Puede tener barras negras y mantener la relación de aspecto.

Viewports personalizados

Usted puede hacer su propio viewport personalizado. Puede tener barras negras y puede o no mantener su relación de aspecto, dependiendo de cómo lo programes. Una vista personalizada se vería así:
`public class Tutorial extends Viewport` . Necesitaría anular la `update(width, height)` , pero eso es todo.

Otro intento sería extender el `ScalingViewport` genérico y proporcionar otro Scaling que aún no se haya suministrado. Supongamos que lo establece en `Scaling.none` . Eso se vería algo así:



Si desea alguna referencia, puede encontrar las clases de viewport integradas aquí mismo .

StretchViewport

El `StretchViewport` es un `Viewport` tipo, que apoya un tamaño de pantalla virtual.
Esto le permite definir un ancho y una altura fijos (independientes de la resolución).
El `StretchViewport` , como su nombre lo indica, estira la pantalla virtual, si la relación de aspecto virtual no coincide con la relación de aspecto real. El OpenGL Viewport no se modificará y no aparecerán barras negras.

Uso:

```
private Viewport viewport;
private Camera camera;

public void create() {
    camera = new OrthographicCamera();
    viewport = new StretchViewport(80, 48, camera);
}

public void resize(int width, int height) {
    viewport.update(width, height);
}
```

FitViewport

FitViewports son viewports que siempre mantienen la relación de aspecto. Lo hace creando barras negras en los bordes donde queda espacio. Este es uno de los viewports más utilizados.

Uso:

```
private Viewport viewport;
private Camera camera;
```

```
public void create() {
    camera = new OrthographicCamera();
    viewport = new FitViewport(80, 48, camera);
}

public void resize(int width, int height) {
    viewport.update(width, height);
}
```

Lea Apoyo a múltiples resoluciones en línea: https://riptutorial.com/es/libgdx/topic/4219/apoyo-a-multiples-resoluciones

Capítulo 3: Ashley Entity System

Observaciones

Ashley Entity System es una biblioteca de Entity System que se administra bajo la organización LibGDX y es adecuada para el desarrollo de juegos. Depende de las clases de utilidad LibGDX, pero puede usarse con otros marcos de juegos Java que no estén basados en LibGDX con algo de trabajo.

Los sistemas de entidades proporcionan una forma diferente de administrar los datos y la funcionalidad hacia grandes conjuntos de objetos sin tener que enriquecer las clases de objetos con herencia.

Utilizar a Ashley podría ser un enfoque útil para aquellos que buscan un enfoque de modelado de objetos como el que proporciona Unity, pero con el alcance de un marco en lugar de un motor de juego.

Examples

Creando un componente

Los componentes son simplemente instancias que implementan la clase de componente Ashley.

```
import com.badlogic.ashley.core.Component;
import com.badlogic.ashley.core.ComponentMapper;

public class Position implements Component {
    public static final ComponentMapper<Position> Map =
                                ComponentMapper.getFor(Position.class);

    public float x = 0f,
                 y = 0f;
}
```

Los mapas de componentes proporcionan una forma rápida de acceder a los componentes de las entidades. Dos formas comunes de administrar sus mapas de componentes es mantener una instancia estática dentro de la clase de su componente o tener una clase / enumeración que contenga todos los mapeadores para todos sus componentes.

No es necesario declarar un asignador para un tipo de componente más de una vez en su aplicación.

Creación de un sistema de entidades

Los sistemas de entidades son la forma en que realiza operaciones funcionales en conjuntos de entidades. Normalmente, los componentes no deben tener una lógica asociada que involucre el conocimiento de los datos o el estado de otras instancias de componentes, ya que ese es el

trabajo de un sistema de entidades. Un sistema de entidad no se puede registrar en más de un motor a la vez.

Los sistemas de entidades no deben realizar más de un tipo de función. Un MovementSystem solo debería manejar el posicionamiento y similares, mientras que algo como un RenderSystem debería manejar el dibujo de entidades.

```
import com.badlogic.ashley.core.Entity;
import com.badlogic.ashley.core.EntitySystem;
import com.badlogic.ashley.core.Family;

public class MovementSystem extends EntitySystem {
    //the type of components necessary for entities to have to be operated on
    private static final Family FAMILY = Family.all(Position.class).get();

    public MovementSystem () {
        super();
    }

    /**
     * The update method called every tick.
     * @param deltaTime The time passed since last frame in seconds.
     */
    public void update (float deltaTime) {
        for (Entity e : this.getEngine().getEntitiesFor(FAMILY)) {
            Position pos = Position.Map.get(e);

            // do entity movement logic on component
            ...
        }
    }
```

