	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					


Tema 5

JBox2D

Incompleto

Índice

1.	Introducción	1
2.	JBox2D en Android Studio	1
3.	Creando el mundo de JBox2D.....	1
4.	Crear el cuerpo de un objeto.....	2
4.1.	Definición del cuerpo	2
4.2.	Forma del cuerpo	3
4.2.1.	Círculos.....	3
4.2.2.	Polígonos.....	3
4.2.3.	Aristas	4
4.2.4.	Formas de cadena.....	5
4.3.	Propiedades	6
4.4.	Userdata	6
4.5.	Creando el cuerpo	6
4.6.	Simular el mundo en JBox2D	7
5.	Aplicar Fuerza e Impulso en el cuerpo.....	8
6.	JBox2D con Android y SurfaceView	9
7.	Bibliografía.....	10

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

1. Introducción

JBox2D es motor de física basado en Java para el desarrollo de juegos. Es una versión java de una librería de C++ muy popular llamada Box2d. Se puede utilizar en los juegos para mover objetos de forma natural. JBox2D funciona bien con objetos que tienen tamaño entre 0.1 y 10 metros.

Jbox2D no tiene interfaz de usuario.

2. JBox2D en Android Studio

Para poder usar JBox2d en Android Studio tendremos que añadir la dependencia:

```
implementation group: 'org.jbox2d', name: 'jbox2d-library', version: '2.2.1.1'
```

En el fichero build.gradle (en la parte del Module).

Por ejemplo:

```
implementation 'androidx.appcompat:appcompat:1.2.0'
implementation 'com.google.android.material:material:1.2.1'
implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
implementation group: 'org.jbox2d', name: 'jbox2d-library', version: '2.2.1.1'
```

3. Creando el mundo de JBox2D


El primer paso para crear usar JBox2D es crear un mundo. El mundo, world, es una colección de cuerpos (bodies), propiedades (fixtures) y restricciones (constraints). Los objetos y las simulaciones son administrados por el Mundo.

A continuación se muestra un fragmento de código para crear un mundo JBox2D:

```
Vec2 gravity = new Vec2(0.0f, -10.0f);
World world = new World(gravity);
boolean doSleep = true;
world.setSleepingAllowed(doSleep);
```

Podemos observar:

- Gravedad: la gravedad es una atracción de masa entre los cuerpos presentes en el Mundo. Como puedes ver en el fragmento anterior, el objeto Vec2 se utiliza para definir la gravedad. El primer parámetro define la gravedad horizontal y el segundo parámetro define la gravedad vertical. En la mayoría de los casos, el primer parámetro se establece en cero, ya que los requisitos para la gravedad horizontal son muy raros. Si se establece un valor negativo para el segundo parámetro, los cuerpos son atraídos hacia abajo mientras que si se establece un valor positivo los cuerpos son atraídos hacia arriba. En la mayoría de los casos se establece un valor negativo en el segundo parámetro
- doSleep: Si doSleep es verdadero, el mundo JBox2d permitirá a los cuerpos dormir cuando descansen. Los cuerpos durmiendo no son parte de las simulaciones; significa un mejor rendimiento ya que las simulaciones de los cuerpos son costosas. En la mayoría de los casos, este parámetro debe ser ajustado a true.

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

4. Crear el cuerpo de un objeto

Para crear cuerpos en el mundo de JBox2D primero tenemos que crear:

- Definición del cuerpo con una posición
- La forma del cuerpo y
- Propiedades para el cuerpo.

4.1. Definición del cuerpo

La definición de cuerpo se crea de la siguiente forma:


```
bodyDef = new BodyDef();
bodyDef.position.set(50, 50);
bodyDef.type = BodyType.DYNAMIC;
```

Sus principales propiedades son:

