








libGDX Crossplatform Game Development Workshop

Flappy Plane

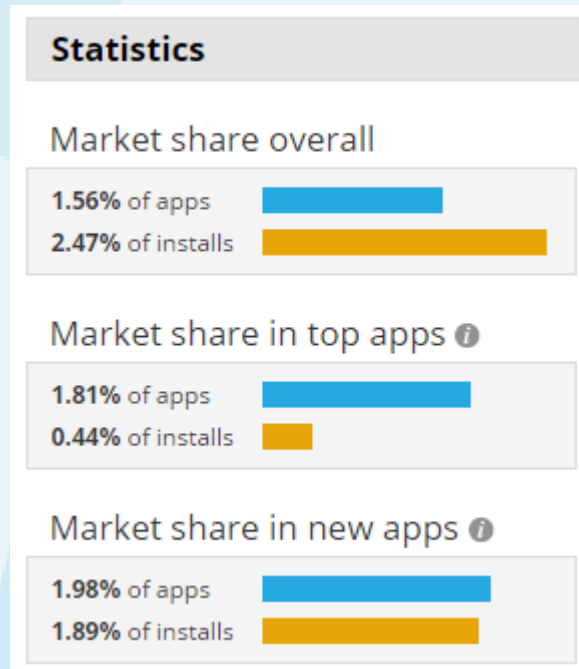


Alberto Cejas Sánchez
David Saltares Márquez

Why **libGDX**?

-  Crossplatform
-  Community
-  Open source
-  Fast
-  Well-documented
-  Multiples levels of abstraction
-  Active development
-  Free!!

Yes, but... really libGDX?



Total apps in Google Play:

1.400.719

Total libGDX apps in Google Play:

21.851

Top categories in Google Play:

