









Taller de Desarrollo de videojuegos multiplataforma con lib**GDX**

Flappy Plane

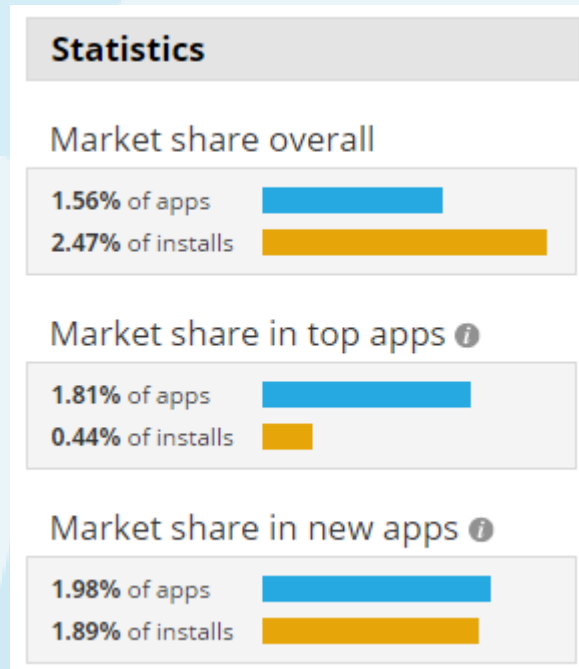


Alberto Cejas Sánchez
David Saltares Márquez

¿Por qué lib**GDX**?

-  Multiplataforma
-  Comunidad
-  Open source
-  Rapidez
-  Buena y amplia documentación
-  Múltiples niveles de abstracción
-  Desarrollo activo
-  ¡¡Totalmente Gratis!!

Sí, sí, pero... ¿de verdad lib**GDX**?



Total apps en Google Play:

1.400.719

Total lib**GDX** apps en Google Play:

21.851

Top categorías en Google Play:

1. Education
2. Entertainment
3. Lifestyle

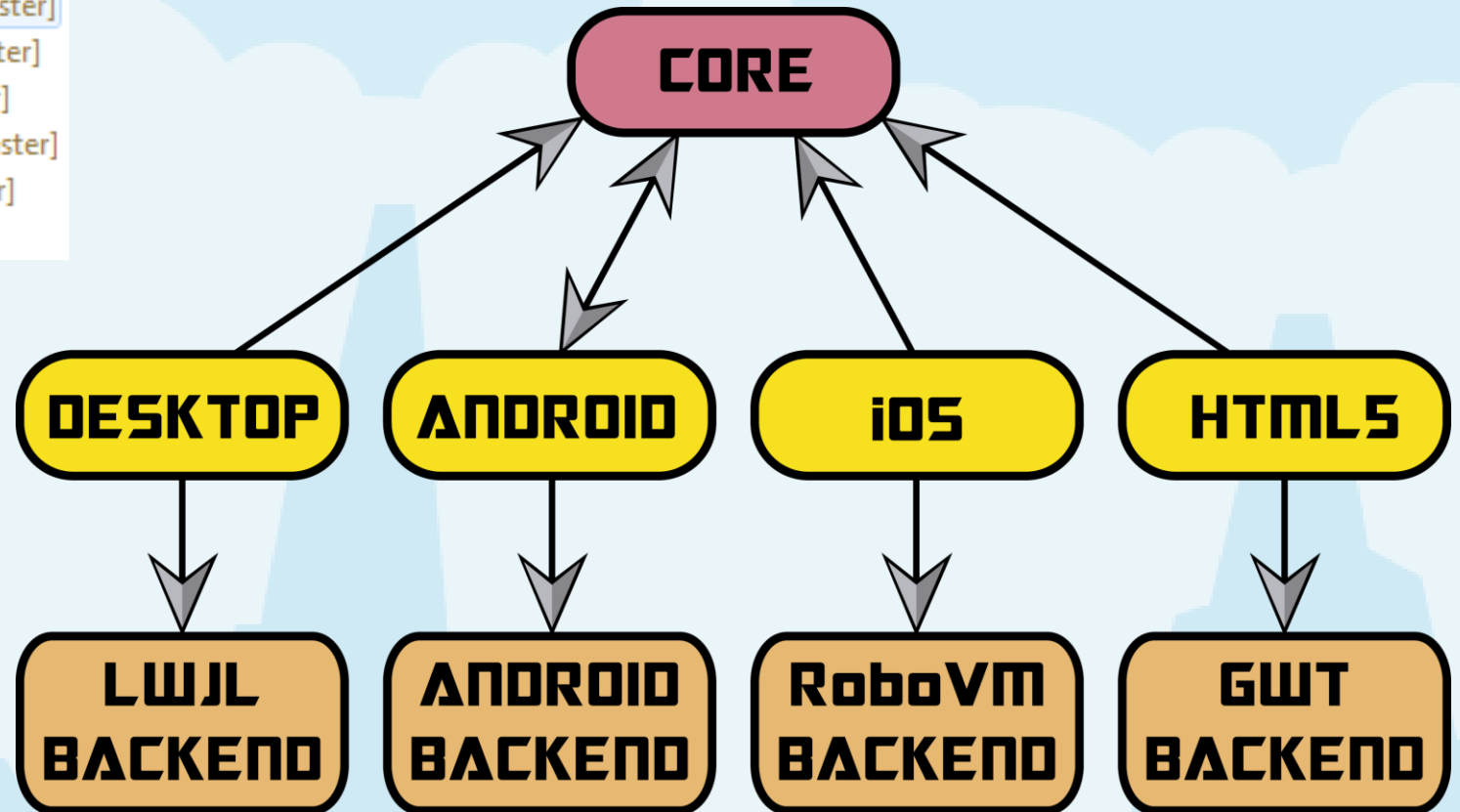


¿Qué voy a hacer?