- Posición: posición del cuerpo en el mundo JBox2d, es decir, coordenadas x e y.
- Tipo de cuerpo: el tipo de cuerpo puede ser estático (static), dinámico (dynamic) o cinemático (kinematic).
- Angle: ángulo del cuerpo en radianes.
- linearVelocity: velocidad lineal del cuerpo.
- angularVelocity: la velocidad angular del cuerpo.
- Amortiguación lineal (linearDamping): establece la pérdida de velocidad durante el movimiento. Valores entre 0 y 1.
- Amortiguación angular (angularDamping): establece la pérdida de velocidad angular durante la rotación. Valores entre 0 y 1
- allowSleep: si el cuerpo puede dormir (y no será contado)
- bullet: para cuerpos que se mueven rápidamente y a los que se les debe impedir que hagan un túnel a través de otros cuerpos en movimiento
- fixedRotation: permite impedir que los cuerpos roten. Útil para personajes.
- awake: si el cuerpo comienza dormido o no.
- active: si el cuerpo comienza activo o no.
- userData: se utiliza para almacenar datos del cuerpo específicos de la aplicación

Cuerpo estático

- Tienen velocidad cero.
- No se mueven en las simulaciones, tienen velocidad 0.
- Pueden ser movidos manualmente por el usuario.
- Tienen una masa infinita.
- No colisionan con otros cuerpos estáticos y cinemáticos.
- Ejemplos: Tierra, Paredes, ...

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Cuerpo dinámico

- Se mueven bajo simulación a una velocidad propia.
- Responden a las fuerzas y se mueven en consecuencia.
- Dinámicos pueden moverse manualmente.
- Siempre tienen masa no nula. Si la masa se pone a cero, entonces adquiere 1Kg de masa.
- Chocan con los tipos de cuerpos dinámicos, cinemáticos y estáticos.
- Ejemplos: Bola, Caja, ...

Cuerpo cinemático

- Bajo simulación se mueven según su velocidad.
- No se mueven según las fuerzas.
- Pueden ser movidos manualmente por el usuario.
- Tienen una masa infinita.
- Solo colisionan con objetos dinámicos.
- Su movimiento no se ve afectado por la gravedad.
- Su movimiento no se ve afectado cuando los cuerpos dinámicos colisionan con él.
- Ejemplo: plataformas móviles en los juegos.

4.2. Forma del cuerpo

A continuación necesitamos definir la forma del cuerpo. JBox2D soporta la forma de círculo, la forma de polígono, aristas y la forma de cadena.

4.2.1. Círculos

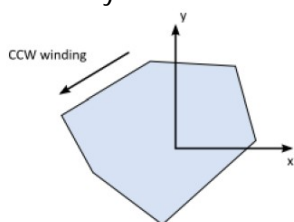
El tamaño de los círculos se define mediante su radio.

Los círculos son sólidos. No se puede hacer un círculo hueco. Sin embargo, puede simular mediante cadenas de segmentos de línea utilizando formas de polígono.

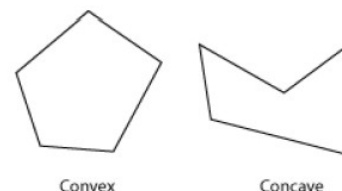
```
CircleShape cs = new CircleShape();
cs.m_radius = 2.5f;
```


4.2.2. Polígonos

Las formas poligonales son polígonos sólidos convexos. Un polígono es convexo cuando todos los segmentos de línea que conectan dos puntos del interior no cruzan ninguna arista del polígono. Los polígonos son sólidos y nunca huecos. Un polígono debe tener 3 o más vértices.



Hay que crear polígonos con un giro en sentido contrario a las agujas del reloj (CCW). Hay que tener cuidado porque la noción de CCW es con respecto a un sistema de coordenadas a la derecha con el eje z apuntando fuera del plano. Esto podría resultar ser en el sentido de las agujas del reloj en su pantalla, dependiendo de su sistema de coordenadas convenciones.



	RAMA:	Informática	CICLO:	Desenvolvemto de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Puedes crear una forma de polígono pasando una matriz de vértices. El tamaño máximo de la matriz tiene un valor por defecto de 8. Esto es suficiente para describir la mayoría de los polígonos convexos.