1. Education
2. Entertainment
3. Lifestyle

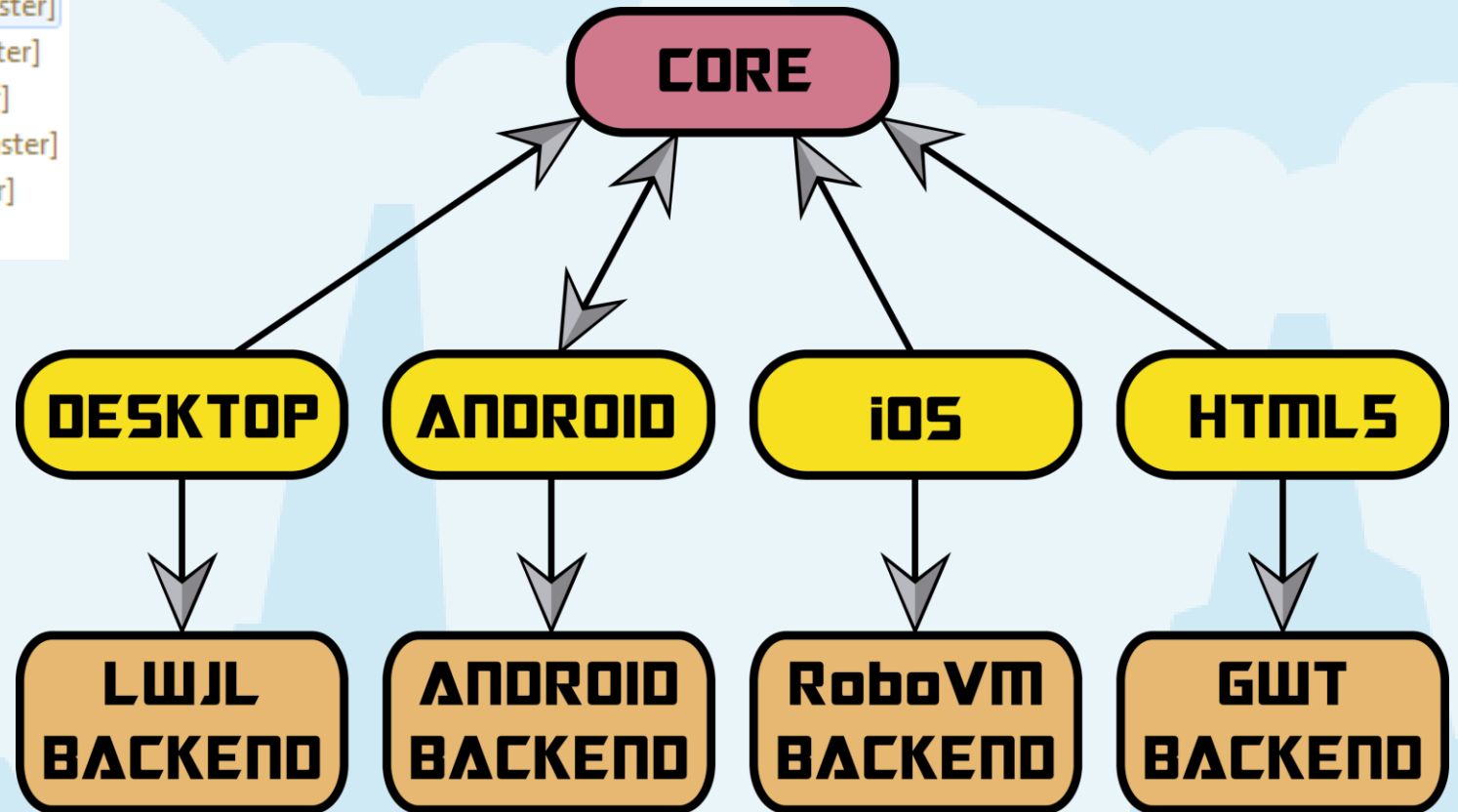


This workshop

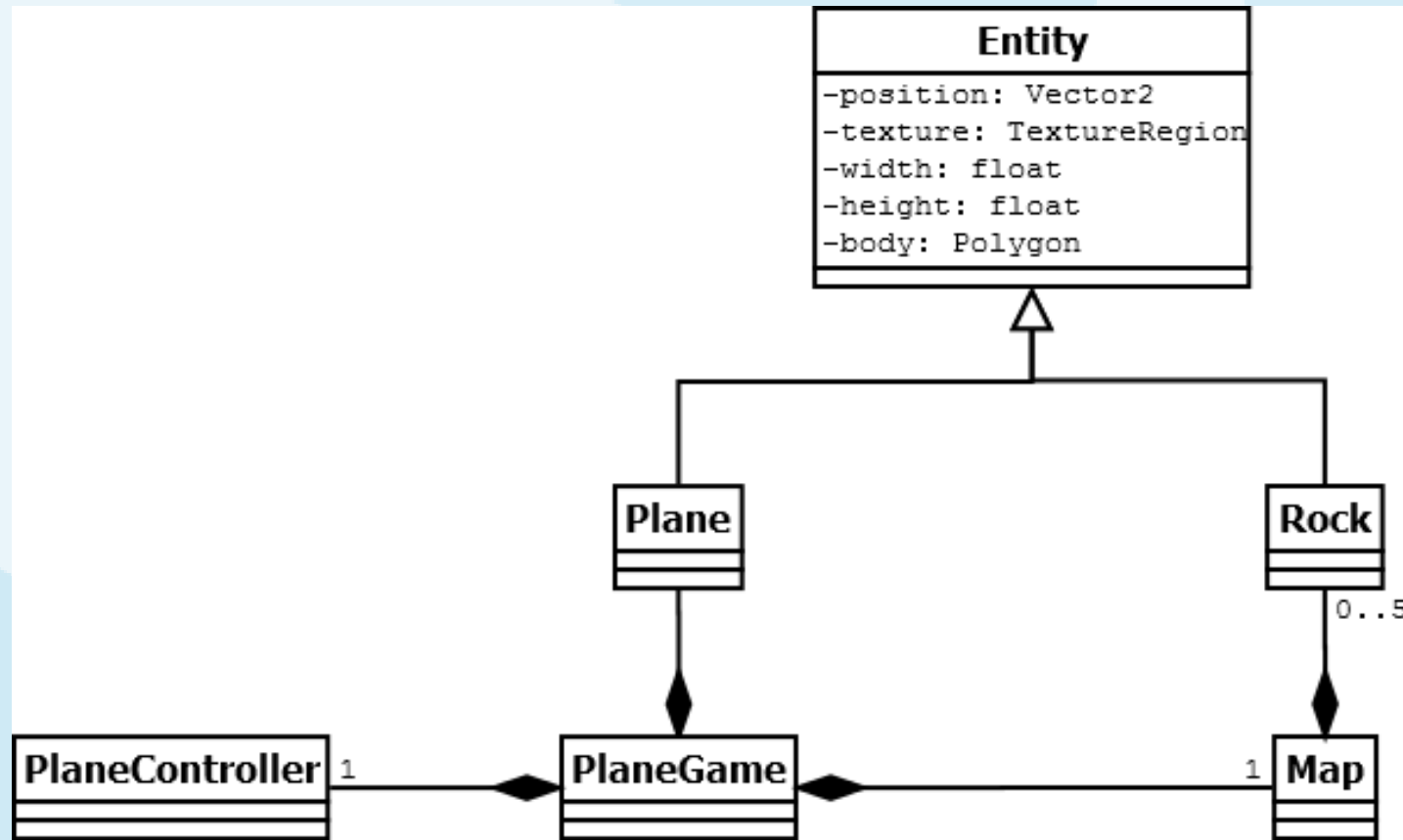


Estructure

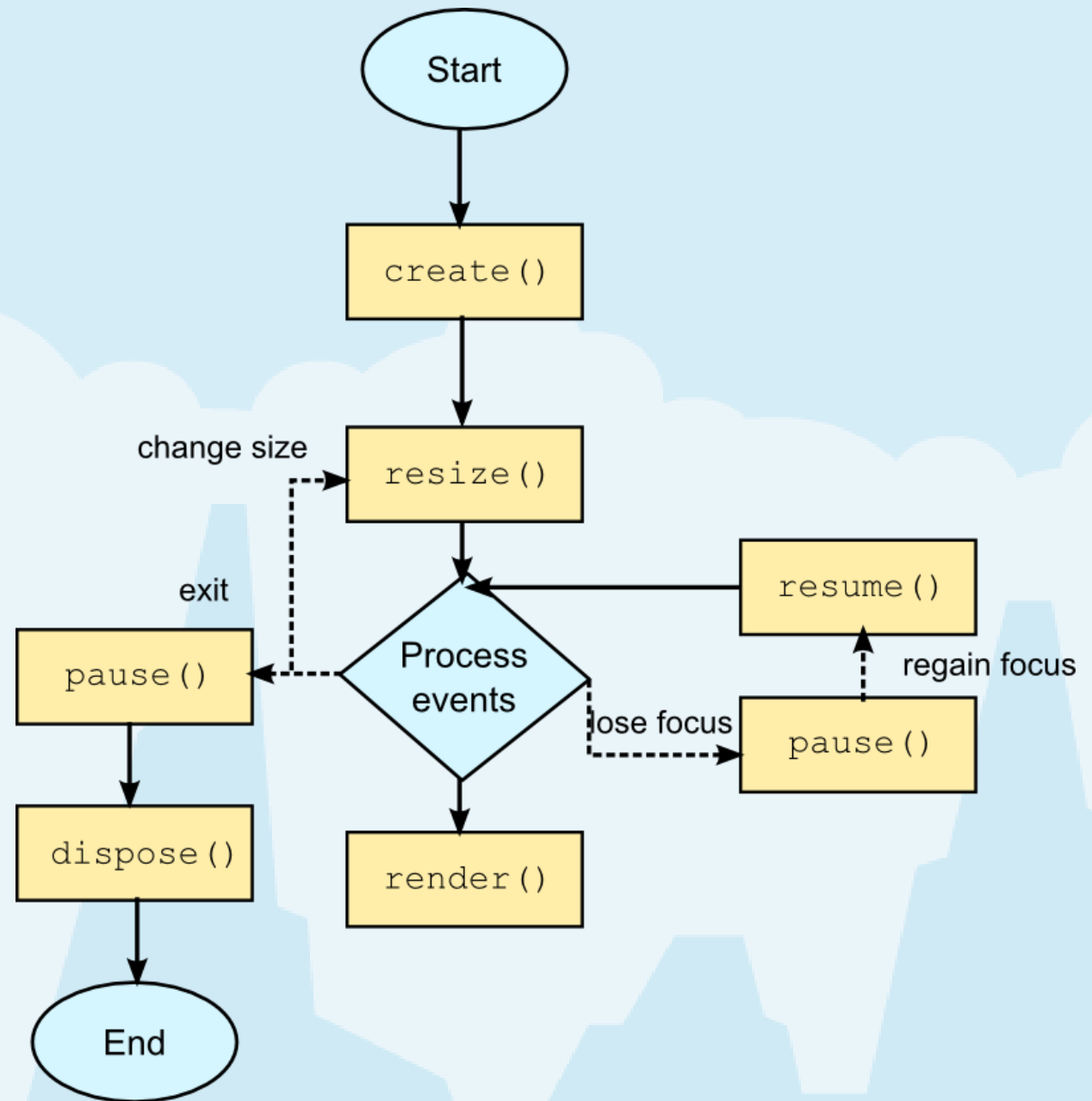
```
> ThePlane-android [flappybird_workshop master]
> theplane-basic-1 [flappybird_workshop master]
> ThePlane-core [flappybird_workshop master]
> ThePlane-desktop [flappybird_workshop master]
> ThePlane-html [flappybird_workshop master]
> ThePlane-ios [flappybird_workshop master]
```