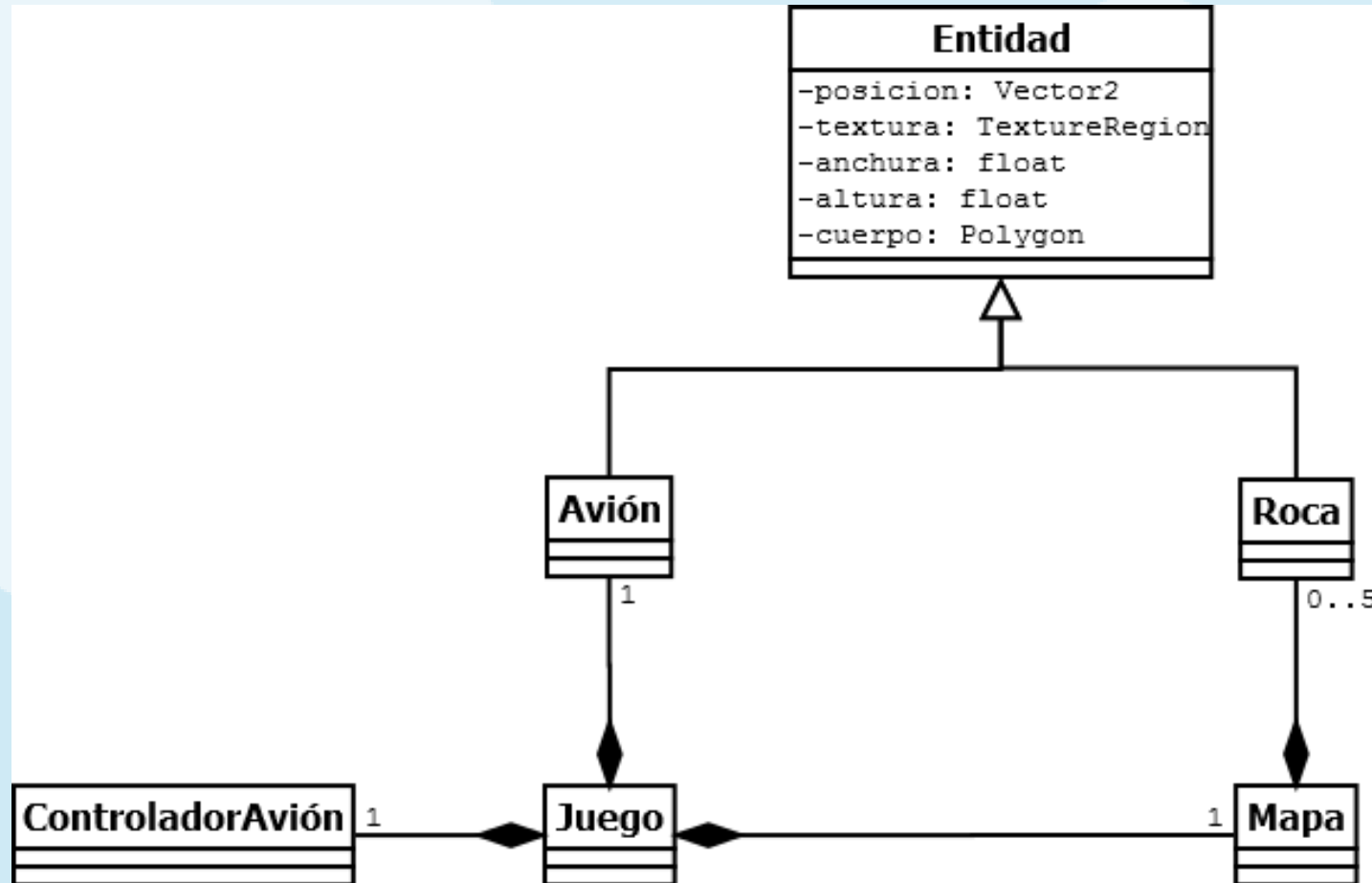


Estructura

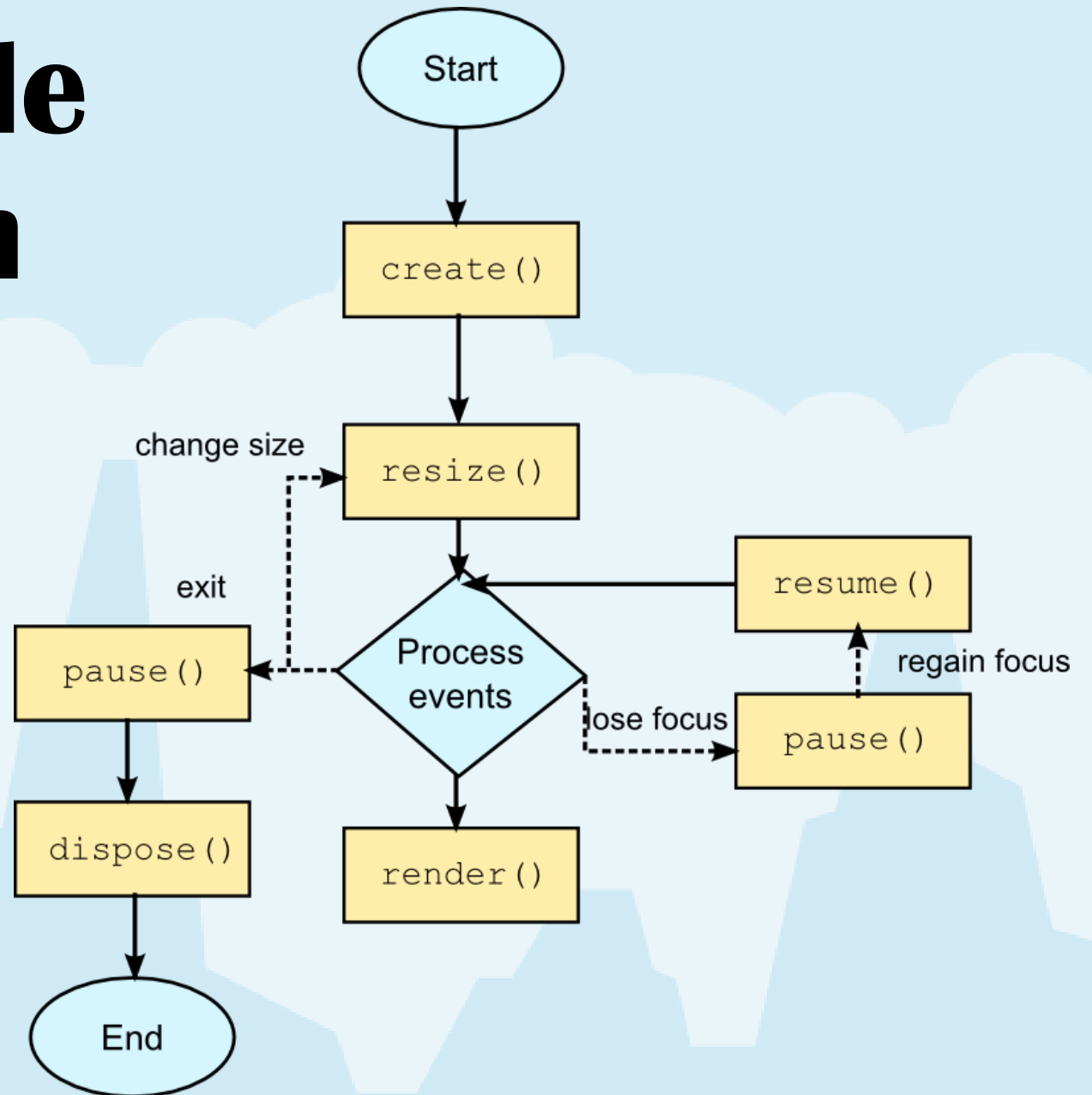
```
> ThePlane-android [flappybird_workshop master]
> theplane-basic-1 [flappybird_workshop master]
> ThePlane-core [flappybird_workshop master]
> ThePlane-desktop [flappybird_workshop master]
> ThePlane-html [flappybird_workshop master]
> ThePlane-ios [flappybird_workshop master]
```



Arquitectura

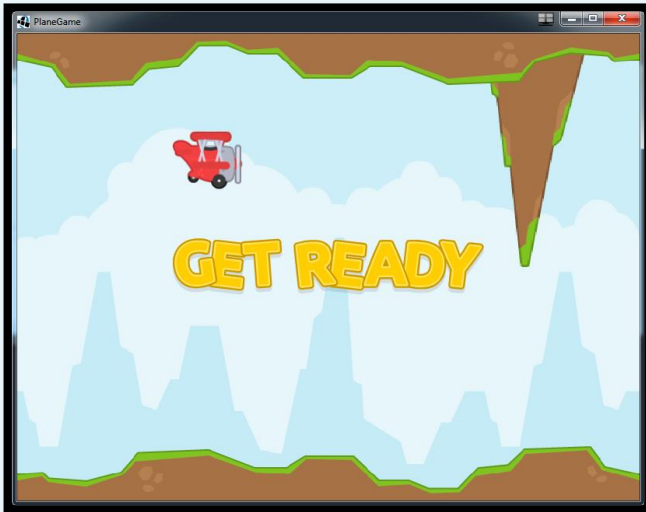


Ciclo de vida de una aplicación Libgdx

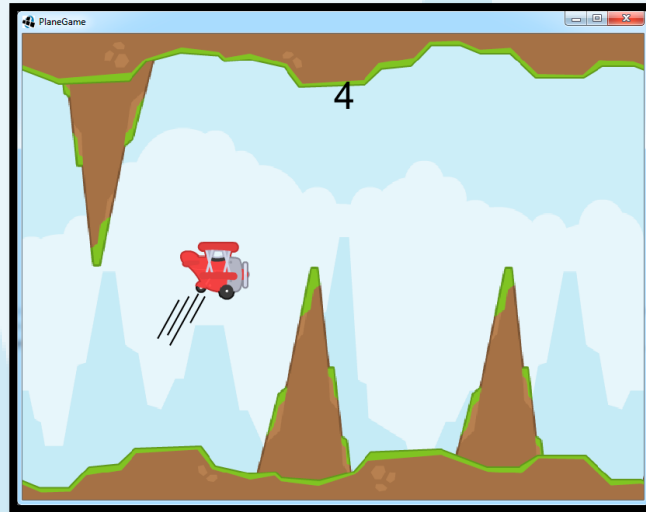


Estados del juego

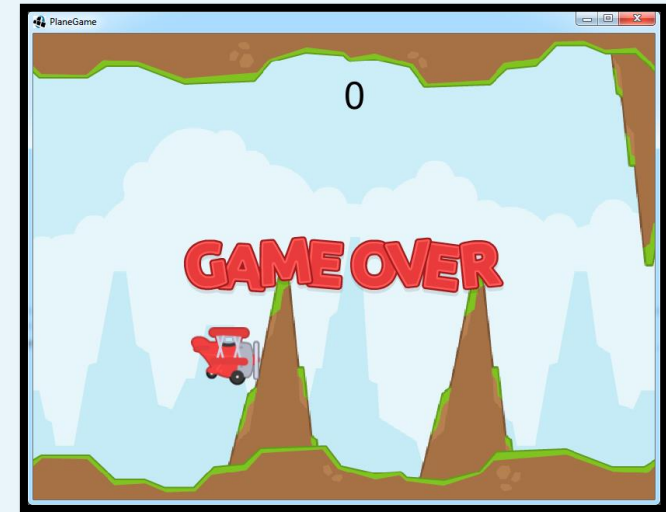
```
static enum GameState {  
    Start, Running, GameOver  
};
```