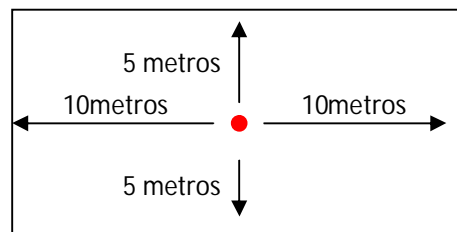
Definimos una forma de tipo polígono:

```
PolygonShape ps = new PolygonShape();
```

Y su forma mediante una caja:

```
ps.setAsBox(10,5);
```

SetAsBox toma como partida de creación el centro la coordenada especificada y crece el valor del primer parámetro tanto a la izquierda como a derecha del centro. Con el segundo parámetro pasa lo mismo crece su valor tanto por arriba como por abajo del centro. Estos tamaños se expresa en metros. El ejemplo anterior creará la siguiente caja:



También se pueden definir la forma de un polígono mediante sus vértices. En sentido antihorario (CCW). Al ser tres vértices definen un triángulo.

```
Vec2[] vertices=new Vec2[3];
vertices[0]= new Vec2(0.0f, 0.0f);
vertices[1]= new Vec2(1.0f, 0.0f);
vertices[2]= new Vec2(0.0f, 1.0f);
ps.set(vertices, vertices.length);
```

4.2.3. Aristas

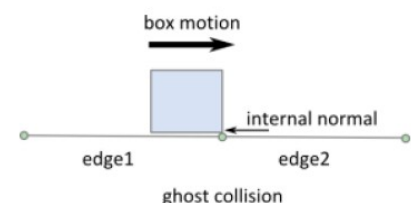
Las aristas son segmentos de línea. Se proporcionan para ayudar a hacer un entorno estático de forma libre. Una limitación importante de las formas de arista es que pueden colisionar con los círculos y polígonos pero no con ellas mismas.


Los algoritmos de colisión utilizados por JBox2D requieren que al menos una de las dos formas que colisionan tenga volumen. Las aristas no tienen volumen, por lo que la colisión arista-arista no es posible.

Esta es una forma de arista.

```
EdgeShape arista=new EdgeShape();
Vec2 v1 = new Vec2(2.0f, 10.0f);
Vec2 v2 = new Vec2(2.0f, 10.0f);
arista.set(v1, v2);
```

En muchos casos, un entorno de juego se construye conectando varias formas de borde de extremo a extremo. Esto puede dar lugar a una situación inesperada cuando un polígono se desliza a lo largo de la cadena de aristas. En la siguiente figura vemos



	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

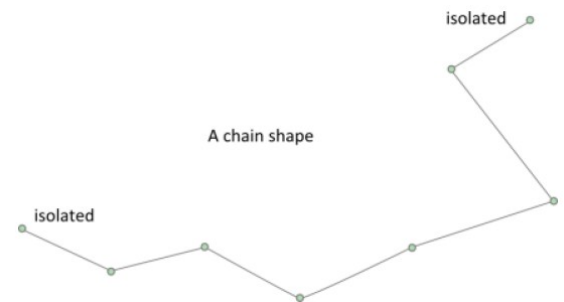
una caja colisionando con un vértice interno. Estas colisiones fantasma se producen cuando el polígono colisiona con un vértice interno generando una normal de colisión interna.

Si la arista1 no existiera esta colisión parecería estar bien. Con la arista1 presente, la colisión interna parece un error. Pero normalmente cuando Box2D colisiona dos formas, las ve de forma aislada.

En general, unir las aristas de esta manera es un poco tedioso. Esto nos lleva al siguiente tipo de forma.

4.2.4. Formas de cadena

La forma de cadena proporciona una manera eficiente de conectar muchas aristas juntas para construir sus mundos de juego estáticos. Las formas de cadena eliminan automáticamente las colisiones fantasma y proporcionan una colisión de dos lados.