A veces es útil extender sus EntitySystems con una funcionalidad adicional como la que se encuentra en los EntityListeners para que solo haga un seguimiento de los tipos de entidades en las que desea operar, en lugar de iterar sobre todas las entidades en el motor en cada ciclo. Los EntityListeners se activan cada vez que se agrega una entidad al mismo motor en el que está registrado el sistema.

```
import com.badlogic.ashley.core.EntityListener;
import com.badlogic.gdx.utils.Array;

public class MovementSystem extends EntitySystem implements EntityListener {
    Array<Entity> moveables = new Array<>();
    ...

    @Override
    public void entityAdded(Entity entity) {
        if (FAMILY.matches(entity)) {
            moveables.add(entity);
        }
    }

    @Override
    public void entityRemoved(Entity entity) {
        if (FAMILY.matches(entity)) {
            moveables.removeValue(entity, true);
        }
```

```
    }

    public void update (float deltaTime) {
        for (Entity e : this.moveables) {
            Position pos = Position.Map.get(e);

            // do entity movement logic on component
            ...
        }
    }
}
```

El seguimiento de un subconjunto de entidades que el sistema procesa en todo momento también puede ser óptimo, ya que puede eliminar una entidad particular del procesamiento de ese sistema sin tener que eliminar el componente asociado o eliminarlos del motor en su totalidad si así lo desea.

Creación de un sistema de entidades ordenadas

> Un `EntitySystem` simple que procesa cada entidad de una familia dada en el orden especificado por un `comparator` y llama a `processEntity()` para cada entidad cada vez que se actualiza el `EntitySystem` . Esto es realmente solo una clase de conveniencia ya que los sistemas de representación tienden a iterar sobre una lista de entidades de una manera ordenada. Agregar entidades hará que la lista de entidades se vuelva a ordenar. Llame a `forceSort()` si cambió sus criterios de clasificación. Para obtener más información, consulte Sistema de ordenación

En el ejemplo de código a continuación, el mejor uso para esto es la representación de sus sprites en un orden ordenado por zindex.

```
public class SpriteComponent implements Component {
     public TextureRegion region;
     public int z = 0;
}

public class Mapper {
     public static ComponentMapper<SpriteComponent> sprite =
ComponentMapper.getFor(SpriteComponent.class);
}

public class RenderingSystem extends SortedIteratingSystem {

    public RenderingSystem () {
         super(Familly.all(SpriteComponent.class).get(), new ZComparator())
    }

    public void processEntity(Entity entity, float deltaTime) {
         if(checkZIndexHasChangeValue()) {
              forceSort();
         }
    }

    private static class ZComparator implements Comparator<Entity> {
        @Override
        public int compare(Entity entityA, Entity entityB) {
```

```
            return (int)Math.signum(Mapper.sprite.get(entityA).z -
Mapper.sprite.get(entityB).z);
        }
    }

}
```

Creación de un sistema de iteración de intervalos

> Un `EntitySystem` simple que procesa una Familia de entidades no una vez por trama, sino después de un intervalo dado. La lógica de procesamiento de la entidad se debe colocar en `processEntity(Entity)` . Para obtener más información, consulte IntervalIteratingSystem

En el ejemplo de código a continuación, el mejor uso para esto es el paso del mundo de la física.

```
public class Constants {
     public final static float TIME_STEP = 1 / 60.0f; // 60 fps
     public final static int VELOCITY_ITERATIONS = 6;
     public final static int POSITION_ITERATIONS = 2;
}

public class PhysicsSystem extends IntervalIteratingSystem {
    public PhysicsSystem () {
         super(Family.all(PhysicsComponent.class), Constants.TIME_STEP);
    }

    @Override
    protected void processEntity(Entity entity) {
         // process the physics component here with an interval of 60fps
    }

    @Override
    protected void updateInterval() {
        WorldManager.world.step(Constants.TIME_STEP, Constants.VELOCITY_ITERATIONS,
Constants.POSITION_ITERATIONS);
        super.updateInterval();
    }

}
```

Lea Ashley Entity System en línea: https://riptutorial.com/es/libgdx/topic/2620/ashley-entity-system

Capítulo 4: Caja2d

Examples

Crear cuerpos Box2D desde un mapa en mosaico

Los objetos creados dentro de un Mapa en mosaico (.tmx), pueden cargarse simplemente como cuerpos en un mundo Box2D usando la clase MapObject de Libgdx de la siguiente manera:

```
public void buildBuildingsBodies(TiledMap tiledMap, World world, String layer){
    MapObjects objects = tiledMap.getLayers().get(layer).getObjects();
    for (MapObject object: objects) {
        Rectangle rectangle = ((RectangleMapObject)object).getRectangle();

        //create a dynamic within the world body (also can be KinematicBody or StaticBody
        BodyDef bodyDef = new BodyDef();
        bodyDef.type = BodyDef.BodyType.DynamicBody;
        Body body = world.createBody(bodyDef);

        //create a fixture for each body from the shape
        Fixture fixture = body.createFixture(getShapeFromRectangle(rectangle),density);
        fixture.setFriction(0.1F);

        //setting the position of the body's origin. In this case with zero rotation
        body.setTransform(getTransformedCenterForRectangle(rectangle),0);
    }
}
```