Architecture

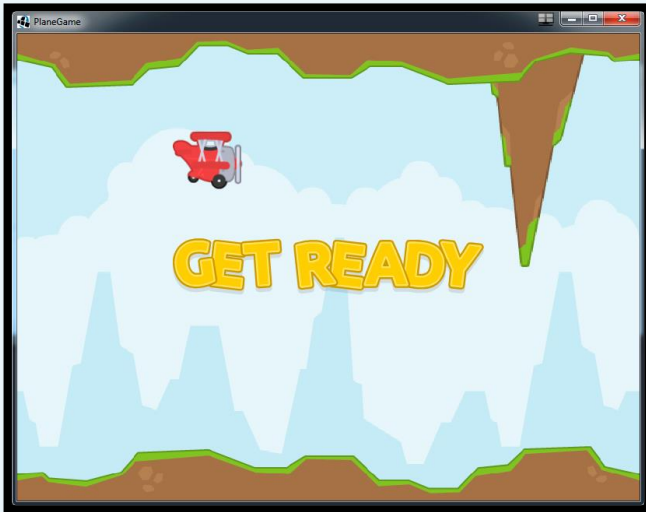


libGDX application lifecycle

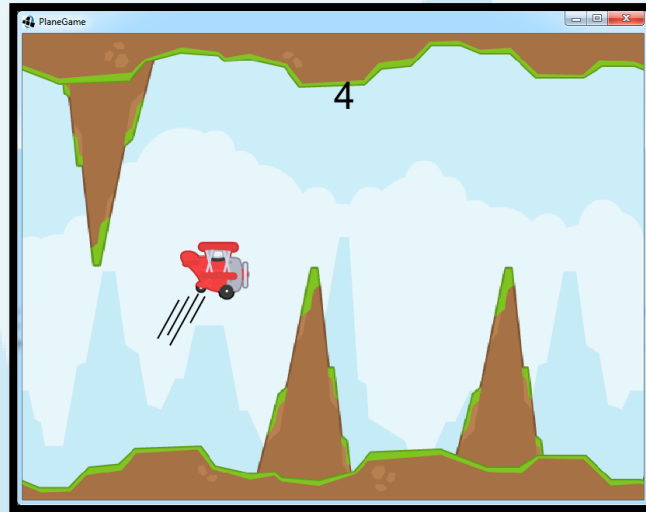


Game States

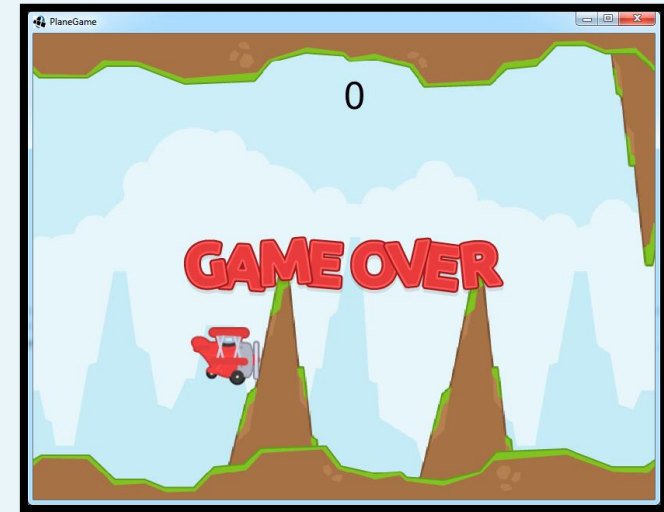
```
static enum GameState {  
    Start, Running, GameOver  
};
```



Start



Running



GameOver

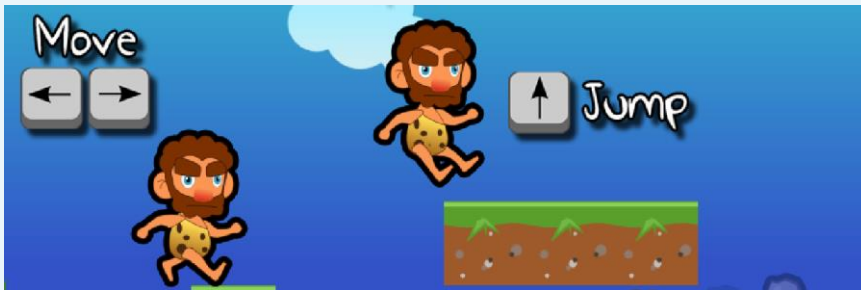
Viewports



StretchViewport



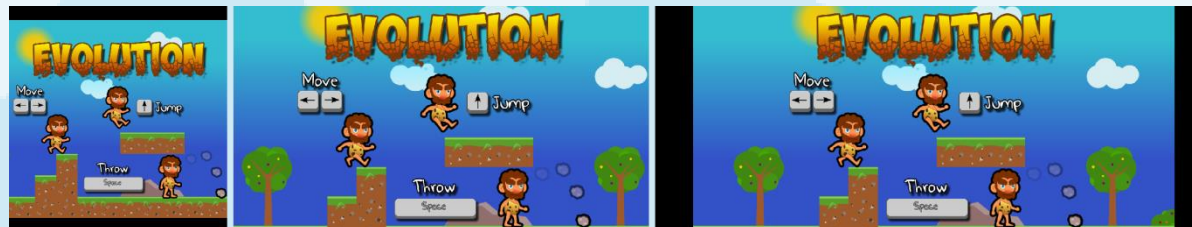
FitViewport



FillViewport



ScreenViewport



ExtendViewport

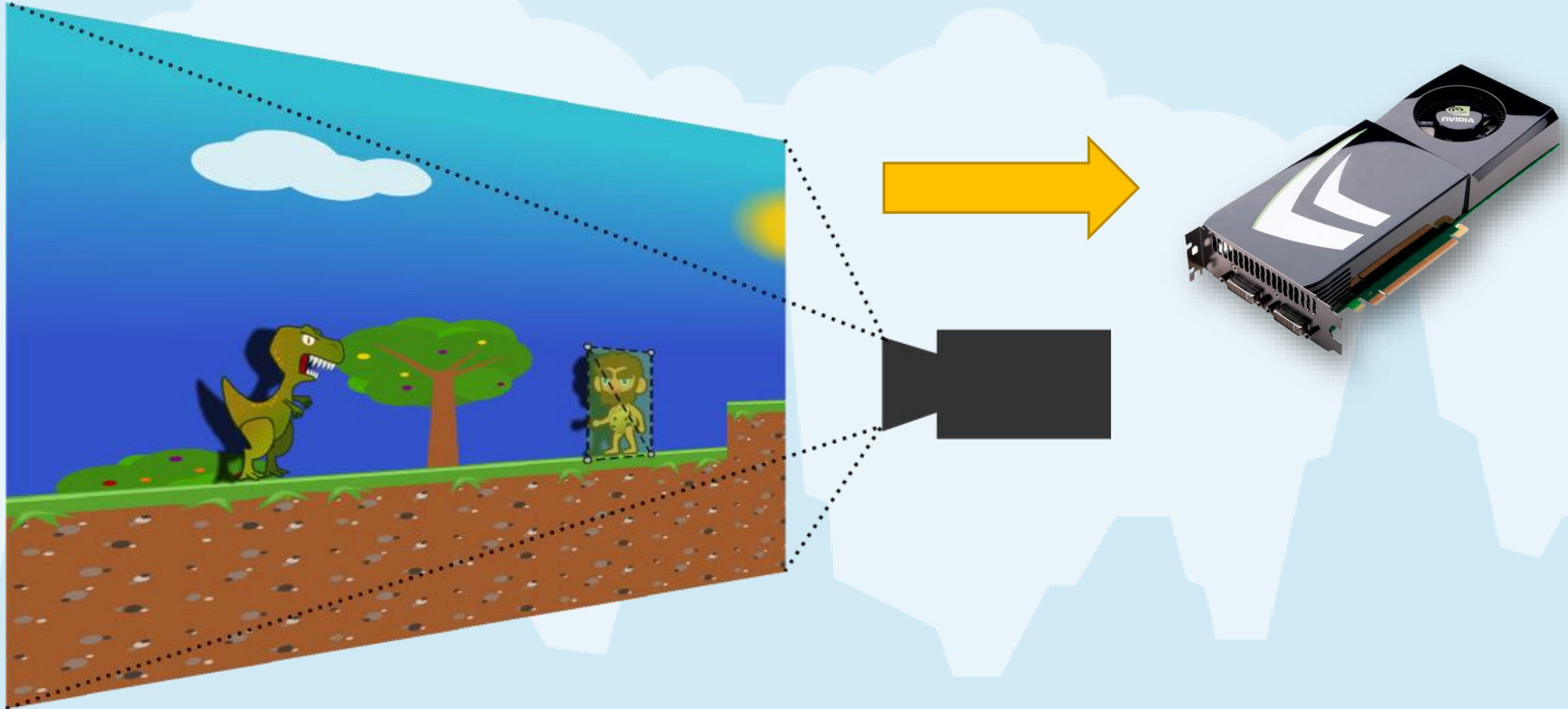
Ok, but... which one today?

```
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);  
}
```