Esta es una forma de cadena con vértices aislados

```
ChainShape formaCadena=new ChainShape();
Vec2[] vertices=new Vec2[4];
vertices[0]= new Vec2(1.7f, 0.0f);
vertices[1]= new Vec2(1.0f, 0.25f);
vertices[2]= new Vec2(0f, 0f);
vertices[3] = new Vec2(-1.7f, 0.4f);
formaCadena.createChain(vertices,vertices.length);
```

Es posible que tengas un mundo de juego con desplazamiento y quieras conectar varias cadenas entre sí. Se pueden conectar cadenas usando vértices fantasma:

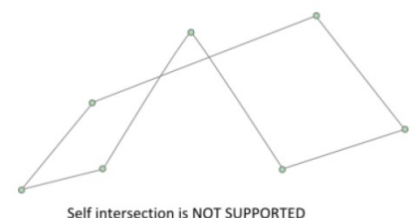
```
formaCadena.setNextVertex(new Vec2(3.0f, 1.0f));
formaCadena.setPrevVertex(new Vec2(-2.0f, 0.0f));
```

También puedes crear bucles automáticamente.

Por ejemplo podemos conectar el primer y último vértice.


```
formaCadena.createLoop(vertices,4);
```

Las formas de cadena con intersecciones no está soportada. Puede que funcione o no. El código que evita las colisiones fantasma asume que no hay auto-intersecciones de la cadena.



Cada borde de la cadena se trata como una forma hija y se puede acceder a ella por índice.

```
EdgeShape arista;
for (int i = 0; i < formaCadena.getChildCount() ; i++) {
    arista=new EdgeShape();
    formaCadena.getChildEdge(arista,i);
}
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

4.3. Propiedades

Mediante FixtureDef se asigna la forma a un cuerpo y añade propiedades materiales del mismo:

- Densidad (density): define la pesadez del cuerpo con respecto a su área. En kg/m^2
- Fricción (friction): define cómo los cuerpos se deslizan, perdiendo velocidad, cuando entran en contacto unos con otros. El valor de la fricción se puede establecer entre 0 y 1. Un valor más bajo significa que los cuerpos son más resbaladizos.
- Restitución (restitution): define la capacidad de rebote de un cuerpo (elasticidad). Valores entre 0 y 1. Aquí, un valor más alto significa un cuerpo más saltarín

Estas propiedades materiales describen cómo deben reaccionar dos cuerpos cuando chocan entre sí.

```
FixtureDef fd = new FixtureDef();
fd.shape = cs;
fd.density = 0.5f;
fd.friction = 0.3f;
fd.restitution = 0.7f;
```

4.4. Userdata

Tanto los cuerpos (Body), las definiciones de cuerpos (BodyDef), la propiedades (FixtureDef) pueden almacenar un objeto en la propiedad userData.

Estos son algunos ejemplos de casos en los se podrían:

- Aplicar daño a un actor usando un resultado de colisión.
- Acceder a una estructura del juego cuando Box2D le notifica que una union va a ser destruida.

Ten en cuenta que los datos del usuario son opcionales y se puede poner cualquier cosa en ellos.

4.5. Creando el cuerpo

Ahora el paso final es crear un cuerpo y añadirle un accesorio. Los cuerpos se crean usando la clase World.

```
body = world.createBody(bodyDef);
body.createFixture(fd);
```


Añadiendo un objeto FixtureDef al cuerpo, y en función de sus propiedades, actualiza automáticamente la masa del cuerpo. Se pueden añadir tantas FixtureDef como se quiera a un cuerpo. Cada uno de ellos contribuye a la masa total.

Una vez que un cuerpo no hace falta se puede destruir mediante el código:

```
world.destroyBody(body);
```

También se pueden eliminar sus propiedades mediante:

```
body.destroyFixture(fd);
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Todo el código visto hasta ahora seria:

```
// Definición del mundo
Vec2 gravity = new Vec2(0.0f, -10.0f);
World world = new World(gravity);
boolean doSleep = true;
world.setSleepingAllowed(doSleep);

// Definición del cuerpo
BodyDef bodyDef = new BodyDef();
bodyDef.position.set(250, 250);
bodyDef.type = BodyType.DYNAMIC;

// Definir la forma del cuerpo
CircleShape cs = new CircleShape();
cs.m_radius = 2.5f;

// Definir fijación del cuerpo
FixtureDef fd = new FixtureDef();
fd.shape = cs;
fd.density = 0.5f;
fd.friction = 0.3f;
fd.restitution = 0.7f;

// Crear el cuerpo y agregarle el fixture
Body body = world.createBody(bodyDef);
body.createFixture(fd);
```

4.6. Simular el mundo en JBox2D


JBox2D utiliza un algoritmo computacional llamado integrador. Los integradores simulan las ecuaciones físicas en puntos discretos del tiempo.