Las siguientes funciones ayudan a asignar las coordenadas del objeto en mosaico a la forma de Box2D.

```
public static final float TILE_SIZE = 16;
//Also you can get tile width with:
Float.valueOf(tiledMap.getProperties().get("tilewidth",Integer.class));

public static Shape getShapeFromRectangle(Rectangle rectangle){
    PolygonShape polygonShape = new PolygonShape();
    polygonShape.setAsBox(rectangle.width*0.5F/ TILE_SIZE,rectangle.height*0.5F/ TILE_SIZE);
    return polygonShape;
}
```

Y esta función ayuda a asignar el centro de un objeto en mosaico a la forma rectangular de Libgdx.

```
public static Vector2 getTransformedCenterForRectangle(Rectangle rectangle){
    Vector2 center = new Vector2();
    rectangle.getCenter(center);
    return center.scl(1/TILE_SIZE);
}
```

Por lo tanto, la primera función se puede utilizar de la siguiente manera:

```
public static final float GRAVITY = 9.8F;

public void createBodies(AssetManager assetManager){
    TiledMap tiledMap = assetManager.get("tiledMap.tmx");
    //create a Box2d world will contain the physical entities (bodies)
    World world = new World(new Vector2(0,GRAVITY),true);

    String layerName = "BuildingsLayers";
    buildBuildingsBodies(tiledMap,world,layerName);
}
```

Lea Caja2d en línea: https://riptutorial.com/es/libgdx/topic/5052/caja2d

Capítulo 5: Ciclo vital

Observaciones

Crear

Este método se llama una vez cuando se inicia la aplicación. En este método, los recursos deben cargarse y las variables deben inicializarse.

Hacer

El método se llama en cada fotograma y se usa para mostrar lo que sea necesario mostrar. También se utiliza para actualizar cualquier variable / clase que deba actualizarse, como una cámara.

Disponer

Este método se llama cuando la aplicación se destruye y se usa para liberar recursos, por ejemplo, `Texture` s o `SpriteBatch` . Sabrá que un objeto debe desecharse si implementa la interfaz `Disposable` .

Pausa

Este método se llama cuando la aplicación está en pausa. Por lo general, cuando la aplicación pierde el enfoque.

Currículum

Este método se llama cuando la aplicación debe reanudarse. Por lo general, cuando la aplicación recupera el enfoque.

Redimensionar

Este método se llama cuando se cambia el tamaño de la aplicación. Este método se utiliza normalmente para cambiar el tamaño de una ventana gráfica.

Examples

Archivo principal del juego

```
class Tutorial extends Game {

    public ScreenNumberOne screenNumberOne;

    public void create(){
        screenNumberOne = new ScreenNumberOne(this);

        this.setScreen(screenNumberOne);
    }

    public void render() {
        super.render();
    }
}
```

Ese es el archivo básico para permitirte hacer múltiples pantallas. Observe que se extiende el `Game` .

Lea Ciclo vital en línea: https://riptutorial.com/es/libgdx/topic/5053/ciclo-vital

Capítulo 6: Moviendo actores en camino con velocidad constante.

Examples

Movimiento simple entre dos ubicaciones.

Para esto la mejor solución es usar `actions` . Para agregar una nueva acción a un actor en `Scene2D` simplemente llame:

```
Action action = Actions.moveTo(x,y,duration);
actorObject.addAction(action);
```

Donde x e y es la ubicación objetivo y la duración es la velocidad de este movimiento en segundos (`float`).

Si desea detener esta acción (y el actor), puede hacerlo llamando a:

```
actorObject.removeAction(action);
```

o puedes eliminar todas las acciones llamando a:

```
actorObject.clearActions();
```

Esto detendrá inmediatamente la ejecución de la (s) acción (es).

La acción moveTo manipula las propiedades x e y del actor, de modo que cuando dibuje el actor en la pantalla, utilice siempre getX () y getY () para dibujar texturas. Al igual que en el siguiente ejemplo:

```
public class MovingActor extends Actor {

    private Action runningAction;
    private float speed = 2f;

    public void moveTo(Vector2 location) {
       runningAction = Actions.moveTo(location.x, location.y, speed);
       this.addAction(runningAction);
    }

    public void stopAction() {
       this.removeAction(runningAction);
    }

    public void stopAllActions() {
       this.clearActions();
    }

    @Override
```

```
    public void draw(Batch batch, float parentAlpha){
        batch.draw(someTexture, getX(), getY());
    }
}
```

Lea Moviendo actores en camino con velocidad constante. en línea: https://riptutorial.com/es/libgdx/topic/6384/moviendo-actores-en-camino-con-velocidad-constante-

Creditos

S. No	Capítulos	Contributors
1	Empezando con libgdx	Aryan, Community, Deniz Yılmaz, gaRos, Jesse Lawson, mttprvst13, Winter, Wyatt, Zoe
2	Apoyo a múltiples resoluciones	Eames, Lake, mttprvst13, Springrbua, Xoppa
3	Ashley Entity System	nhydock, ronscript
4	Caja2d	Alex Sifuentes, Alvaro Hernandez
5	Ciclo vital	Benedikt S. Vogler, mttprvst13, nhydock, Skrelp
6	Moviendo actores en camino con velocidad constante.	gaRos