Start



Running



GameOver

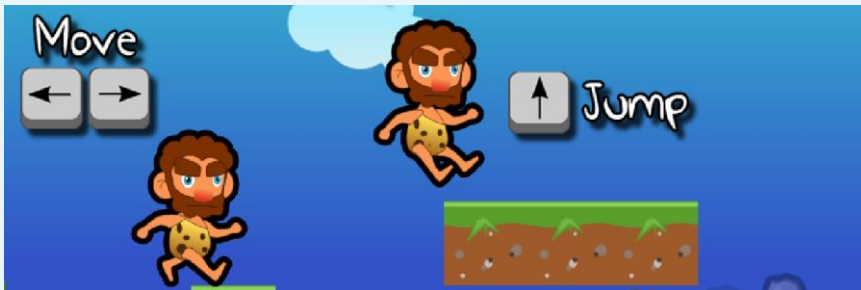
Viewports



StretchViewport



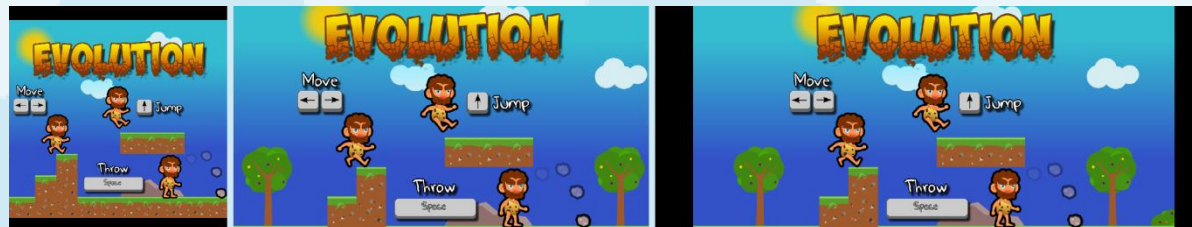
FitViewport



FillViewport



ScreenViewport



ExtendViewport

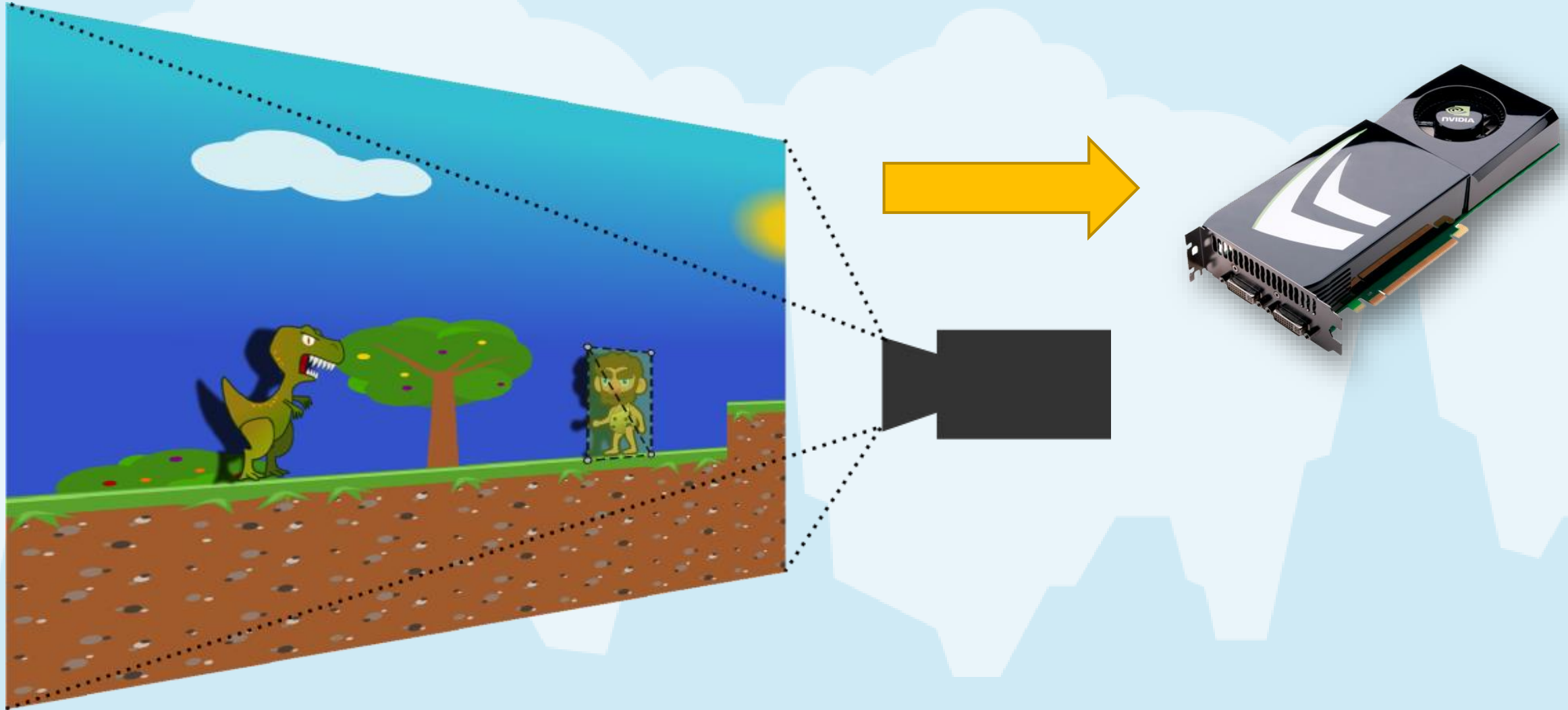
Ok, pero... ¿Cuál uso hoy?

```
private Viewport viewport;  
private Viewport uiViewport;  
private OrthographicCamera camera;  
private OrthographicCamera uiCamera;
```

```
camera = new OrthographicCamera();  
uiCamera = new OrthographicCamera();  
viewport = new FitViewport(SCENE_WIDTH, SCENE_HEIGHT, camera);  
uiViewport = new FitViewport(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(), uiCamera);  
  
// Center camera  
viewport.update(SCENE_WIDTH, SCENE_HEIGHT, true);  
uiViewport.update(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(), true);
```

