Kotlin

Jetpack Compose - Asteroids

Alejandro M. Ogalla López

02-03-2022

Índice

1 Modificaciones realizadas

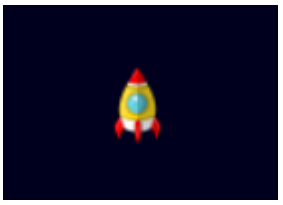
En este documento se detallan todas las modificaciones que se han realizado sobre el repositorio Asteroids-compose-for-desktop. La lista completa de ramas del repositorio puede verse en este enlace, y la lista de pull requests en este otro.

1.1 Imagen de la nave

Esta modificación se ha hecho a través de la rama **CambiarImagenNave** por medio de la Pull Request #1 - Cambiar imagen nave. En esta implementación se ha añadido el siguiente código en la función **Ship**, que se encuentra en el archivo *ShipComponent.kt*.

```
val imageModifier = Modifier
Image(
    bitmap = imageFromResource( path: "nave.png"),
     contentDescription: "image",
    imageModifier,
    contentScale = ContentScale.Fit
)
```

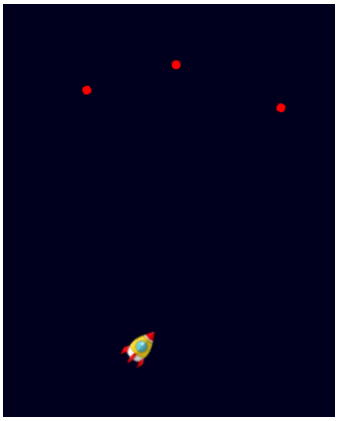
Con esto, la nave ahora se ve de esta forma:



1.2 Diferentes disparos

Se ha modificado la forma de los disparos de la nave mediante la pull request #7 - Forma de las balas modificada, proveniente de la rama BalasDiferentes.

Con esos cambios, el resultado es el siguiente:



1.3 Clase abstracta EnemyData

En el juego, los enemigos son tres asteroides, que en realidad son tres objetos iguales, instancias de AsteroidData. En las PR #3 Enemigo genérico y #6 Corregido valor del tamaño de un enemigo, que parten de la rama EnemigoGenerico se ha creado una clase abstracta **EnemyData** que representa a cualquier enemigo. Por lo tanto, una vez creada esta clase, definimos que AsteroidData herede de EnemyData (que a su vez hereda de GameObject), ya que es un caso particular de enemigo.

En el fichero *GameObject* definimos esta clase:

```
abstract class EnemyData(speed: Double = 0.0, angle: Double = 0.0, position: Vector2 = Vector2.ZERO) :
    GameObject(speed, angle, position) {
    override var size: Double = 120.0
}
```

Y modificamos la herencia en AsteroidData:

```
class AsteroidData(speed: Double = 0.0, angle: Double = 0.0, position: Vector2 = Vector2.ZERO) :
    EnemyData(speed, angle, position) {
    override var size: Double = 120.0
}
```

En principio, con esta clase abstracta no hemos conseguido nada, mas allá de una definición más intuitiva y clara del código. Sin embargo, esto permitirá la generación de enemigos diferentes con comportamiento similar, gracias a la herencia de una misma clase (EnemyData).

1.4 Creación de varios enemigos diferentes

A partir de la clase abstracta vista en 1.3, definimos un tipo de enemigo diferente: un alienígena de Los Simpson. Esto se hace en el fichero *GameObject.kt*, tal como puede verse en la pull request #4 - Varios enemigos distintos:

```
class AsteroidData(speed: Double = 0.0, angle: Double = 0.0, position: Vector2 = Vector2.ZERO) :
    EnemyData(speed, angle, position) {
    override var size: Double = 120.0
}

class SimpsonAlienData(speed: Double = 0.0, angle: Double = 0.0, position: Vector2 = Vector2.ZERO) :
    EnemyData(speed, angle, position) {
    override var size: Double = 120.0
}
```