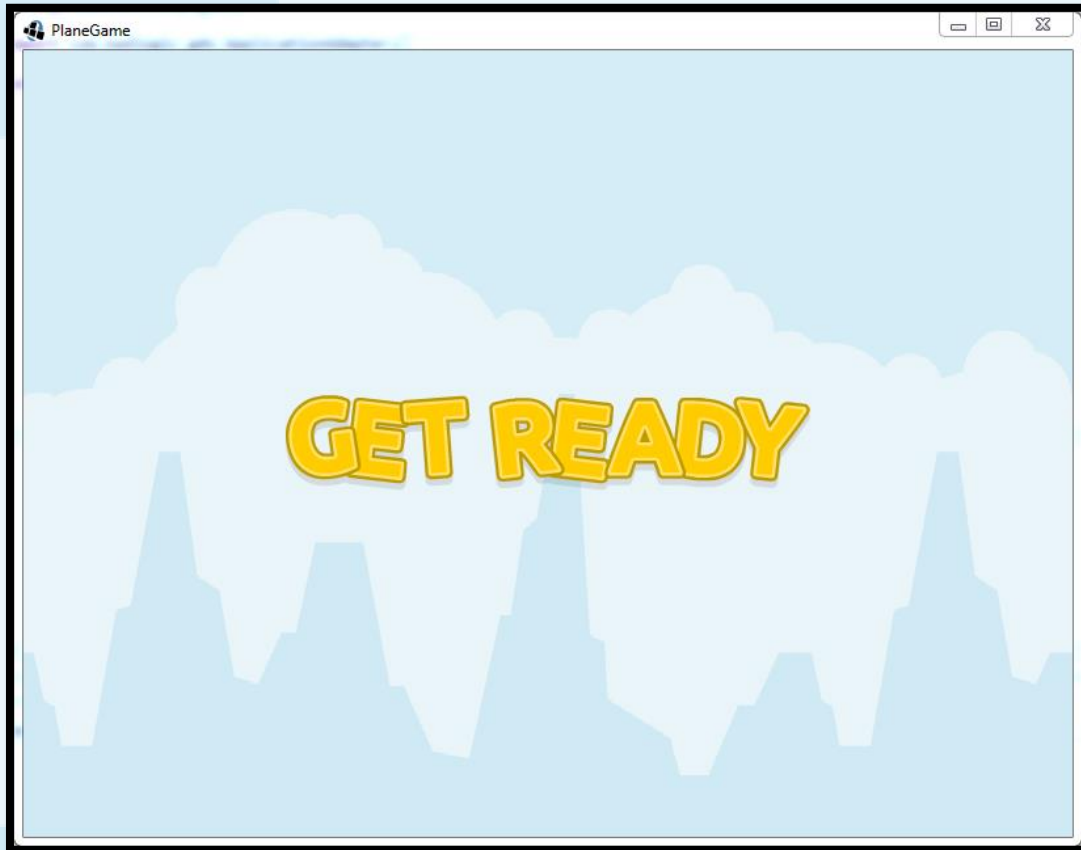
SpriteBatch



Step 1






Result:





To do:

PlaneGame

-  GameState: *Start*
-  Draw world (Map)
-  Draw User Interface (Get ready)

Map




-  Load *background.png*
-  Draw *background.png*

Step 1



Clues:

- Load `ready.png` as a `Texture` within `create()`
- Instantiate `Map` within `create()`
- Switch to `GameState.Running` when user touches the screen (`Gdx.input.justTouched()` within `updateStart()`)
- Load `background.png` as a `Texture` within `create()` of `Map`
- Implement `Map` `draw` method using `batch.draw(Texture texture, float x, float y, float width, float height)`. Make use of it within `drawWorld()` in `PlaneGame`
- Implement `drawUI()` which is responsible for drawing `ready` in the middle of the screen while the game is in `GameState.Start` state. Make use of `batch.draw(Texture texture, float x, float y)`. You will need `getWorldWidth()` and `getWorldHeight()` from `Viewport` class, besides of `getWidth()` and `getHeight()` from `Texture` class.
- Free resources in `dispose()`

PlaneGame

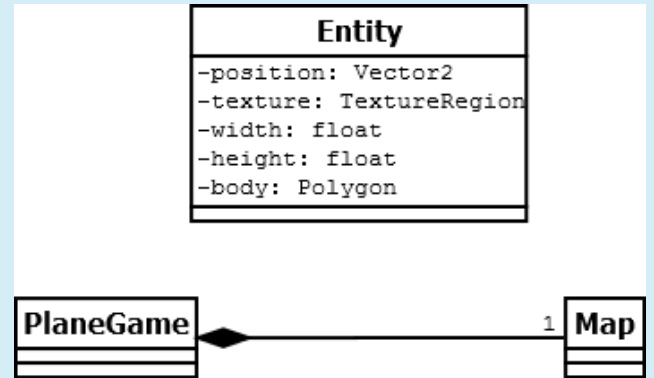
-  `GameState: Start`
-  `Draw world (Map)`
-  `Draw User Interface (Get ready)`

Map

-  `Load background.png`
-  `Draw background.png`

Step 2






To do:



Entity

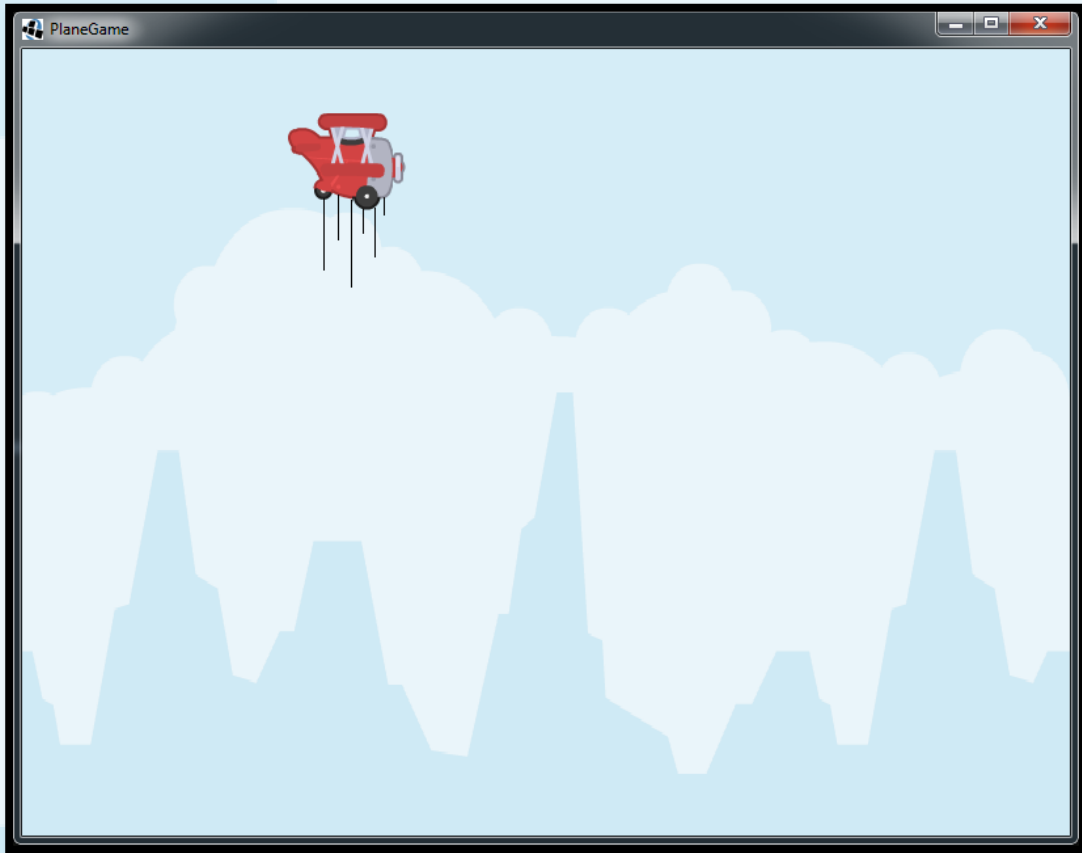
Clues:

- The origin of a body must be its center: `body.setOrigin(x,y)`
- `setPosition`, `setRotation` and `translate` must apply to the `body` too
- Use `draw(TextureRegion region, float x, float y, float width, float height)` in order to implement `entitiy.draw(SpriteBatch batch)`
- Use `boolean Intersector.overlapConvexPolygons(Polygon p1, Polygon p2)` in order to check if two `bodies` **collide/overlap**

Original	Rotated 180° 
 Origin	 Origin
 Origin	 Origin

Step 3

Result:



To do:

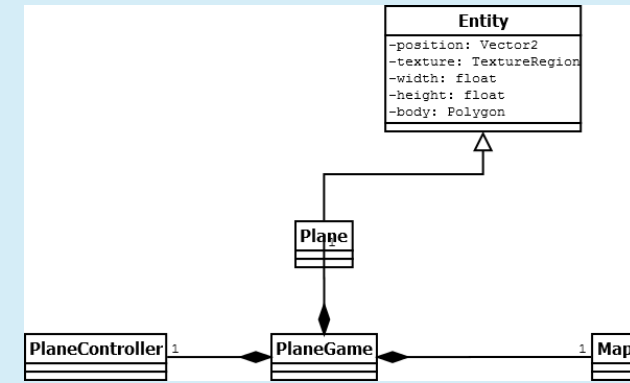
✈️ **Plane**

✈️ **PlaneGame**

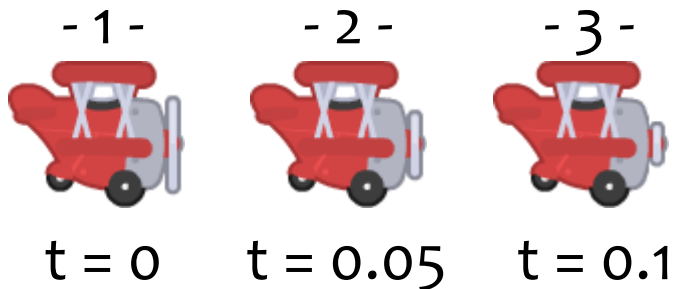
✈️ GameState: *Running*

✈️ Add plane and update it

✈️ Draw plane



Step 3 – introduction



Animation

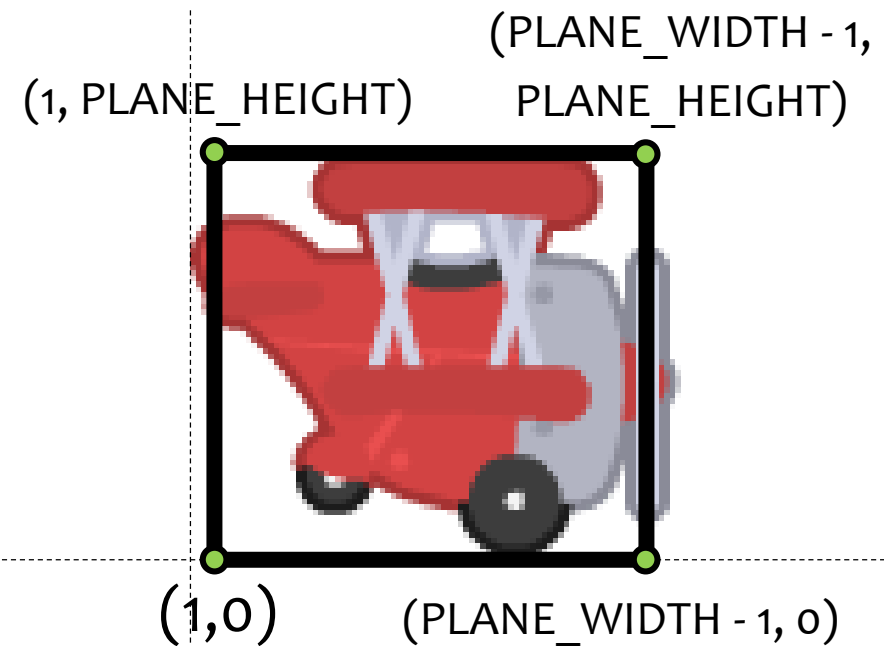
statetime = 0.03



statetime = 0.08



statetime = 0.24



static Polygon createShape()


Step 3 – part 1


Clues:


- Within the constructor of **Plane**, load **plane1.png** into **frame1** which is an instance of **Texture** class. Repeat it with **frame2** y **frame3**.
- Within the same constructor, initialize **animation** through **Animation(float frameDuration, TextureRegion... keyFrames)**, where **frameDuration** is **0.05** seconds and **keyframes** are **frame1**, **frame2** y **frame3**. Make the animation loops: **animation.setPlayMode(PlayMode.LOOP);**
- Within **reset()**, set the starting velocity for the **Plane** to **(0,0)** and a desired starting position
- Within **update(float delta)**, increase **stateTime** with **delta** time. Add gravity force to **velocity** vector through **add** method and scale it to the aforementioned **delta** time with **scl**. Then move the **Plane** instance with **translate** according to its resulting **velocity**.
- Within **draw()**, set the region to draw according to the current **stateTime**, making use of **setRegion** (previously defined in the **Entity** class) and **animation.getKeyFrame**. Don't forget to call **draw** from **Entity** -> **super.draw(batch)**
- Free resources with **dispose()**

 **Plane**

 **PlaneGame**

 GameState: *Running*

 Add plane and update it

 Draw plane


Step 3 – part 2


Clues:


- `touchdown` method from `PlaneController` will be executed whenever the user touches the screen, therefore this is the place to update the plane's velocity, keeping the same X velocity but setting its Y velocity to `PLANE_JUMP_IMPULSE`
- Instantiate `Plane` and `PlaneController` within `create()` from `PlaneGame`
- Implement `updateCamera()` so as the `camera` stays in the same position as the `plane`, but adding 20 to its X value. Then call `camera.update()`
- Within `updateStart()`, set `PlaneController` as the input processor and set a starting speed for `plane`
- Within `updateRunning()`, update `plane` with the elapse `delta` time
- Draw `plane` within `drawWorld()`
- Free resources in `dispose()`

 **Plane**

 **PlaneGame**

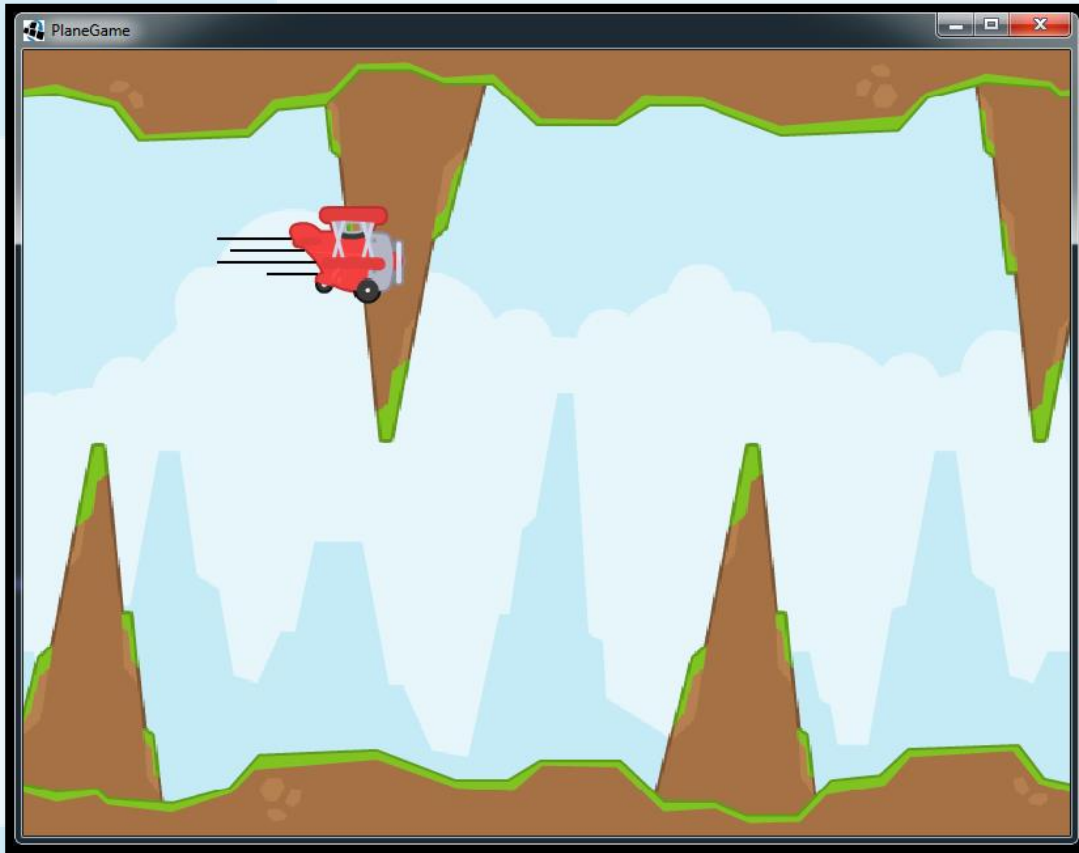
 GameState: *Running*

 Add plane and update it

 Draw plane

Step 4

Result:



To do:

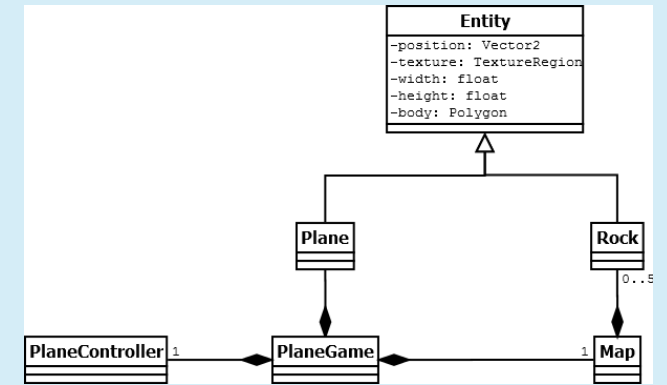
🚁 Rock

🚁 PlaneGame

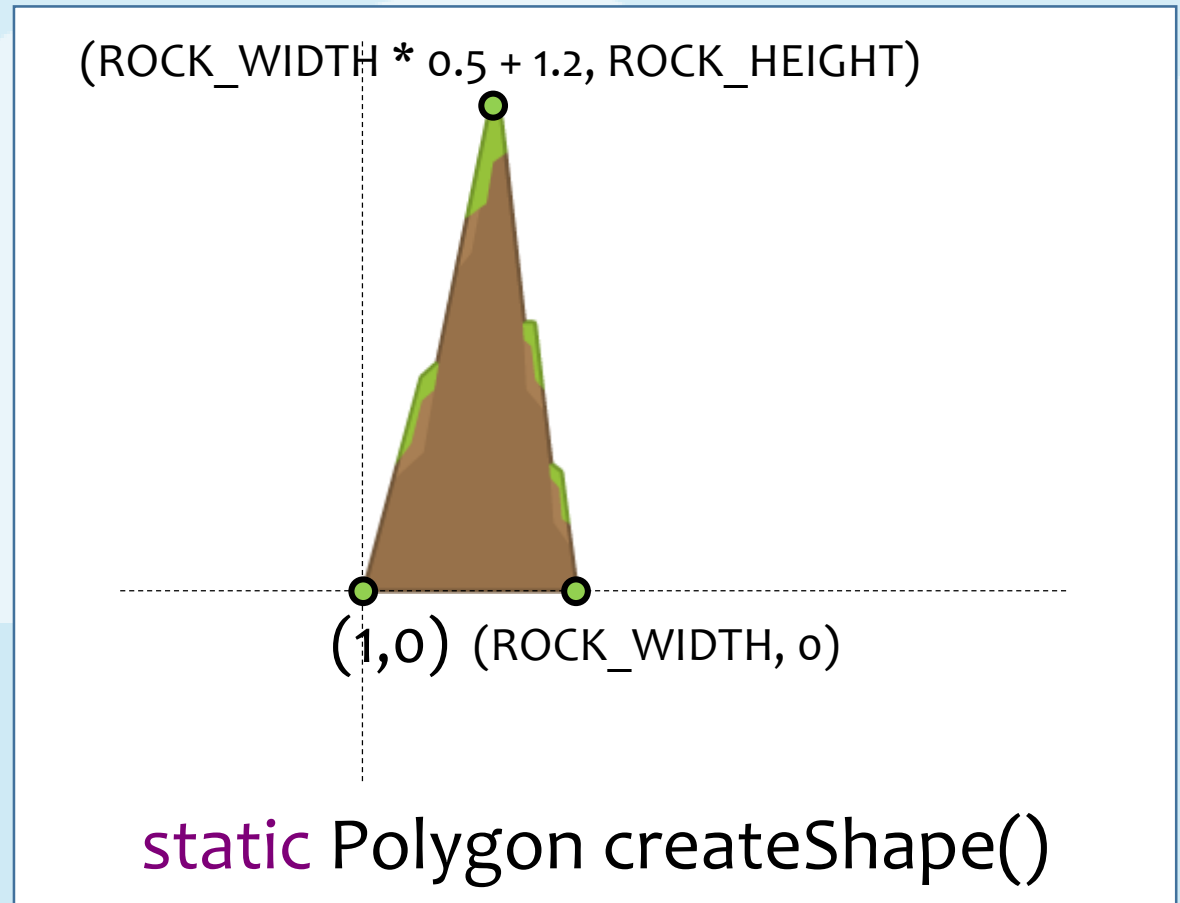
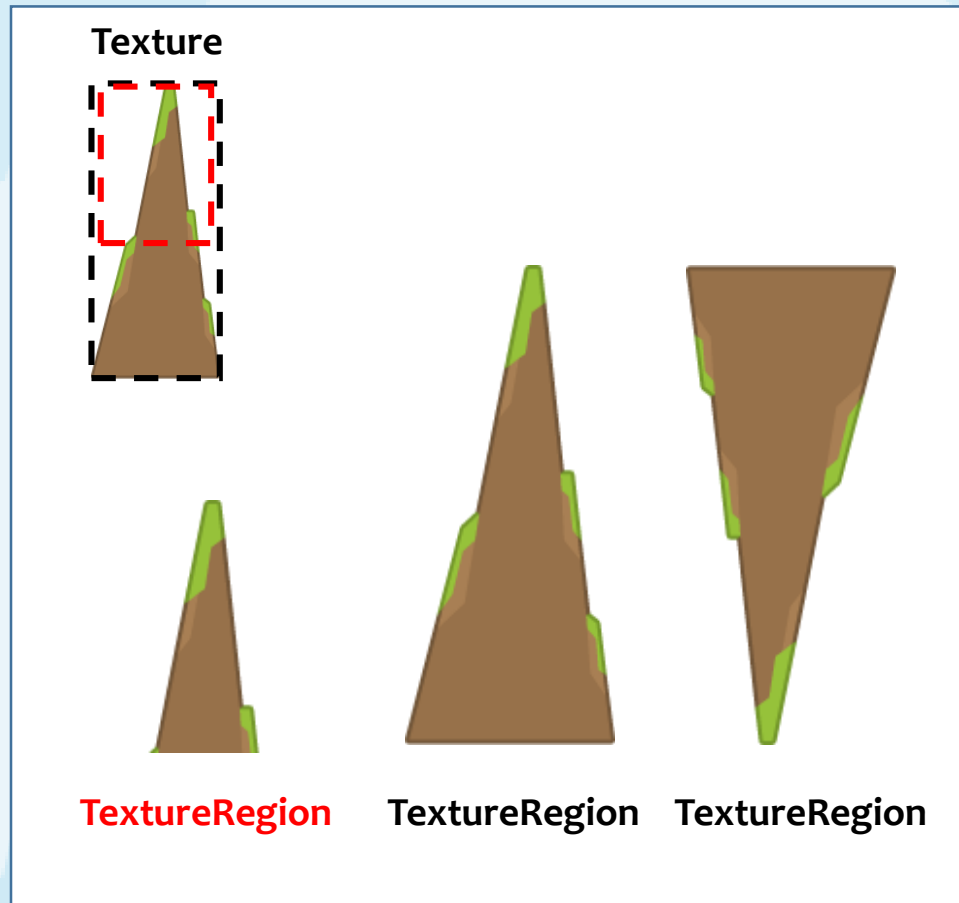
🚁 GameState: *Running*

🚁 Map

🚁 rocks, ceiling, ground

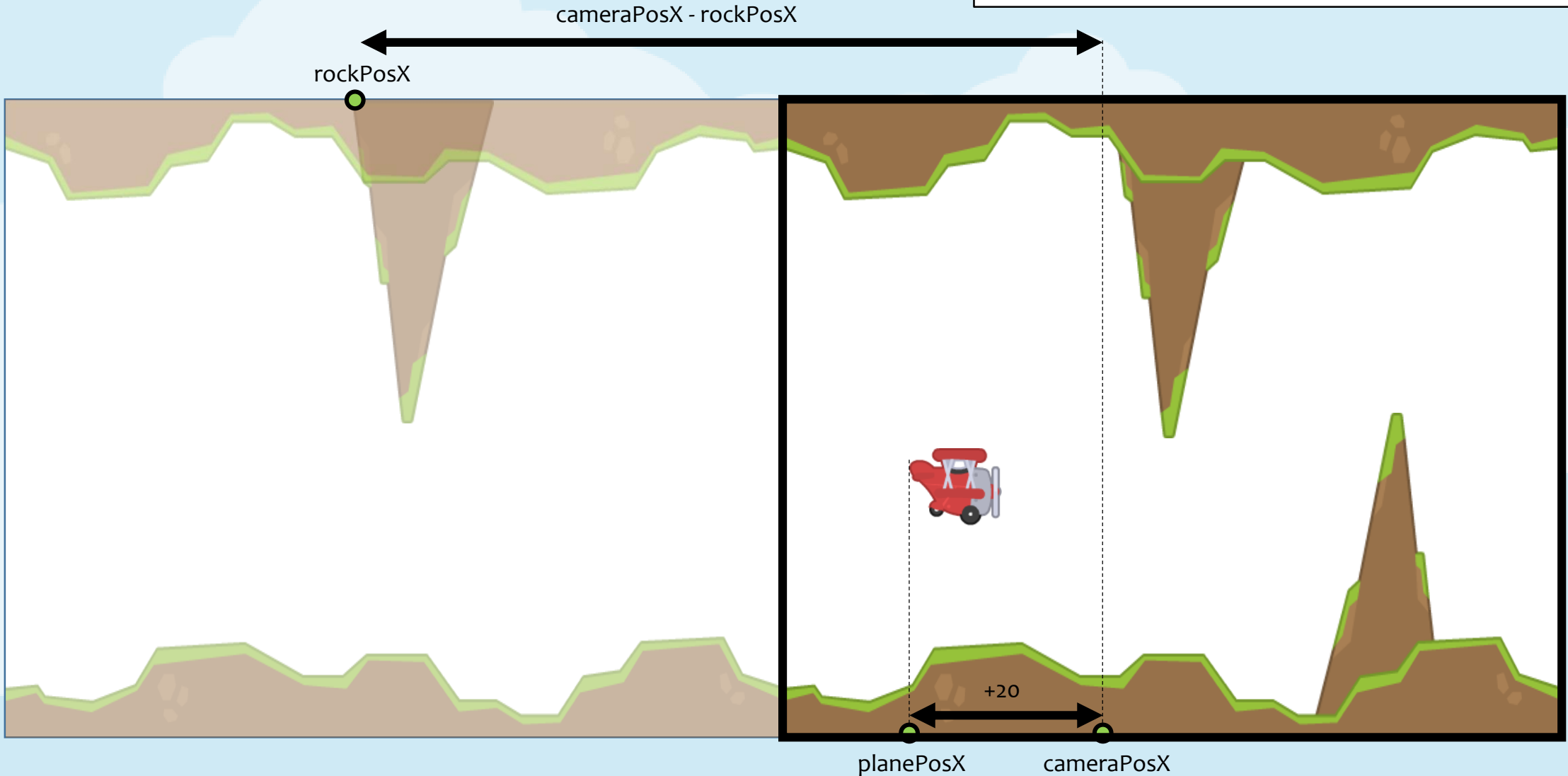


Paso 4 – introduction



Paso 4 – introduction

```
if (cameraPosX - rockPosX > SCENE_WIDTH*0.5 + ROCK_WIDTH) {  
    // If not visible, translate the rock  
}
```



Step 4 – part 1

Clues:

- Implement **Rock**, making use of the received position (x,y) in its constructor. Moreover, implement **createShape()** according to the previous slide.
- Extend **Mapa** with the **Texture** and **TextureRegion** from the **Rock** (**rock.png**) and the ground (**ground.png**). Take into account that we must have a flipped region for each of them. We can use: **flip(true, true)**
- Implement **reset()**, clearing the rocks **Array** with **clear()** and creating 5 new instances. To this end, a random boolean will be generated through **MathUtils.randomBoolean()**. The rock's **X** position will be $60 + i * 25$, where **i** is the rock's number. The **Y** position will depend on the former boolean, if **true**, **30**, otherwise **0**. The boolean will also be useful to determine if we are rotating the body 180° and using the flipped region. Do not forget to add the rocks to the **Array**.

 **Rock**

 **PlaneGame**

 **GameState: Running**

 **Map**

 **rocks, ceiling, ground**


Step 4 – part 2

Clues:


- Implement `update()` from `Map` so as the hidden rocks increase their `X` position in `5 * 25`. Take into account that the boolean stuff from the previous paragraph applies to to this one. Moreover, don't forget to reset the rotation to 0 before applying a new one.
- Add an extra parameter `offsetX` to `draw` from `Map` class. Also draw the rocks through its base method `draw(SpriteBatch batch)`. Then draw twice the `ground` and the `ceiling`. The first one in the `X` position that `offsetX` determines. The second one in `offsetX + GROUND_WIDTH`
- Within `updateRunning()` from `PlaneGame`, increase the flown distance in the accumulator `groundOffsetX` with `SCENE_WIDTH` everytime that the plane flies `SCENE_WIDTH` 1.5 times. Then, call `update` from `Map` with `X` position of the `camera`.
- Free resources in `dispose()`