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

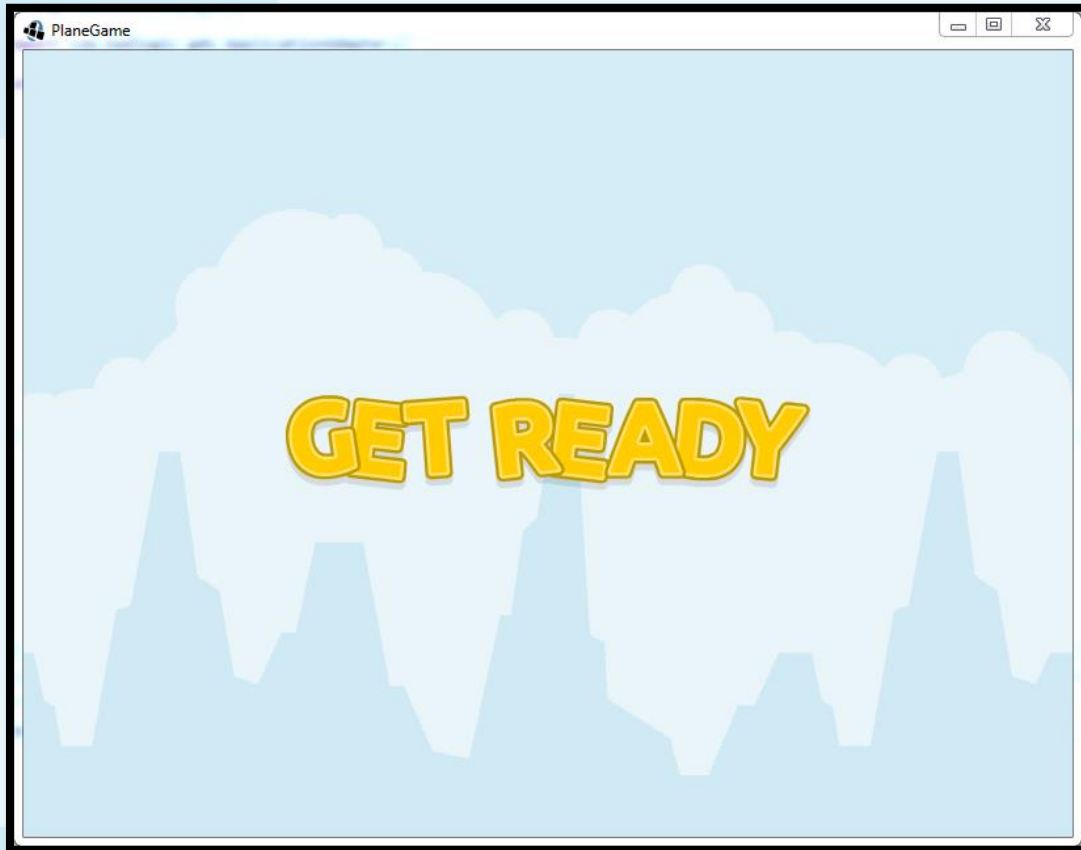
SpriteBatch



Paso 1






Resultado:





Implementar:

Juego

-  GameState: *Start*
-  Dibujar mundo (*Mapa*)
-  Dibujar User Interface (*Get ready*)

Mapa




-  Cargar *background.png*
-  Dibujar *background.png*

Paso 1



Pistas:

- Cargar `ready.png` como `Texture` en `create()`
- Instanciar `Mapa` en `create()`
- Pasar a `GameState.Running` cuando el usuario pulse en la pantalla (`Gdx.input.justTouched()`) dentro de `updateStart()`
- Cargar `background.png` como `Texture` en `create()` de `Mapa`
- Implementar método `draw` en `Mapa` con `batch.draw(Texture texture, float x, float y, float width, float height)` y usarlo dentro de `drawWorld()` en `PlaneGame`
- Dibujar `ready` en el **centro** de la pantalla cuando el juego esté en `GameState.Start` dentro de `drawUI()`. Haz uso de `batch.draw(Texture texture, float x, float y)`. Necesitarás `getWorldWidth()` y `getWorldHeight()` de la clase `Viewport`, además de `getWidth()` y `getHeight()` de la clase `Texture`.
- Liberar recursos en `dispose()`

Juego

-  `GameState: Start`
-  Dibujar mundo (`Mapa`)
-  Dibujar User Interface (`Get ready`)

Mapa

-  Cargar `background.png`
-  Dibujar `background.png`

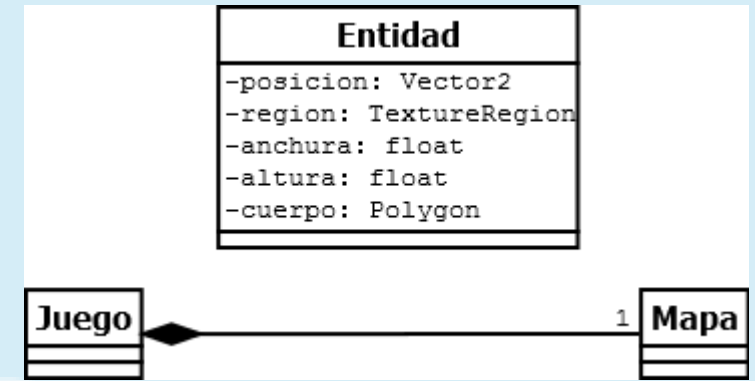
Paso 2


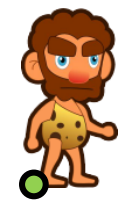

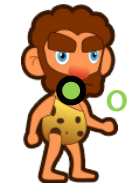

Implementar:

Entidad

Pistas:

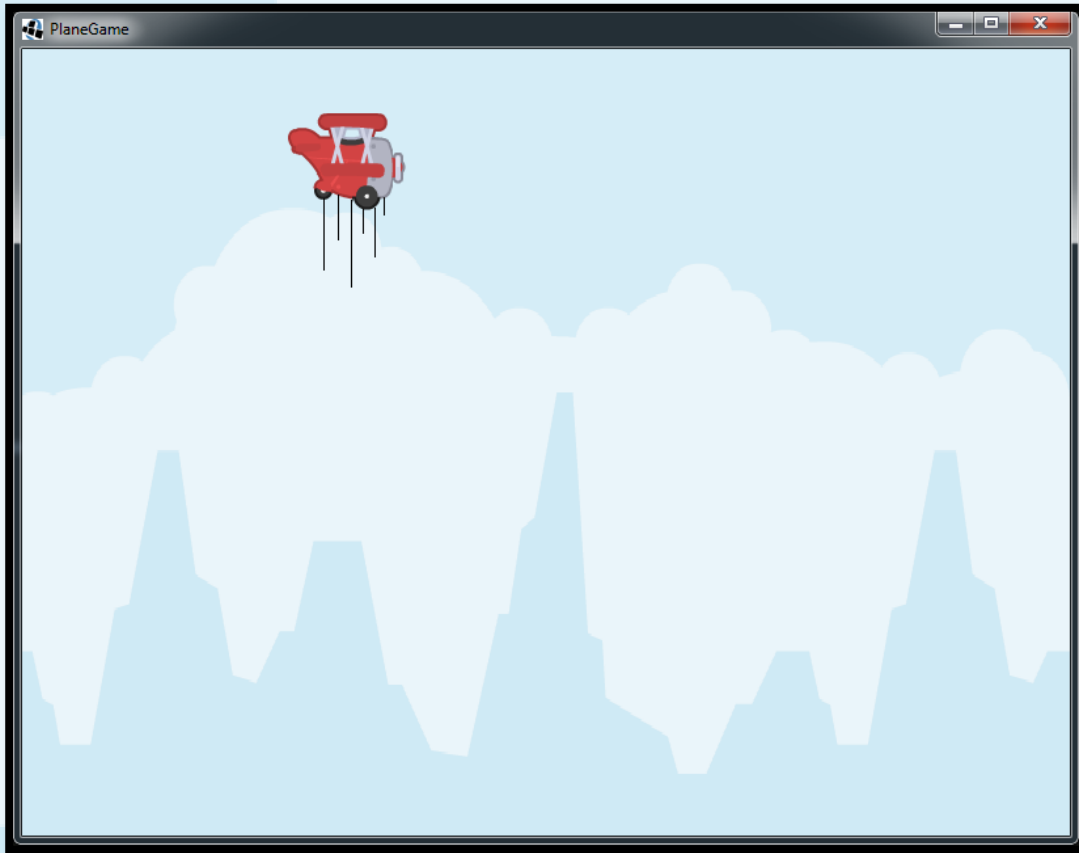
- El **origen** de *cuerpo* debe ser su **centro**: *cuerpo.setOrigin(x,y)* dentro del constructor
- *setPosition*, *setRotation* y *translate* deben afectar también al *cuerpo*
- Usa *draw(TextureRegion region, float x, float y, float width, float height)* para implementar *entitiy.draw(SpriteBatch batch)*
- Usa *boolean Intersector.overlapConvexPolygons(Polygon p1, Polygon p2)* para comprobar si el *cuerpo* de dos entidades se **solapan/colisionan**.



Original	Rotada 180° 
 Origen	
 Origen	

Paso 3

Resultado:



Implementar:

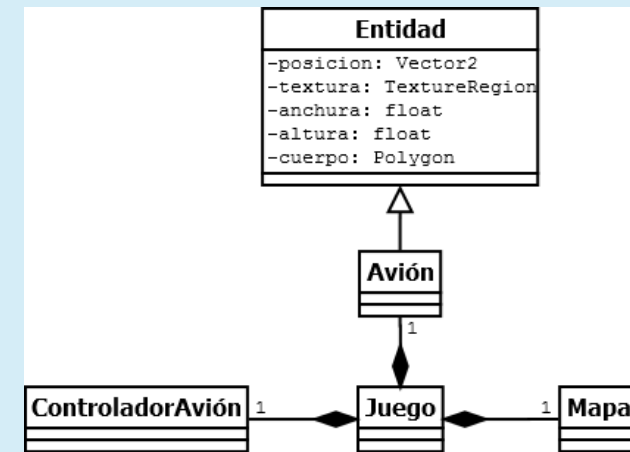
✈️ **Avión**

✈️ **Juego**

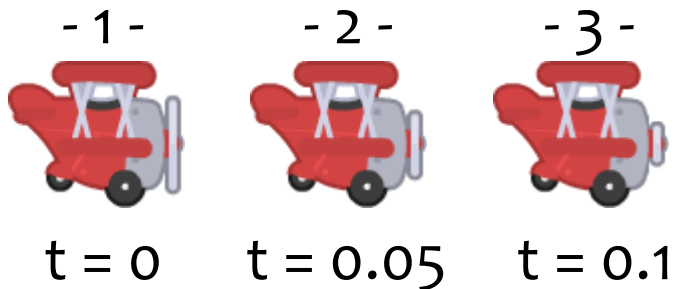
✈️ GameState: *Running*

✈️ Añadir avión y actualizarlo

✈️ Dibujar avión



Paso 3 – introducción



Animation

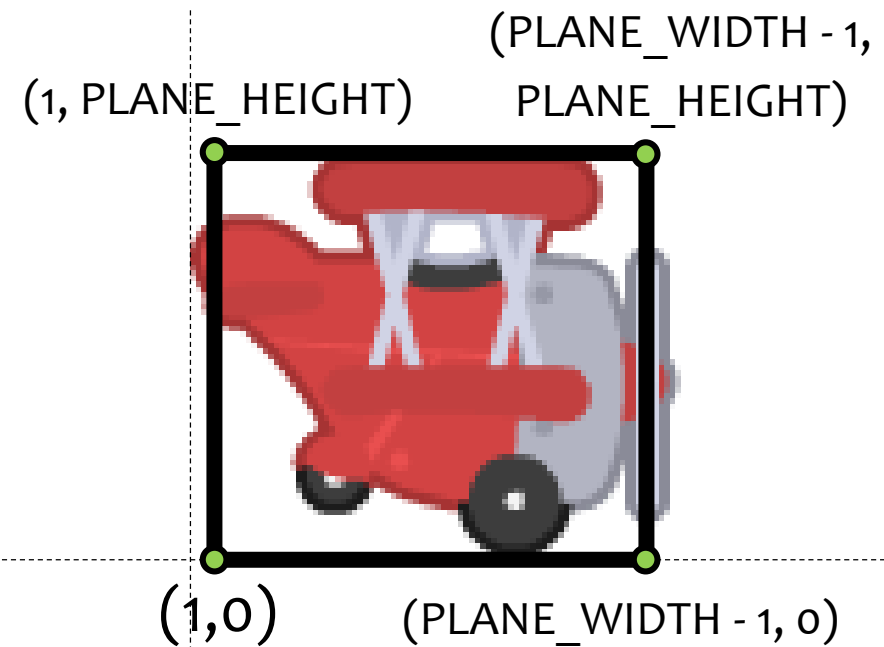
statetime = 0.03



statetime = 0.08



statetime = 0.24




static Polygon createShape()


Paso 3 – parte 1


Pistas:


- Cargar `frame1.png` en la variable `frame1` de tipo `Texture` en el constructor de `Avion`. Repetir con `frame2` y `frame3`.
- En el mismo constructor, inicializar `animation` a través de `Animation(float frameDuration, TextureRegion... keyFrames)`, donde `frameDuration` será `0.05` segundos y los `keyframes` serán `frame1`, `frame2` y `frame3`. Hacer que se repita la animación llamando a: `animation.setPlayMode(PlayMode.LOOP);`
- Dentro de `reset()`, poner la velocidad inicial de `Avion` a `(0,0)` y colocarlo en la posición inicial
- Dentro de `update(float delta)`, aumentar `stateTime` con el `delta` transcurrido. Añadir el efecto de la gravedad al vector `velocity` con el método `add`, y escalarlo al tiempo `delta` transcurrido con el método `scl`. Por último mover el `Avion` con `translate` según indica `velocity`.
- En `draw()`, establecer la región a dibujar según el `stateTime` actual, haciendo uso de `setRegion` de su clase base `Entidad`, y de `animation.getKeyFrame`. Para dibujarlo, llamar a `draw` de `Entidad` con `super.draw(batch)`
- Liberar recursos en `dispose()`

 **Avión**

 **Juego**

 GameState: *Running*

 Añadir avión y actualizarlo


 Dibujar avión


Paso 3 – parte 2


Pistas:


- En `ControladorAvion`, `touchdown` se ejecutará cada vez que el usuario pulse en la pantalla. Por lo que debemos actualizar la velocidad de `avion`, manteniendo su velocidad en `X` pero impulsando su velocidad en `Y` según `PLANE_JUMP_IMPULSE`
- Instanciar `Avion` y `ControladorAvion` en `create()` de `Juego`
- Implementar `updateCamera()` de manera que sitúe `camera` en la misma posición `X` que el `avion`, sumándole `20`. Luego, llama `camera.update()`
- En `updateStart()`, establece `ControladorAvion` como procesador de entrada e arranca `avion` con su velocidad inicial.
- En `updateRunning()`, actualiza `avion` con el `delta` transcurrido
- Dibuja `avion` dentro de `drawWorld()`
- Liberar recursos en `dispose()`