Además del integrador, JBox2D también utiliza un constraint solver. El cual resuelve todas las restricciones de la simulación, una por una. Una sola restricción puede ser resuelta perfectamente. Sin embargo, cuando resolvemos una restricción, alteramos ligeramente otras restricciones. Para obtener una buena solución, es necesario iterar sobre todas las restricciones varias veces.

Hay dos fases en el solucionador de restricciones: una fase de velocidad y una fase de posición.

- En la fase de velocidad el solucionador calcula los impulsos necesarios para que los cuerpos se muevan correctamente.
- En la fase de posición el solucionador ajusta las posiciones de los cuerpos para reducir el solapamiento y la separación de las uniones.

Cada fase tiene su propia cuenta de iteraciones.

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

Por esto para simular el mundo, necesitamos llamar al método de paso del mundo, el cual tiene tres parámetros:

- El tiempo entre cada ejecución. En el ejemplo anterior la simulación avanzará 1/60 de segundo en cada llamada al método del paso
- El número de iteraciones en la fase de velocidad. Define con qué precisión se simulará la velocidad. Un valor de iteración más alto aumenta la precisión de la simulación de la velocidad pero disminuye el rendimiento. El valor de iteración de velocidad recomendado es 6
- El número de iteraciones en la fase de posición. Es similar a la iteración de velocidad, un valor más alto significa una simulación de posición más precisa pero un menor rendimiento. El valor recomendado de iteración de posición es 3.

Por lo que el código quedaría:

```
float timeStep = 1.0f / 60.f; // 60 frames por segundo
int velocityIterations = 6;
int positionIterations = 2;

world.step(timeStep, velocityIterations, positionIterations);
Vec2 position = body.getPosition();
float angle = body.getAngle();
Log.i("aaa", position.x+" "+" "+angle+" "+Utils.world.getBodyCount());
```

En el fragmento de código anterior, en cada paso de la simulación obtenemos los nuevos detalles de posición y ángulo del cuerpo imprimiéndolos en la consola.

5. Aplicar Fuerza e Impulso en el cuerpo


Muchas veces habrá situaciones en las que en el mundo de JBox2D queramos mover los cuerpos manualmente. Entonces, ¿cómo lo hacemos? En el mundo real aplicamos la fuerza para mover cualquier cosa. Esto también se aplica en el mundo de JBox2D. Para mover los cuerpos en el mundo de JBox2D tenemos que aplicar la Fuerza o el Impulso.

Hay una diferencia entre fuerza e impulso.

- La fuerza actúa gradualmente sobre la velocidad de los cuerpos. Imagina que en el mundo real si queremos mover una gran caja de un lugar a otro, entonces empezaremos a empujarla. Significa que gradualmente aplicaremos fuerza sobre esa caja; como resultado la caja comenzará a moverse. La fuerza en el mundo de JBox2D también funciona de manera similar.
- E impulso actúa instantáneamente, en un paso de fracción de tiempo. El impulso es como si golpeáramos una bola de billar con un taco o moviéramos objetos con la ayuda de explosivos. Aplicar el impulso es similar a aplicar la fuerza. Necesitamos usar la función `applyLinearImpulse()` en lugar de `applyForce()`

A continuación se muestra un fragmento de código para aplicar fuerza sobre el cuerpo `body` que ya hemos definido previamente.

```
Vec2 force = new Vec2(0, 150.0f); // Fuerza que se aplica en los dos ejes
Vec2 point = body.getWorldPoint(body.getWorldCenter()); // Punto donde se aplica la fuerza, en
//este caso el centro del cuerpo
body.applyForce(force, point); // aplicamos la fuerza
```

	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

El siguiente fragmento de código de muestra es para aplicar el impulso.

```
Vec2 force = new Vec2(0, 50.0f); // Impulso que se aplica en los dos ejes
Vec2 point = body.getWorldPoint(body.getWorldCenter()); // Punto donde se aplica el impulso, en
// este caso el centro del cuerpo
body.applyLinearImpulse (force ,point); // aplicamos el impulso
```

6. JBox2D con Android y SurfaceView

En esta aplicación crearemos una pelota que, mediante el uso de JBox2D rebote en las paredes.