Además, en la PR #5 - Varios enemigos distintos se define el comportamiento del juego con respecto a este nuevo enemigo. En primer lugar, en el fichero *Main* debemos instanciar el enemigo cuando exista en la lista de objetos *gameObjects*:

```
game.gameObjects.forEach {  it: GameObject
    when (it) {
        is ShipData -> Ship(it)
        is BulletData -> Bullet(it)
        is AsteroidData -> Asteroid(it)
        is SimpsonAlienData -> SimpsonAlien(it)
```

SimpsonAlien es una función 'composable' que se define en *SimpsonAlienComponent.kt*:

```
@Composable
fun SimpsonAlien(simpsonAlienDataData: SimpsonAlienData) {
    val simpsonAlienSize = simpsonAlienDataData.size.dp
    Box(
        Modifier
            .offset(simpsonAlienDataData.xOffset, simpsonAlienDataData.yOffset)
            .size(simpsonAlienSize)
            .rotate(simpsonAlienDataData.angle.toFloat())
            .clip(CircleShape)
            .background(Color.Transparent)
    )
```

Por otra parte, en la función **update** del fichero *Game.kt* creamos una lista con todos los enemigos, ya que ahora no son sólo de un tipo:

```
val asteroids = gameObjects.filterIsInstance<AsteroidData>()
val aliens = gameObjects.filterIsInstance<SimpsonAlienData>()

val listaEnemigos: MutableList<EnemyData> = asteroids.toMutableList()
listaEnemigos.addAll(aliens.toMutableList())
```

En las siguientes líneas, es necesario realizar la diferenciación entre unos enemigos y otros, porque al impactar una bala contra ellos, su comportamiento es diferente: aparecen dos nuevos enemigos (más pequeños) del mismo tipo del que acaba de ser eliminado. Por lo tanto, utilizamos estas cláusulas *if*:

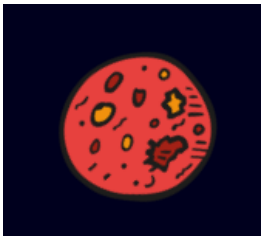
```
if (enemy is AsteroidData) {
    gameObjects.add(AsteroidData(
        speed: enemy.speed * 2,
        angle: Random.nextDouble() * 360.0,
        enemy.position
    ).apply { this: AsteroidData
        size = enemy.size / 2
    })
}

if (enemy is SimpsonAlienData) {
    gameObjects.add(SimpsonAlienData(
        speed: enemy.speed * 2,
        angle: Random.nextDouble() * 360.0,
        enemy.position
    ).apply { this: SimpsonAlienData
        size = enemy.size / 2
    })
}
```

1.5 Imagen de los enemigos

Al igual que en 1.1 hicimos con la imagen de la nave, también se ha cambiado la del enemigo AsteroidData, como puede verse en la pull request #2 - Cambiar imagen asteroides.

```
})
val imageModifier = Modifier
Image(
    bitmap = imageFromResource( path: "asteroid.png"),
    contentDescription: "image",
    imageModifier,
    contentScale = ContentScale.Fit
)
```

Con este cambio, el asteroide ahora se ve así:

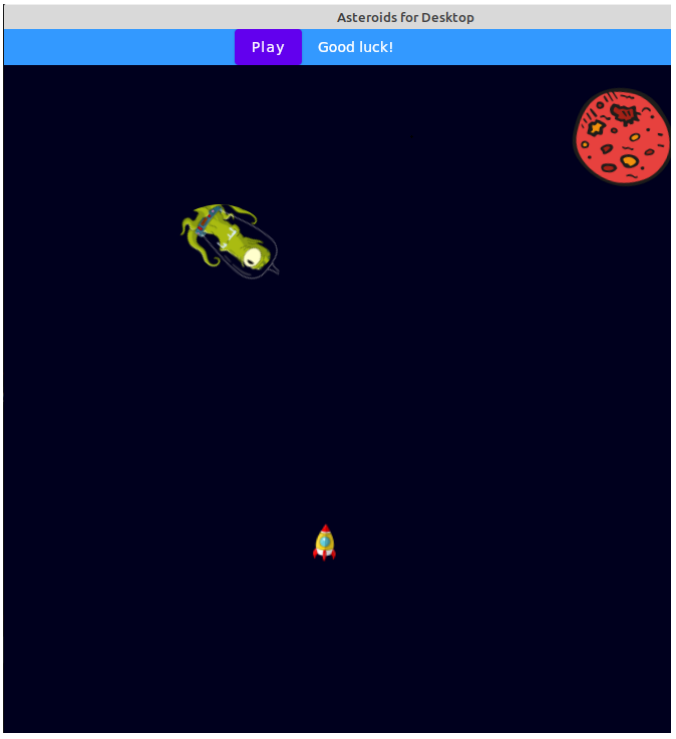


1.5.1 Imágenes diferentes para cada enemigo

Una de las cosas que podemos hacer, ahora que hemos definido enemigos diferentes en 1.4, es que tenga, cada uno, una imagen distinta.

Del mismo modo, realizamos la modificación para SimpsonAlienData, siendo este el resultado:

Gracias a la diferenciación que aporta la herencia, podemos ver dos enemigos visualmente diferentes:



1.6 Enemigo final

Otra mejora que se añade al juego es la introducción de un enemigo final que aparecerá cuando se hayan eliminado todos los enemigos que aparecen en la escena inicial. Estas adiciones se pueden ver en la PR #8 - Final boss

Creamos el fichero *FinalBossComponent.kt*, que incorpora la función 'composable' **FinalBoss**:

```
@Composable
fun FinalBoss(finalBossData: FinalBossData) {
    val finalBossSize = finalBossData.size.dp
    Box(
        Modifier
            .offset(finalBossData.xOffset, finalBossData.yOffset)
            .size(finalBossSize)
            .rotate(finalBossData.angle.toFloat())
            .clip(CircleShape)
            .background(Color.Transparent)
    )
```

Y creamos la clase *FinalBossData*, que es un tipo de *EnemyData*:

```
class FinalBossData(speed: Double = 0.0, angle: Double = 0.0, position: Vector2 = Vector2.ZERO) :
    EnemyData(speed, angle, position) {
    override var size: Double = 120.0
}
```

También es necesario añadirlo a la lista de enemigos totales, aunque aún no vaya a aparecer en escena:

```
val listaEnemigosTotales:MutableList<EnemyData> = asteroids.toMutableList()
listaEnemigosTotales.addAll(aliens.toMutableList())
listaEnemigosTotales.addAll(finalBoss.toMutableList())
```

Añadimos una cláusula *if* en *Game.kt*, además de las mencionadas en 1.4, que define el comportamiento del juego con un enemigo final:

```
if (enemy is FinalBossData) {
    // Bullet <-> FinalBoss interaction
    if (enemy.position.distanceTo(least.position) < enemy.size) {
        gameObjects.remove(least)
        vidasFinalBoss--
        if (vidasFinalBoss <= 0) {
            finalBossEliminado = true
            gameObjects.remove(enemy)
            winGame()
        }
    }
}
```

Definimos también dos tipos de comportamientos diferentes. Con las listas de enemigos definidas anteriormente, indicamos las siguientes dos posibilidades:

- Hemos derrotado a todos los enemigos 'normales' (nivel 1).
- Hemos derrotado, además, al enemigo final, es decir, a todos los enemigos posibles del juego.

Para diferenciar estas dos opciones, creamos una variable Boolean que nos indicará en todo momento si el enemigo final ya ha sido derrotado o no.

```
var finalBossEliminado: Boolean = false
```

Con ello, definimos el comportamiento del juego en función de si hemos derrotado al enemigo final. Si no quedan enemigos pero el enemigo final aún no ha sido eliminado, lo sacamos a escena, es decir, se llamará a la función **showFinalBoss**:

```
// Enemy <-> Ship interaction
if (listaEnemigosTotales.any { asteroid -> ship.overlapsWith(asteroid) }) {
    endGame()
}

// Win LEVEL 1 condition
if (listaEnemigosTotales.isEmpty() && !finalBossEliminado) {
    showFinalBoss()
}
```

En cambio, cuando éste sea eliminado, se llamará a la función **endGame**, y el juego finaliza.