 **Avión**

 **Juego**

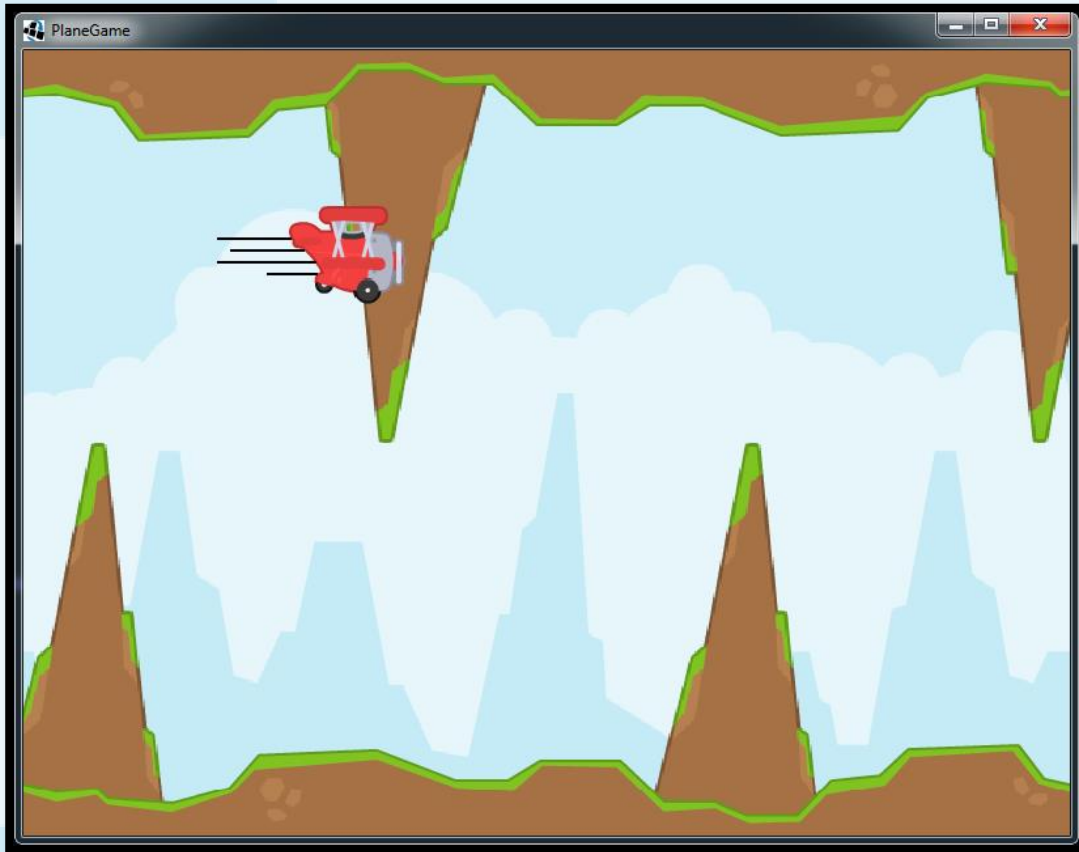
 GameState: *Running*

 Añadir avión y actualizarlo

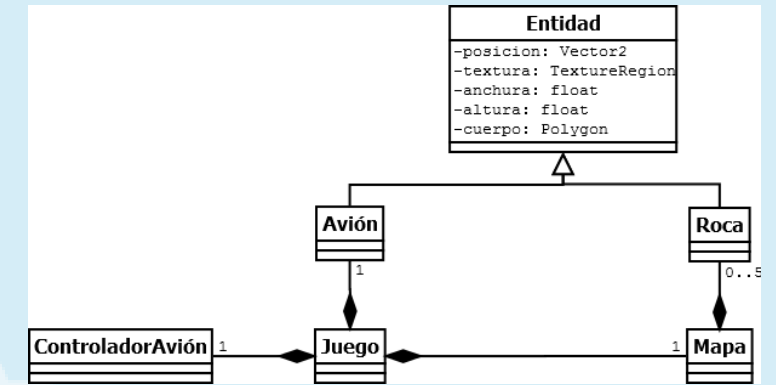
 Dibujar avión

Paso 4

Resultado:



Implementar:



✈️ **Roca**

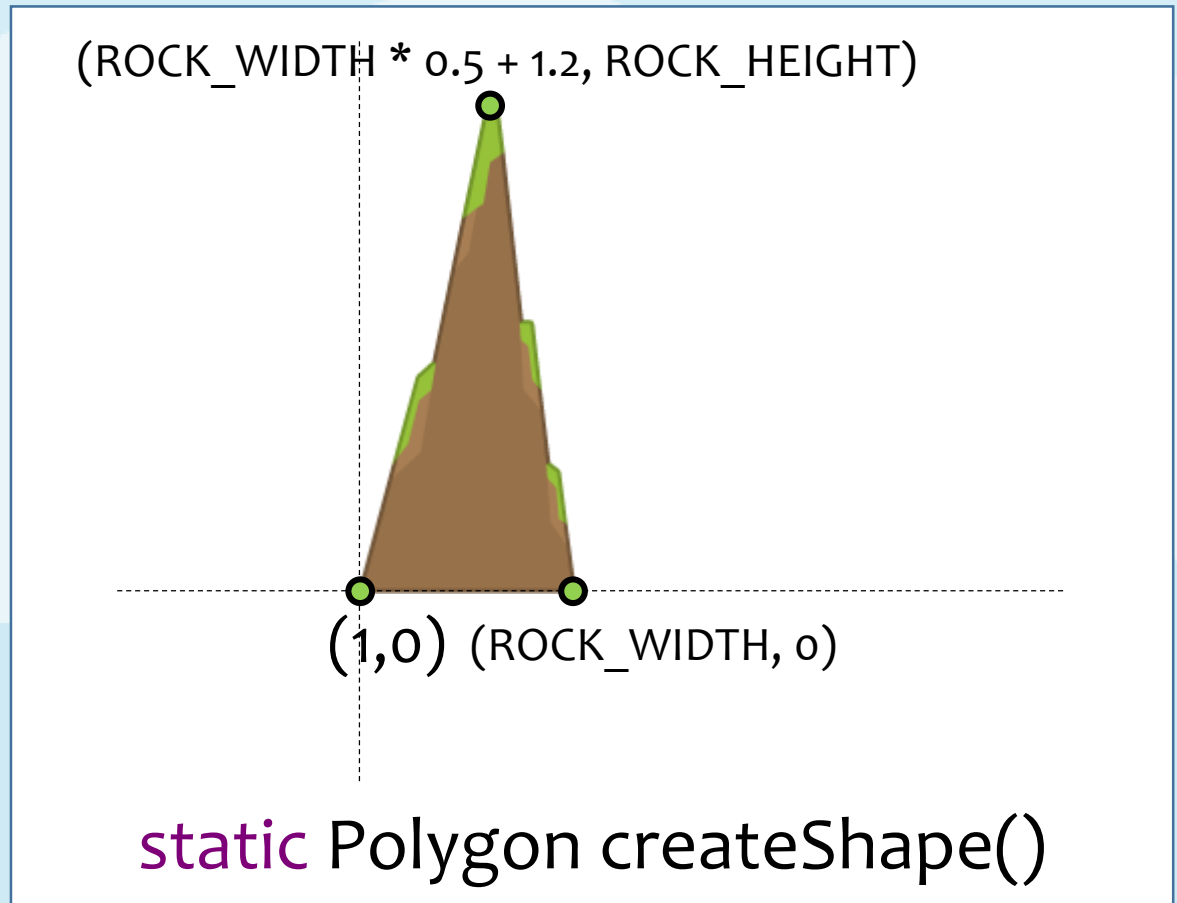
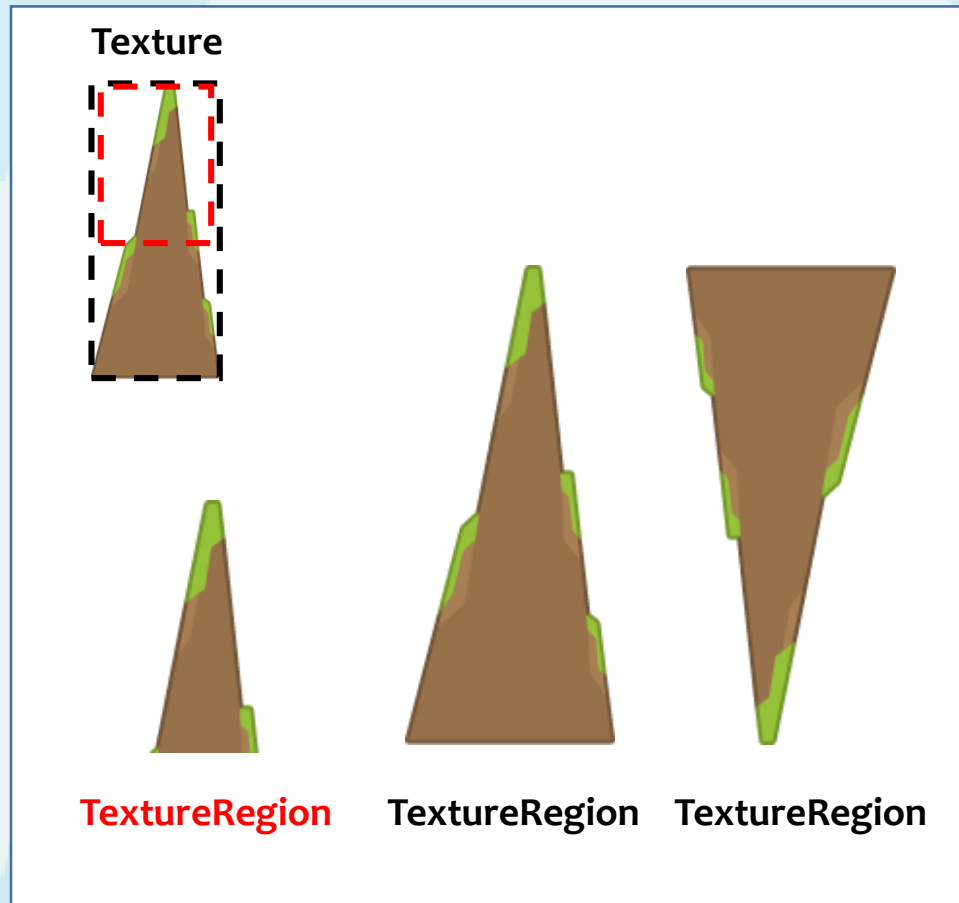
✈️ **Juego**