Nuestro primer paso es crear el proyecto Android, que en nuestro caso se llamara PruebaJbox2D con un surfaceView y añadirle la dependencia de JBox2D, al proyecto, como se vio en un punto anterior.

Añadiremos el método "addGround". Este método nos ayudará a añadir suelos y plataformas en nuestro mundo JBox2D. El suelo es un componente esencial de nuestra aplicación. Aquí es donde nuestra bola rebotará y finalmente se parara sobre ella. Si no añadimos el suelo, la bola se moverá infinitamente hacia abajo.

```
// Este método añade un suelo a la pantalla.
public Rect addGround(int x, int y, float ancho, float alto ){
    PolygonShape ps = new PolygonShape();
    ps.setAsBox(ancho,alto);

    FixtureDef fd = new FixtureDef();
    fd.shape = ps;

    BodyDef bd = new BodyDef();
    bd.position= new Vec2(x,y);
    world.createBody(bd).createFixture(fd);
    return new Rect(x,y,x+(int)ancho,y+(int)alto);
}
```

Como pueden ver, estamos usando la forma de polígono para crear el suelo. Cubriré las formas de JBox2D en detalle en un futuro post.


A continuación añadiremos el método "addWall". Este método nos permitirá añadir paredes a nuestro mundo JBox2D. Las paredes son necesarias porque no queremos que nuestra pelota saltarina salga del área visible de la aplicación.

```
// Este método crea una pared.
public Rect addWall(int x, int y, float ancho, float alto){
    PolygonShape ps = new PolygonShape();
    ps.setAsBox(ancho,alto);

    FixtureDef fd = new FixtureDef();
    fd.shape = ps;
    fd.density = 1.0f;
    fd.friction = 0.3f;

    BodyDef bd = new BodyDef();
    bd.position.set(x, y);

    world.createBody(bd).createFixture(fd);
    return new Rect(x,y,x+(int)ancho,y+(int)alto);
}
```

	RAMA:	Informática	CICLO:	Desenvolvimento de Aplicacions Multiplataforma		
	MÓDULO	Programación Multimedia y Dispositivos Móviles				CURSO: 2º
	PROTOCOLO:	Apuntes clases	AVAL:	1	DATA:	2021/2022
	UNIDAD COMPETENCIA					

7. Bibliografía

- JBox2d
 - <http://www.jbox2d.org/> Página web
 - <https://jar-download.com/artifacts/org.jbox2d/jbox2d-library/2.1.2.3/documentation> API
 - <http://thisiswhatiknowabout.blogspot.com/2011/12/jbox2d-tutorial.html>
 - <http://frisip.blogspot.com/2011/12/android-tips-jbox2d-tutorial.html>
 - <https://github.com/jbox2d/jbox2d>: Página en github
 - <https://github.com/jbox2d/jbox2d/wiki> Wiki
 - <https://github.com/jbox2d/jbox2d/wiki/Testbed> Wiki de los ejemplos de código
 - <https://github.com/jbox2d/jbox2d/tree/master/jbox2d-testbed> Ejemplos de código
 - <https://code.google.com/archive/p/jbox2d/downloads> Página antigua de descargas
 - <https://mvnrepository.com/artifact/org.jbox2d/jbox2d-library/2.2.1.1> Repositorios

- Box2d: Proyecto original del que JBox2d es un *fork*.
 - <https://box2d.org/documentation/> Documentación online.
 - https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/box2d/Box2D_v2.2.1.zip Biblioteca Box2d donde dentro del directorio *Documentation* está la API y el manual de utilización.

- Física: capa que simplifica el uso de JBox2d
 - <http://ricardmarxer.com/fisica/>
 - http://ricardmarxer.com/fisica/2012/nov_zzzinc/#1
 - <http://mycours.es/fisica/>