 **Rock**

 **PlaneGame**

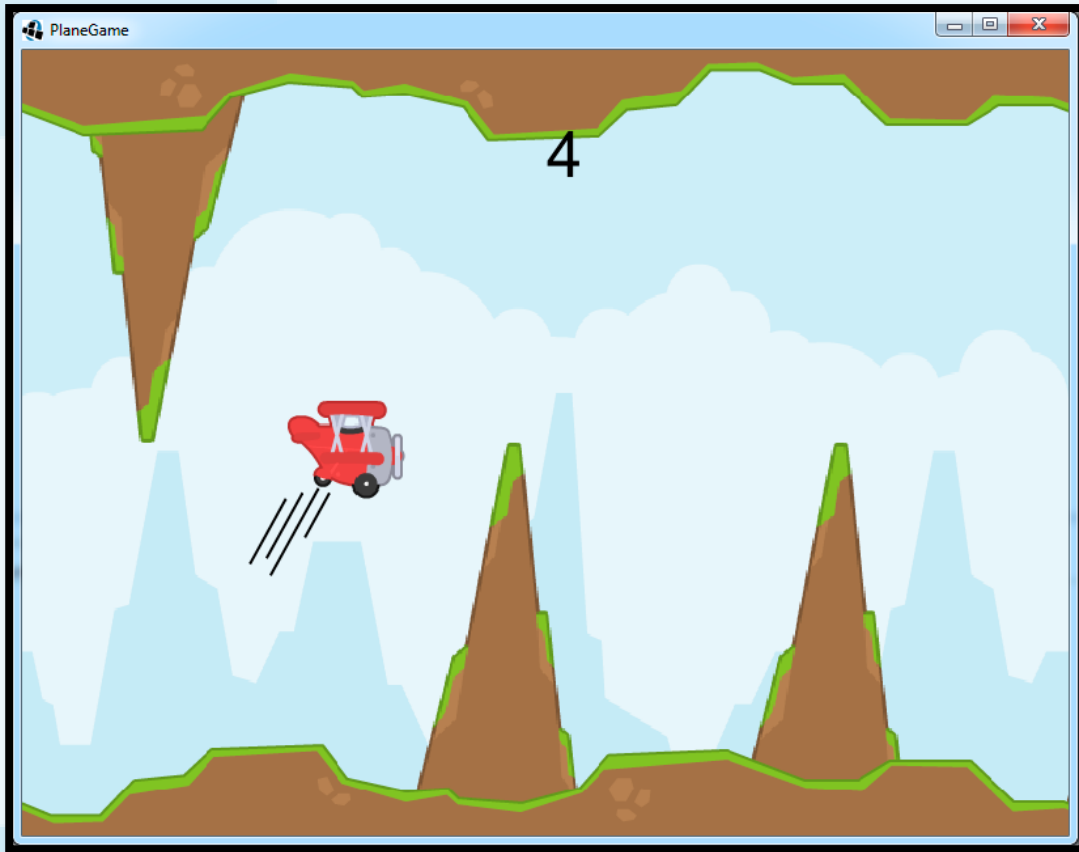
 `GameState: Running`

 **Map**

 `rocks, ceiling, ground`






Step 5

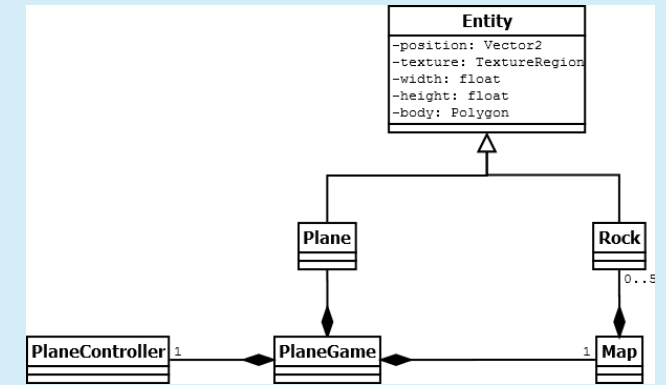
Result :



To do:

PlaneGame

-  GameState: *Running*
-  Collisions
-  Score
-  GameState: *GameOver*
-  Music y effects




Step 5 – part 1

Clues:

- Create a `score` variable with `0` value
- Instantiate `music.mp3` as `Music` with `Gdx.audio.newMusic(Gdx.files.internal("music.mp3"))`. Make it loop with `setLooping(true)` and call `play()`.
- Repeat the previous instructions with `explode.wav`. Take into account that instead of being of type `Music`, it's a `Sound`.
- Within `update` from `Map`, set `counted` to false for each `Rock` that you translate.
- Implement `checkCollisions()` so as it checks whether every and each of the rocks `collide` with the `plane`. If so, `setInputProcessor(null)`, set the plane's velocity to `0` and switch to `GameState.GameOver` (play `explode` too). Finally, you can take advantage of the necessary loop to increase the `score` whenever the `plane` has left behind the position of the current `Rock`. Don't count twice the same `Rock`!. Check collisions with `ceiling` y `ground`, but instead of using `collide`, the top limit will be `SCENE_HEIGHT - GROUND_HEIGHT + 2` while the bottom limit will be `GROUND_HEIGHT - 2`.
- Free resources in `dispose()`

PlaneGame

 GameState: *Running*

 Collisions

 Score

 GameState: *GameOver*


 Music y effects

Step 5 – part 2

Clues :

- Instantiate `arial.fnt` as a `BitmapFont` called `font` using `Gdx.files.internal`. Set its `Color` to `Color.BLACK` with `setColor`
- Load `gameover.png` as a `Texture` within `create()`
- Draw `gameover` while the game is under `GameState.GameOver` just as done before with `ready`.
- Draw the score in the position (`SCENE_WIDTH * 0.5, SCENE_HEIGHT - 60`).
- Free resources in `dispose()`

PlaneGame

 GameState: *Running*

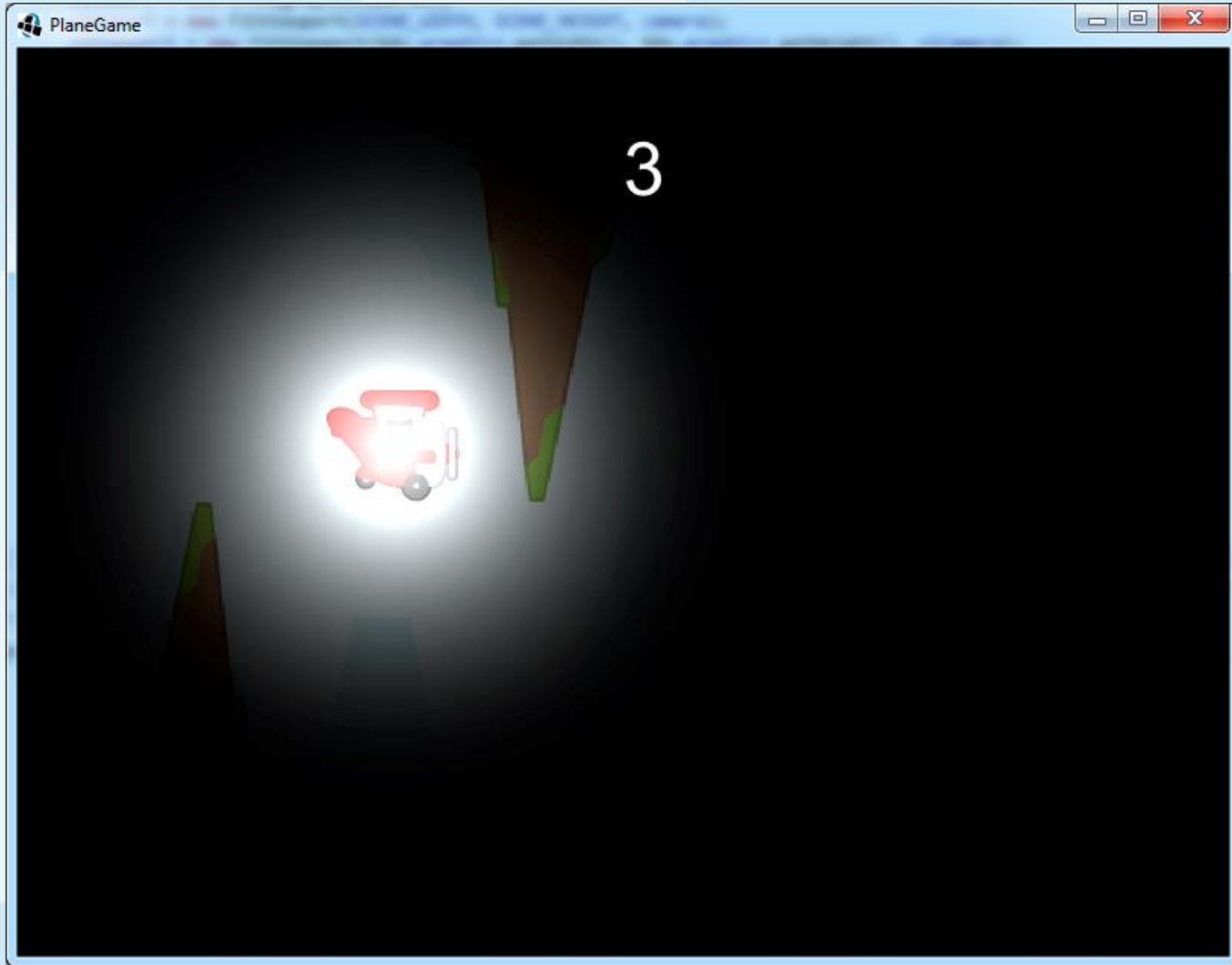
 Collisions







 Score

 GameState: *GameOver*

 Music y effects

Improving the game



-  Lights
-  Increasing velocity
-  Explosions
-  Engine smoke
-  More obstacles
-  Coins

Mastering libGDX?