✈️ GameState: *Running*

✈️ **Mapa**

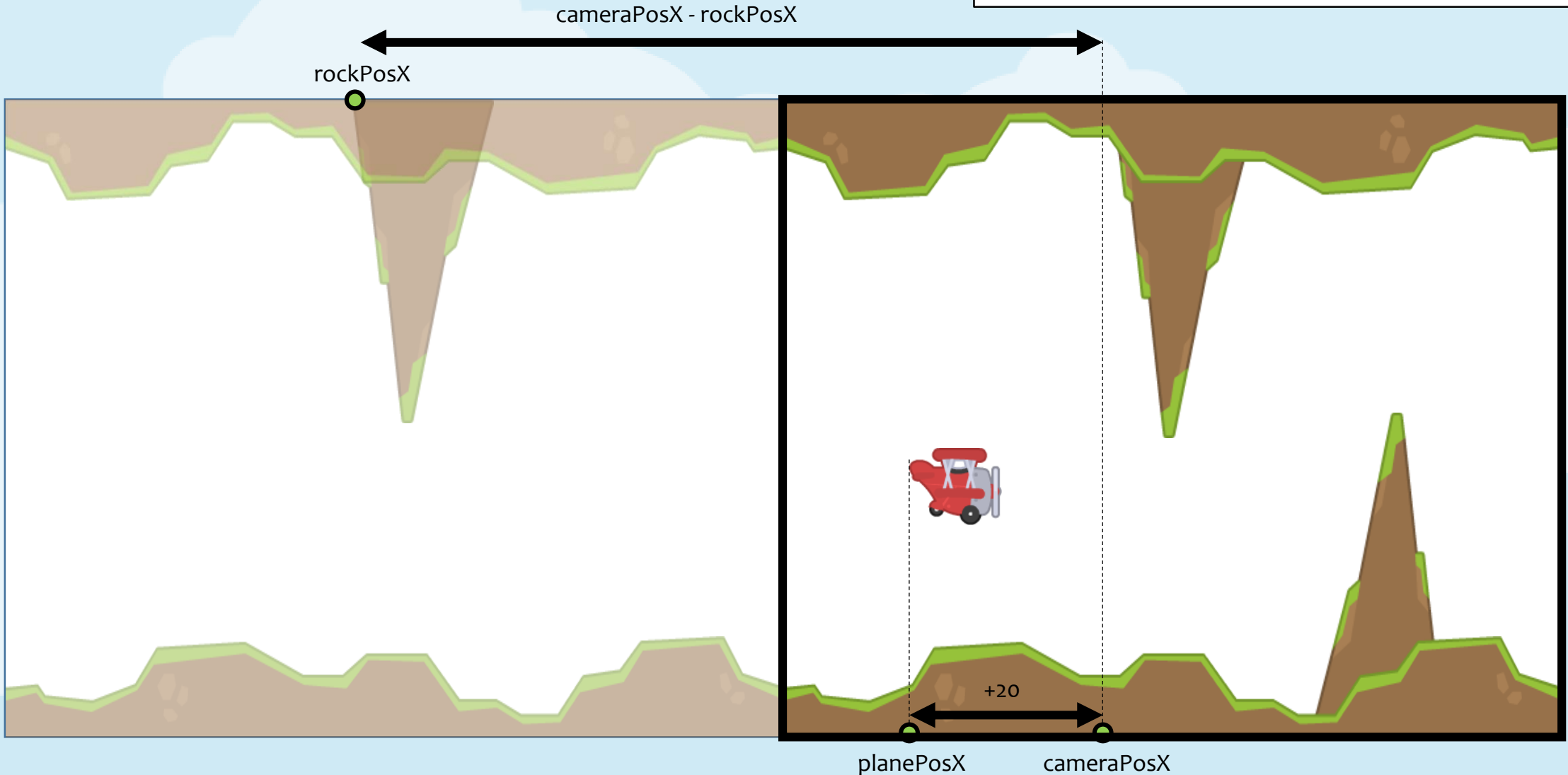
✈️ Rocas, suelo, techo

Paso 4 – introducción



Paso 4 – introducción

```
if (cameraPosX - rockPosX > SCENE_WIDTH*0.5 + ROCK_WIDTH) {  
    // TRASLADA LA ROCA HACIA DELANTE  
}
```





Paso 4 – parte 1

Pistas:


- Implementar **Roca**, hacer uso de la posición (x,y) recibida en su constructor. Además, implementar **createShape()** según la transparencia de introducción.
- Ampliar **Mapa** con **Texture** y **TextureRegion** de la **Roca** (**rock.png**) y el suelo (**ground.png**). Ten en cuenta que para cada una de ellas debe haber una region volteada con **flip(true, true)**
- Implementar **reset()**, vaciando el **Array** de rocas con **clear()** y creando 5 nuevas. Para ello, se generará un booleano aleatorio haciendo uso de **MathUtils.randomBoolean()**. La posición **X** de la roca será $60 + i * 25$, donde **i** es el número de roca. La posición **Y** dependerá del booleano anterior, si es **true**, será **30**, sino **0**. Además el booleano también marcará la región y la rotación, de manera que si es **true**, se utilizará la region volteada y una rotación de **180°**. Si es **false**, la region por defecto y una rotación de **0°**. No olvides añadir las rocas al **ArrayList**.

 **Roca**

 **Juego**

 GameState: *Running*

 **Mapa**


 Rocas, suelo, techo

Paso 4 – parte 2

Pistas:


- Implementar `update()` de `Mapa`, de manera que las rocas que ya no se vean según la condición de la transparencia de introducción, aumenten su posición `X` en $5 * 25$. Ten en cuenta todo lo referente al booleano del apartado anterior ya que también se aplica aquí. Además, no olvides reiniciar la rotación a 0 antes de aplicarle una nueva.
- Añade un parámetro llamado `offsetX` a `draw`. En la misma función, dibuja las rocas con el método base `draw(SpriteBatch batch)`. Luego, dibuja dos veces el `suelo` y el `techo`. La primera en la posición `X` que marque `offsetX`. La segunda en `offsetX + GROUND_WIDTH`
- En `updateRunning()` de `Juego`, cada vez que se recorra `SCENE_WIDTH` 1.5 veces, aumentar distancia recorrida por `avion` en el acumulador `groundOffsetX` con `SCENE_WIDTH`. Luego llamar a `update` de `Mapa` con la posición `X` de `camera`.
- Liberar recursos en `dispose()`

 **Roca**

 **Juego**

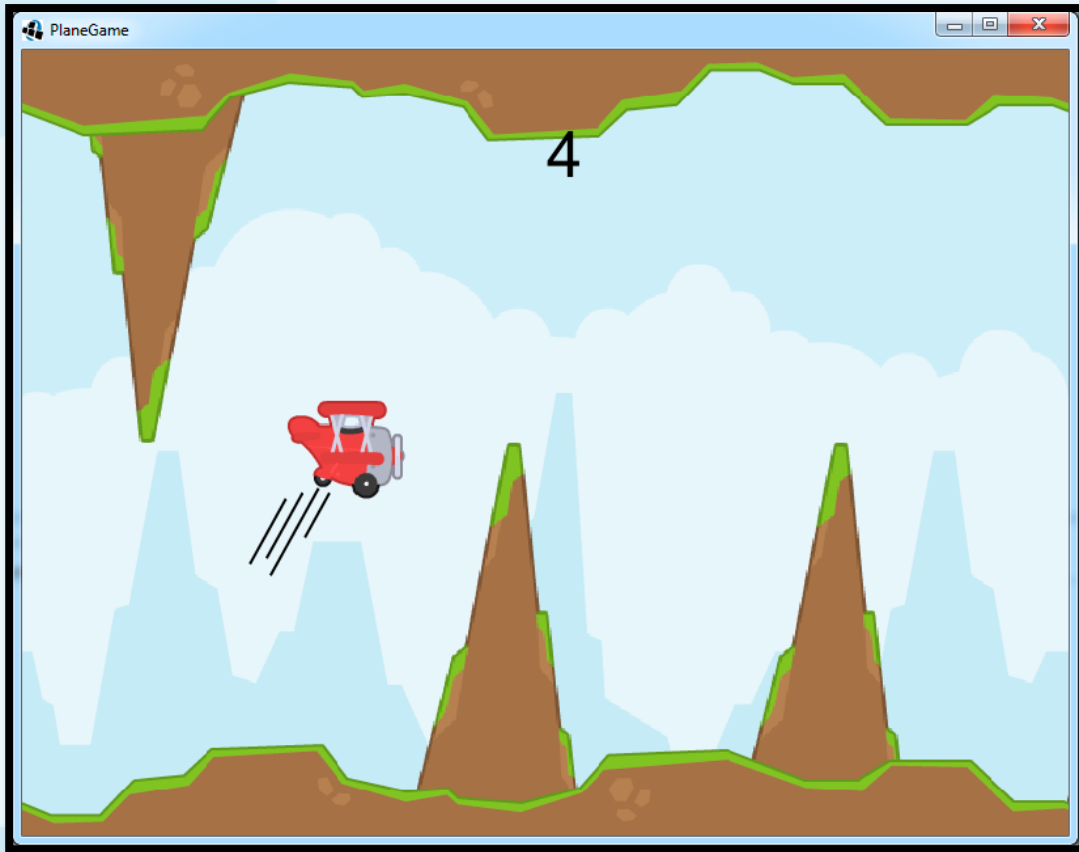
 `GameState: Running`

 **Mapa**

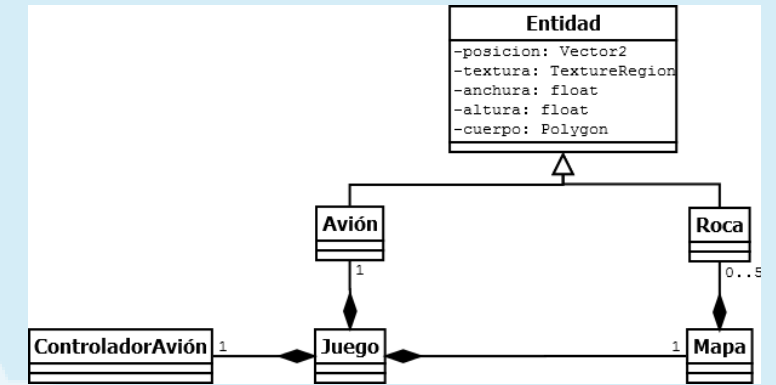
 `Rocas, suelo, techo`

Paso 5






Resultado:



Implementar:



Juego

-  GameState: *Running*
-  Colisiones
-  Marcador
-  GameState: *GameOver*
-  Musica y efectos


Paso 5 – parte 1

Pistas:

- Crea una variable `marcador` con valor `0`
- Instanciar `music.mp3` como `Music` con `Gdx.audio.newMusic(Gdx.files.internal("music.mp3"))`. Haz que repita en bucle con `setLooping(true)` y llama a `play()`.
- Repite el paso anterior pero con `explode.wav`. Ten en cuenta que en lugar de ser de tipo `Music` es de tipo `Sound`.
- En update de `Mapa`, pon `counted` de cada `Roca` que muevas a `false`.
- Implementa `checkCollisions()` de manera que compruebe si cada una de las rocas `collide` con `avion` y quitándole el control al `ControladorAvion` con `setInputProcessor(null)`, estableciendo la velocidad de `avion` a `0` y pasando al modo `GameState.GameOver` si no lo está ya (y con ello reproducir `explode`). Por último, puedes aprovechar el bucle para aumentar `marcador` si `avion` ha pasado la posición de la `Roca` actual. ¡Cuida de no contar dos veces la misma `Roca`!. Repite lo anterior con `techo` y `suelo`, pero en lugar de usar `collide`, el limite superior de altura será `SCENE_HEIGHT - GROUND_HEIGHT + 2` y el límite inferior `GROUND_HEIGHT - 2`.
- Liberar recursos en `dispose()`


Juego

 GameState: *Running*

 Colisiones

 Marcador

 GameState: *GameOver*

 Musica y efectos


Paso 5 – parte 2

Pistas:

- Instanciar `arial.fnt` como `BitmapFont` llamada `font` usando `Gdx.files.internal`. Pon su color `Color.BLACK` con `setColor`
- Cargar `gameover.png` como `Texture` en `create()`
- Dibujar `gameover` cuando estemos en `GameState.GameOver` de la misma forma que con `ready`.
- Dibujar el marcador en la posición `(SCENE_WIDTH * 0.5, SCENE_HEIGHT - 60)`.
- Liberar recursos en `dispose()`


Juego

 GameState: *Running*

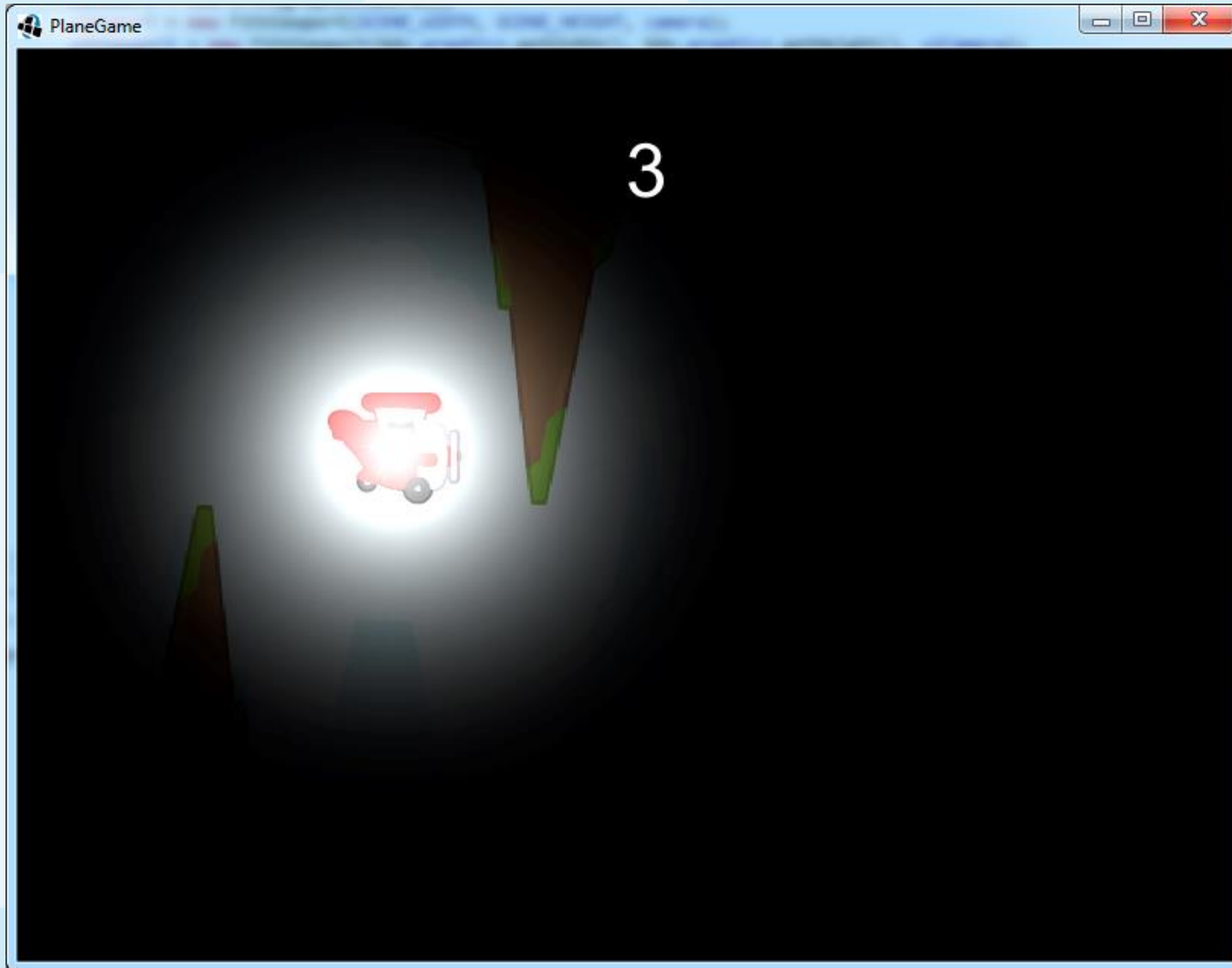
 Colisiones







 Marcador

 GameState: *GameOver*

 Musica y efectos

Ampliaciones y Mejoras



-  Luces/oscuridad
-  Velocidad creciente
-  Fuego al chocar
-  Humo del motor
-  Obstáculos variados
-  Monedas

¿Dominar libGDX?