1.7 Vidas del enemigo final

El enemigo final es más difícil de derrotar que los demás. Por eso mismo, por medio de las PR #8 - Final boss, #9 - Modificado el número de vidas del enemigo final y #10 - Corregido error tras la aparición del enemigo final, se define el enemigo final como un enemigo al que hay que disparar 10 veces hasta derrotarle.

En el comportamiento del FinalBossData definimos que pierda una de sus vidas con cada disparo, y que el juego termine con victoria si éstas llegan a cero.

```
if (enemy is FinalBossData) {
    // Bullet <-> FinalBoss interaction
    if (enemy.position.distanceTo(least.position) < enemy.size) {
        gameObjects.remove(least)
        vidasFinalBoss--
        if (vidasFinalBoss <= 0) {
            finalBossEliminado = true
            gameObjects.remove(enemy)
            winGame()
        }
    }
}
```

El contador de vidas debe reiniciarse en cada partida, por lo que lo indicamos al inicio de la función **startGame**:

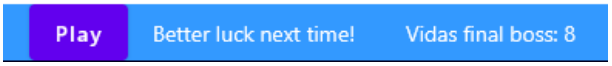
```
fun startGame() {
    finalBossEliminado = false
    vidasFinalBoss = 10
```

1.7.1 Contador de vidas

El usuario debe saber en todo momento cuántas vidas la quedan al enemigo final. Para ello, en el Main, añadimos un texto a la fila superior, junto al botón **'Play'**:

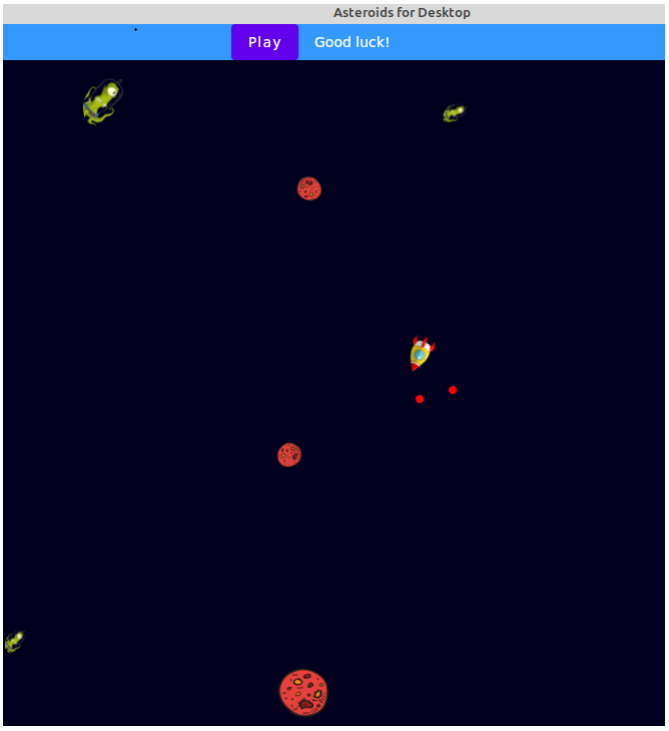
```
    Text( text: "Play")
}
Text(game.gameStatus, modifier = Modifier.align(Alignment.CenterVertically).padding(horizontal = 16.dp), color = Color.White)
if(game.vidasFinalBoss <= 10) textoVidasBoss = "Vidas final boss: ${game.vidasFinalBoss}"
else textoVidasBoss = ""
Text(textoVidasBoss, modifier = Modifier.align(Alignment.CenterVertically).padding(horizontal = 16.dp), color = Color.White)
```

Como no queremos que este texto aparezca cuando el enemigo final aún no ha aparecido, definimos la condición mostrada en la imagen. Este es el resultado:



2 Resultado final

Con todas las modificaciones mencionadas anteriormente, al ejecutar el juego esto es lo que veremos:



Y esto una vez derrotados todos los enemigos normales: