

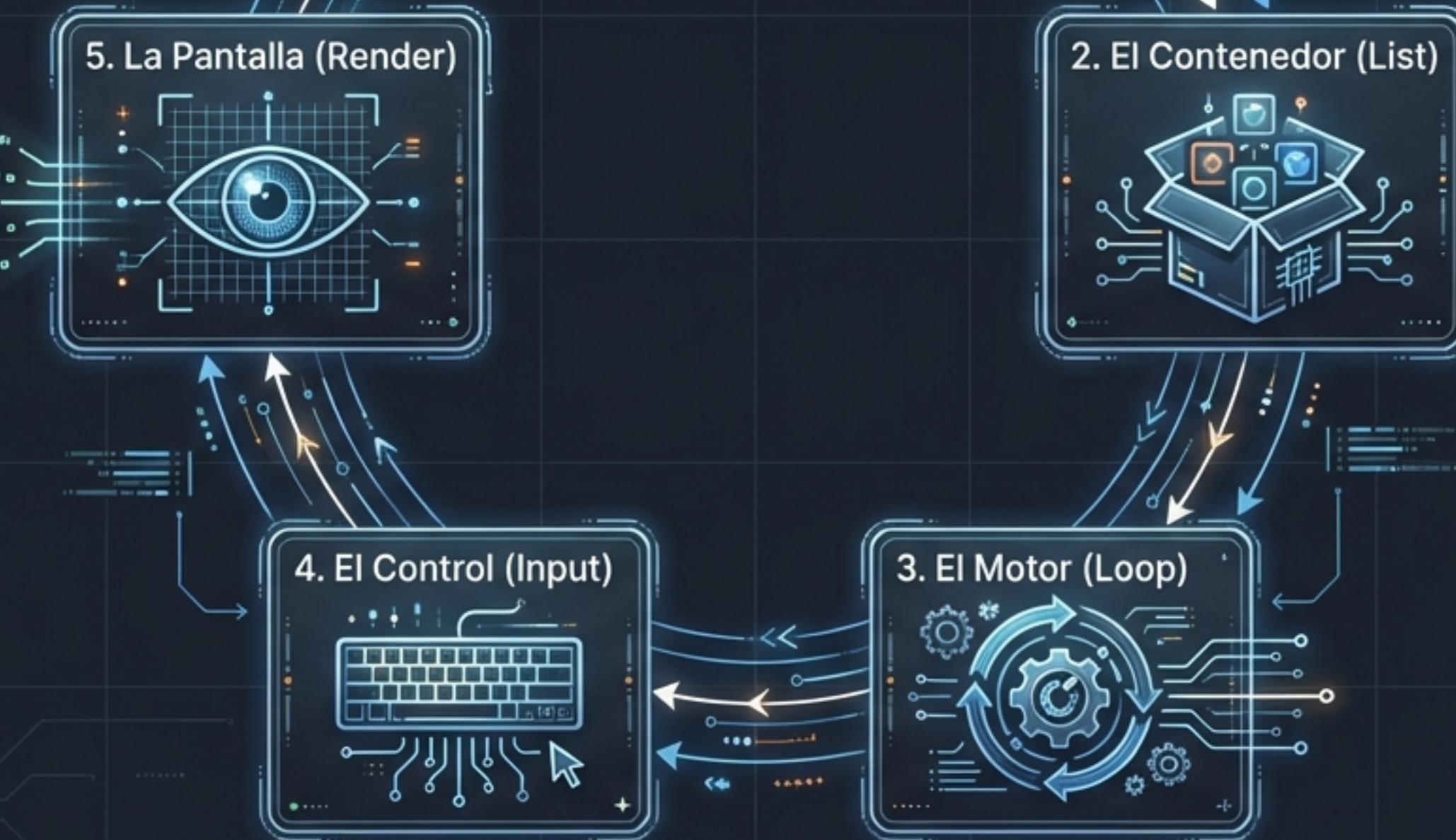
Kotlin Space Invaders: Lógica e Implementación

Desmitificando el código detrás de un motor de juego básico.

Anatomía de Sistemas · Lectura Técnica

La Visión Global: 5 Pilares del Código

Antes de escribir una sola línea, debemos entender la arquitectura. El juego no es más que una lista de objetos que se actualiza y se dibuja repetidamente dentro de un bucle infinito.

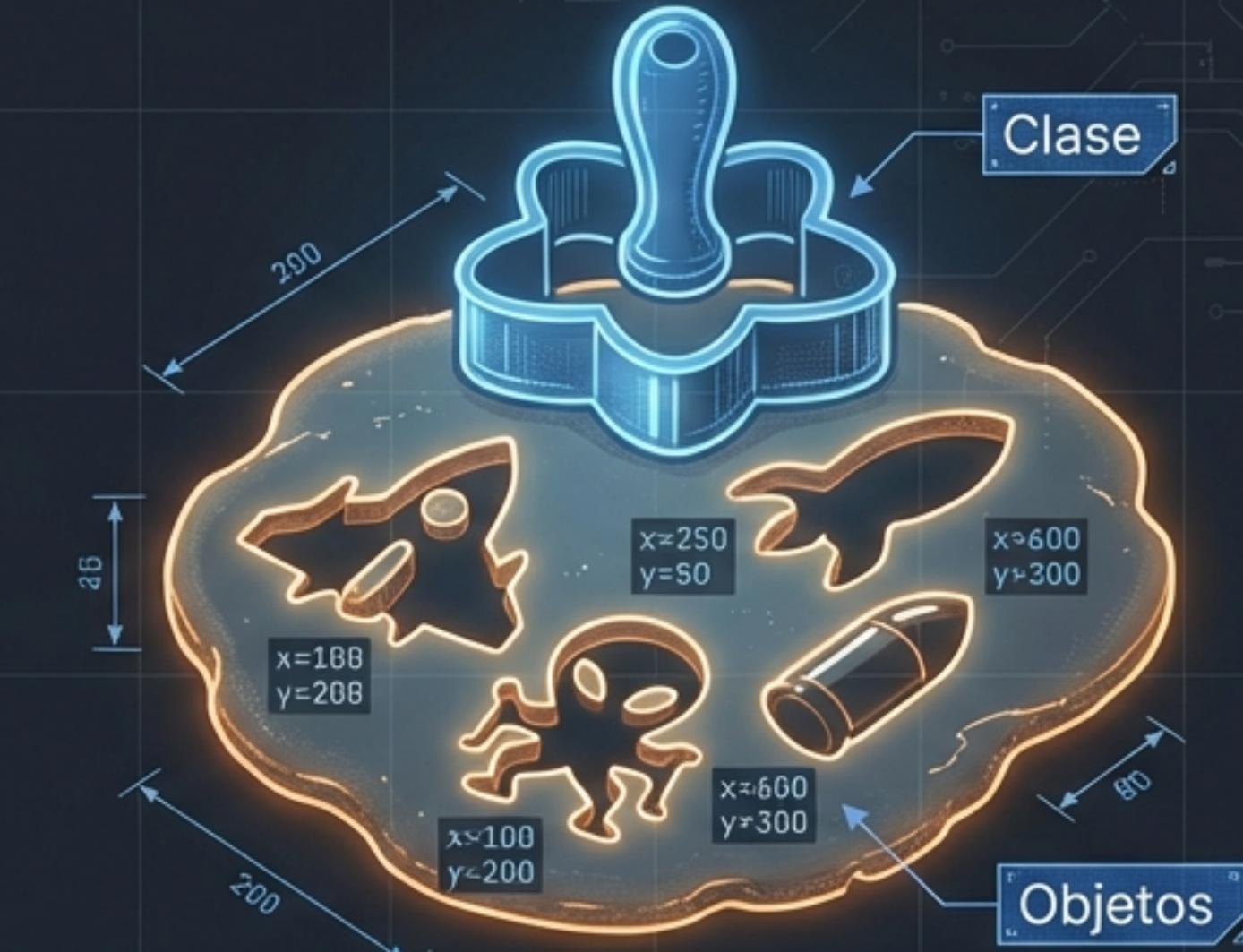


El ADN del Juego: La Clase `Entidad`

```
class Entidad(  
    var x: Int,  
    var y: Int,  
    val tipo: String  
)
```

Estado Mutable.
Cambian cuando la entidad se mueve.

Identidad Inmutable.
NAVE, ALIEN o BALA.
No cambia.

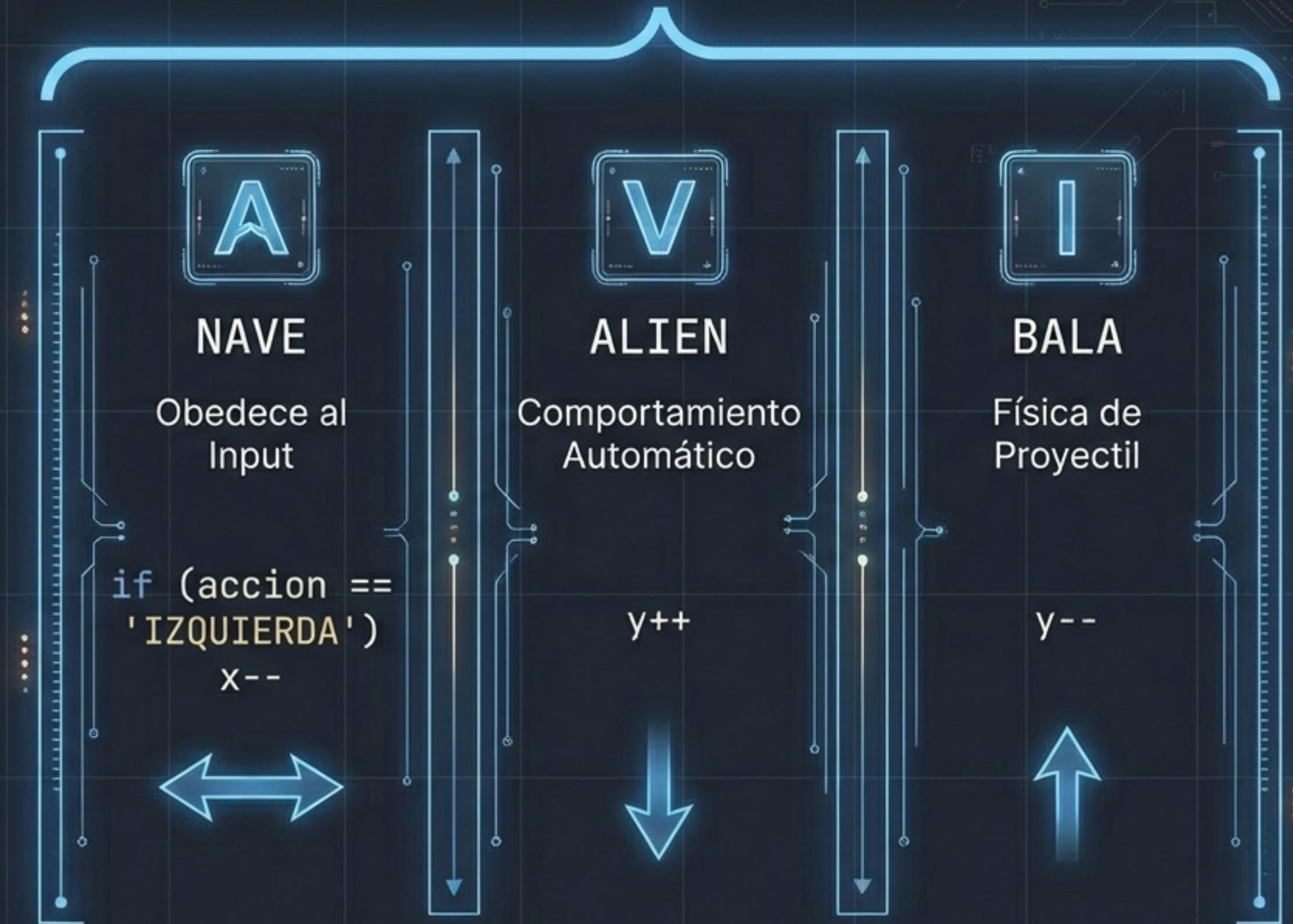


La Clase es el molde; los Objetos son las galletas. Definimos la estructura una vez, pero creamos muchas instancias con coordenadas únicas.

Polimorfismo Simplificado con `when`

Una sola función `mover(accion)` gobierna todo. Usamos `when(tipo)` para que cada entidad reaccione según su naturaleza.

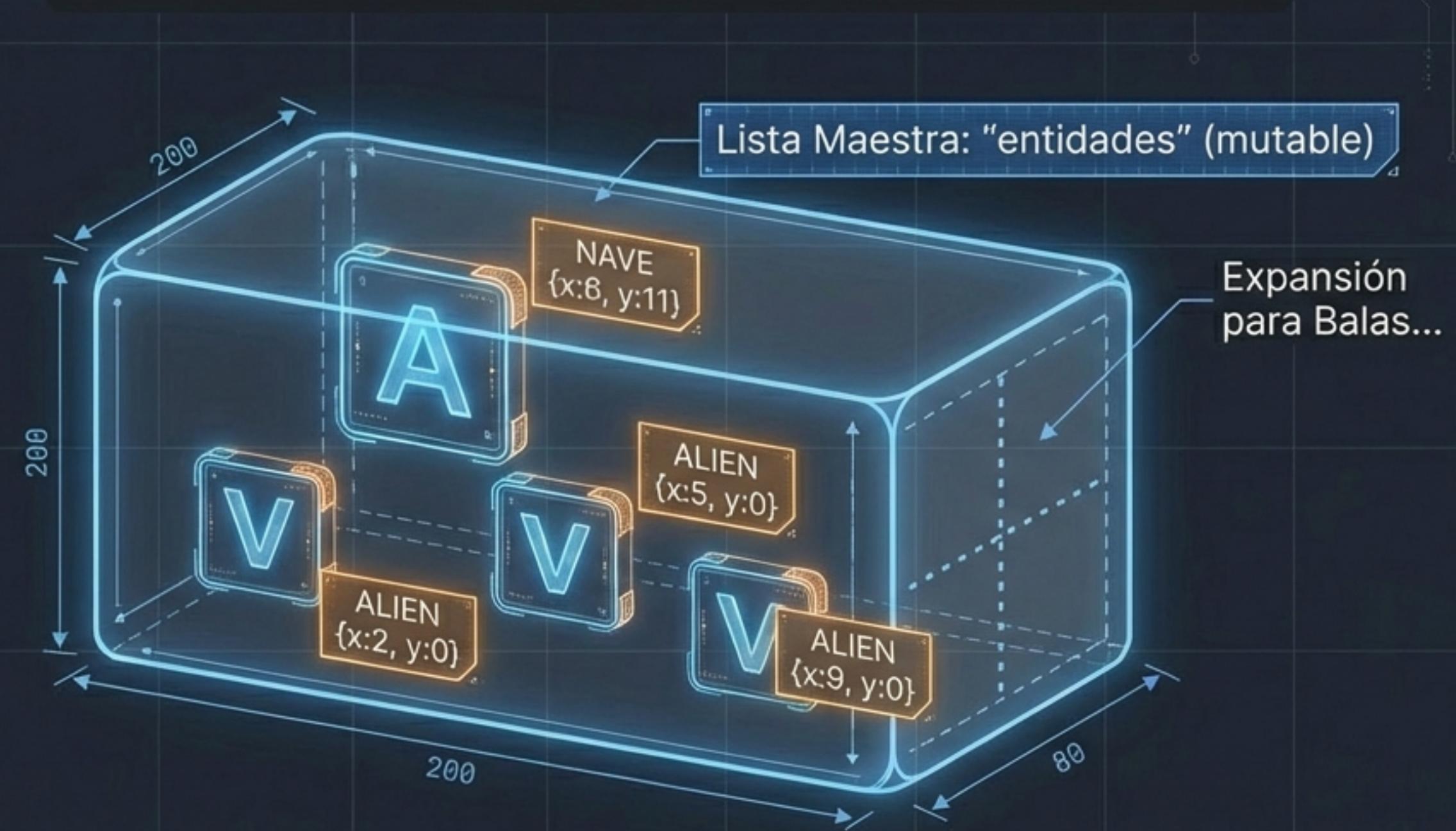
fun mover()



La Puesta en Escena: Inicialización

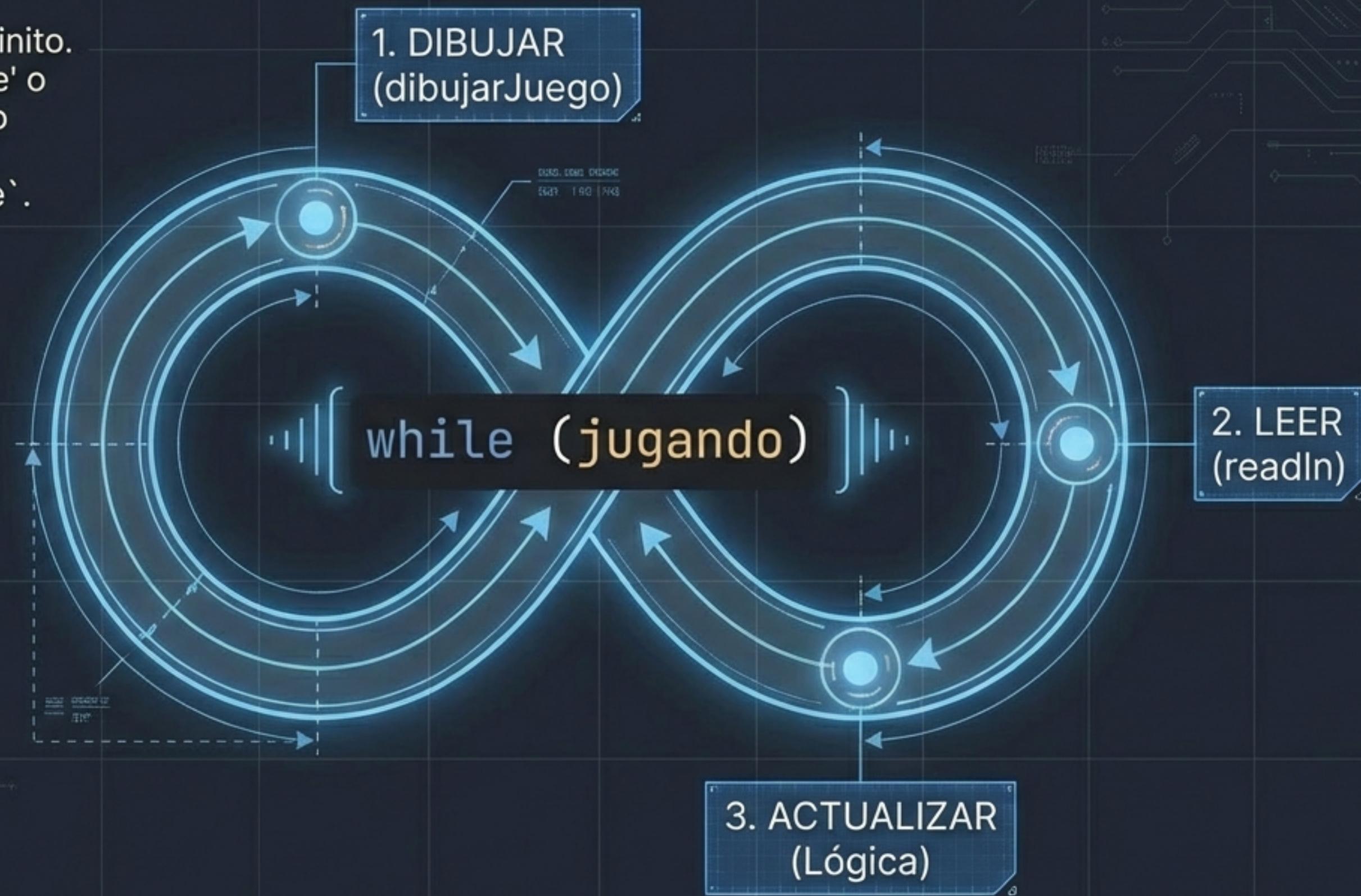
Creamos una Lista Maestra que mezcla todo. Al principio, solo contiene la Nave y 3 Enemigos. Esta lista es dinámica: crecerá con las balas y se encogerá con las muertes.

```
val entidades = mutableListOf<Entidad>()  
  
entidades.add(nave)  
// + 3 Aliens aleatorios
```

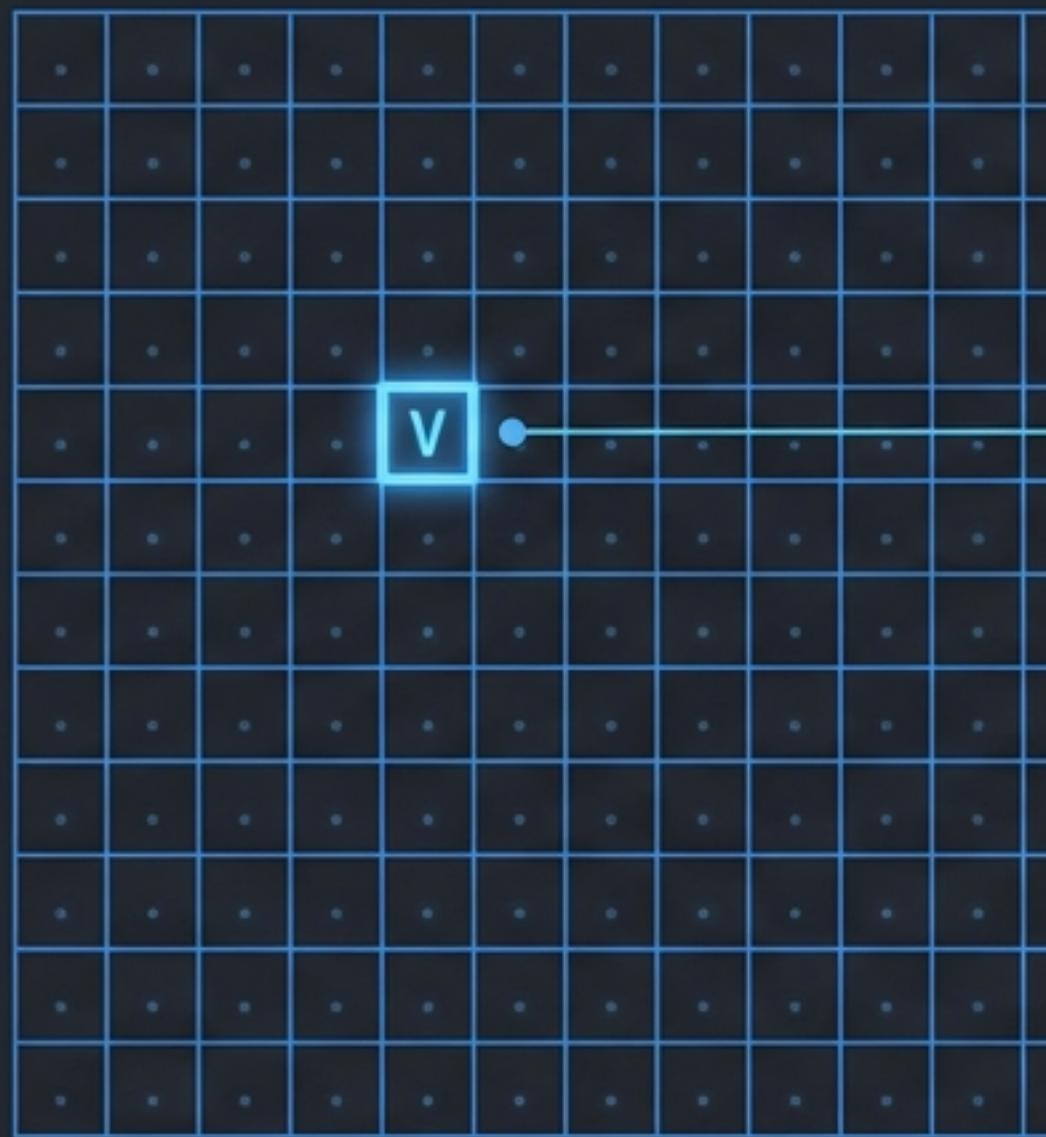


El Corazón del Juego: El Bucle 'While'

El juego vive dentro de un ciclo infinito. Cada vuelta del bucle es un 'Frame' o turno. El programa queda atrapado aquí voluntariamente hasta que la variable 'jugando' cambia a 'false'.

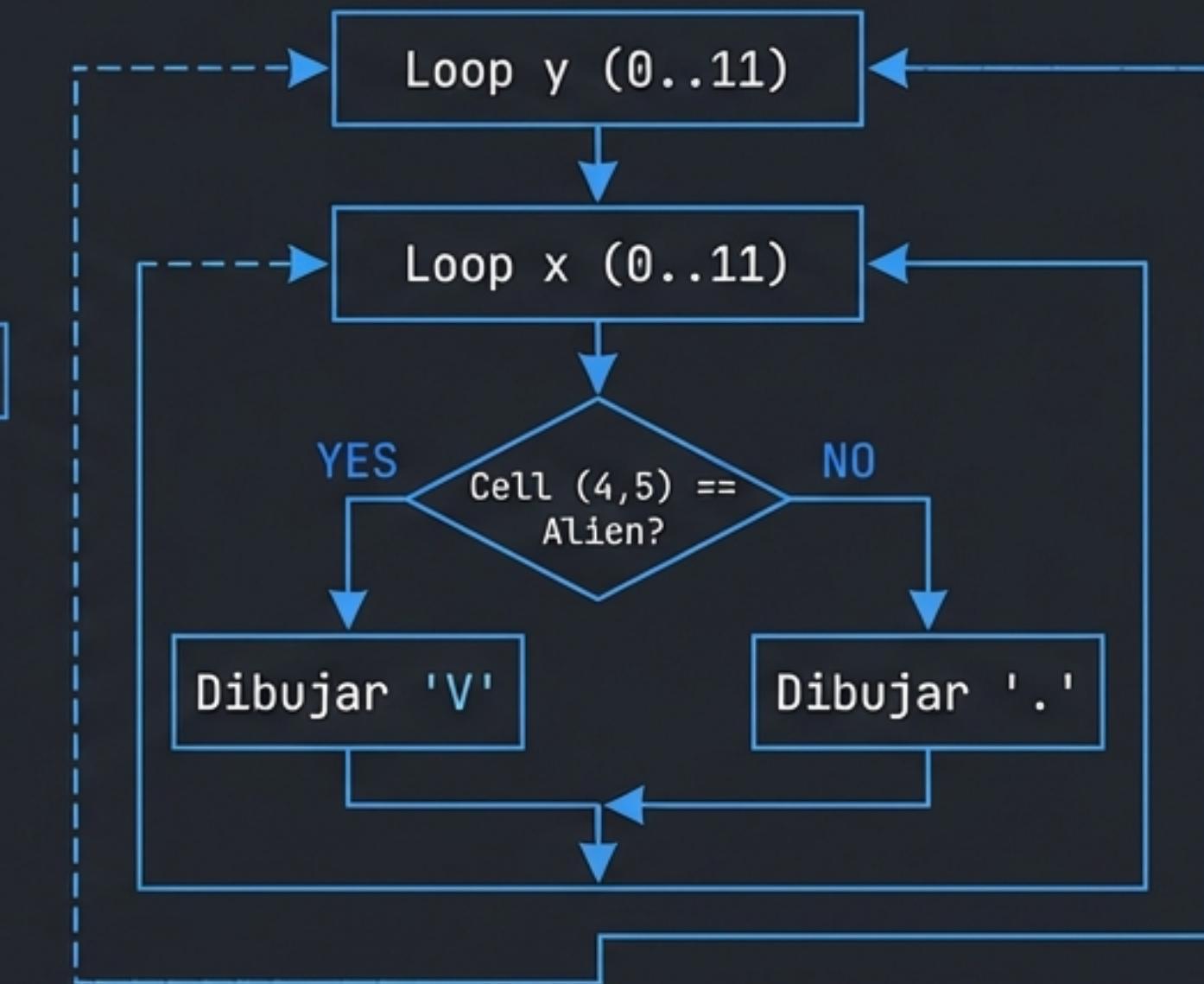


Renderizado: Traduciendo Datos a Píxeles



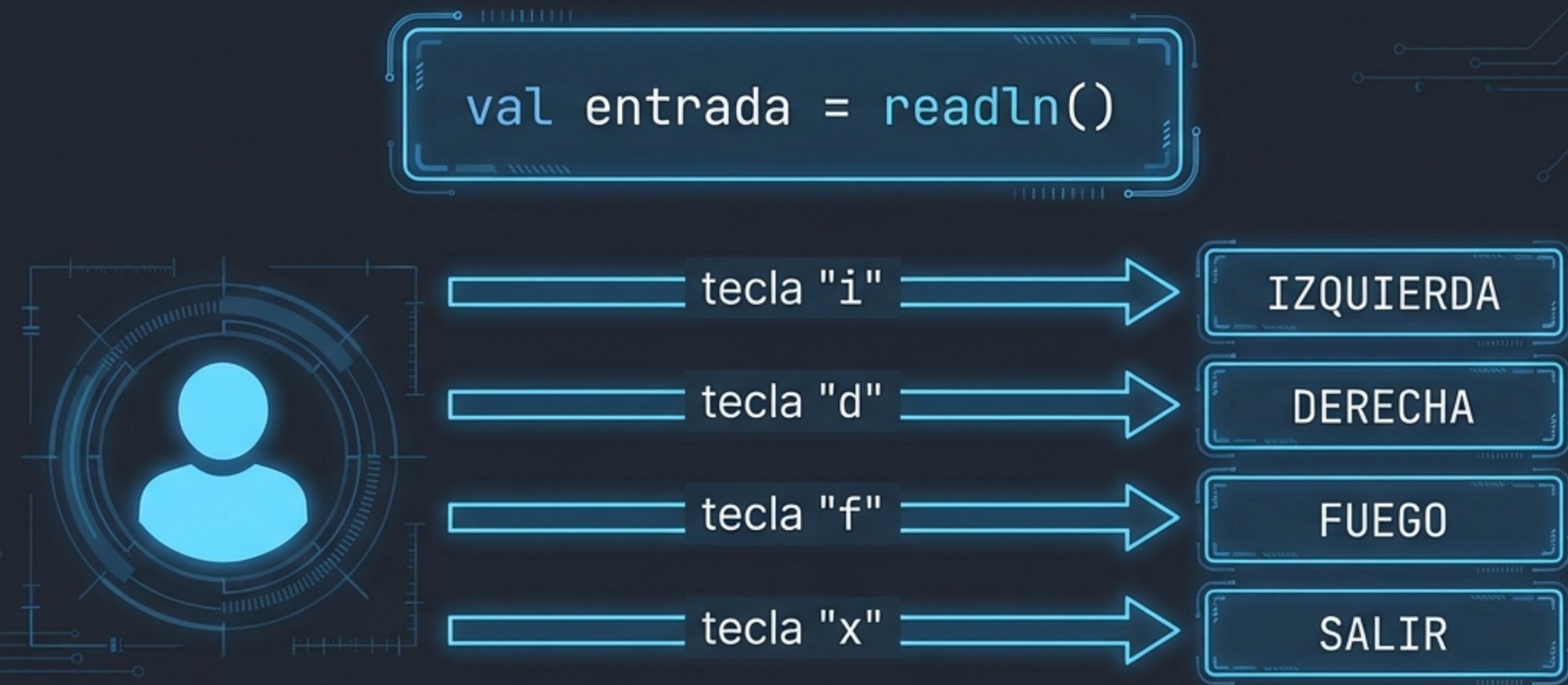
ALIEN (x:5, y:4)

Punto de datos a píxel



Dos bucles anidados escanean cada celda. Si `lista.find` encuentra una entidad, dibuja su ícono. Si no, dibuja espacio vacío.

El Control: Escuchando al Jugador



`readln()` pausa la ejecución del universo. Nada se mueve hasta que el jugador presiona Enter. Convertimos teclas cortas en acciones semánticas legibles.

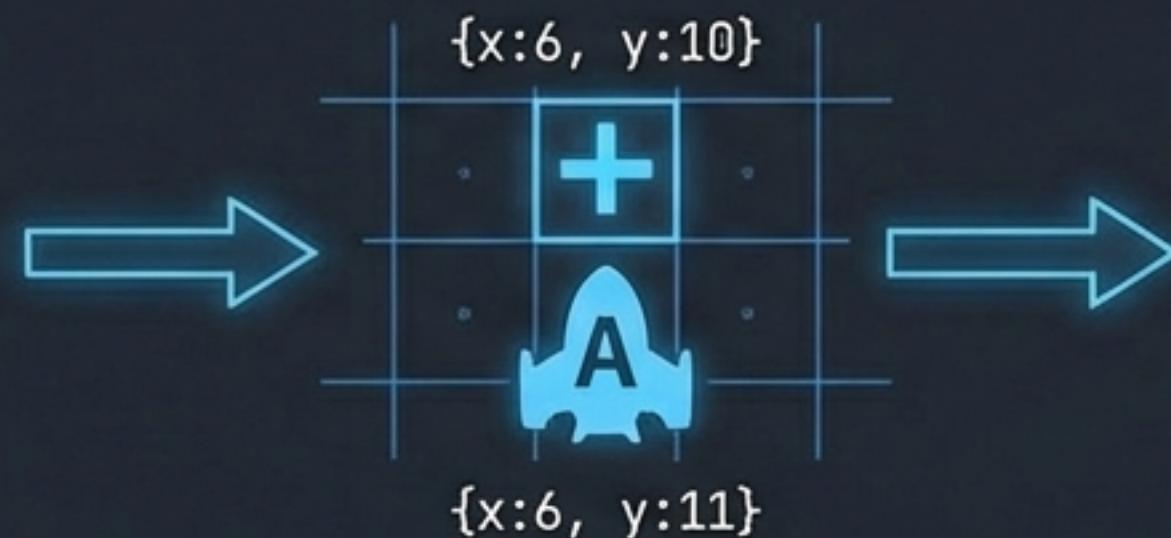
Gestión del Caos: Creación Dinámica

La lista no es estática. Cada disparo nace como una nueva instancia. Si disparas 10 veces, la lista crece en 10 objetos.

Input

```
if (accion == "FUEGO")
```

Genesis



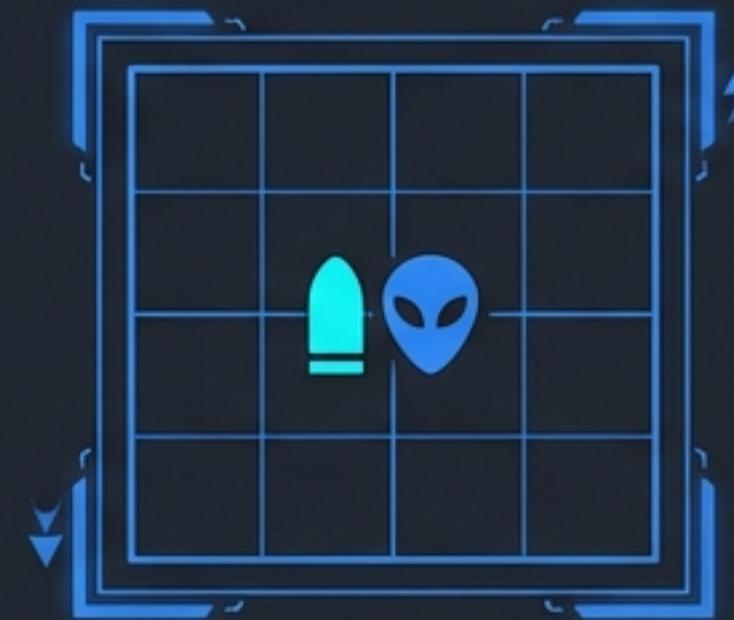
Resultado

```
[  
    NAVE {x:6, y:11, icon:'A'},  
    BALA {x:6, y:10}  
]
```

Matemáticas de la Muerte: Colisiones

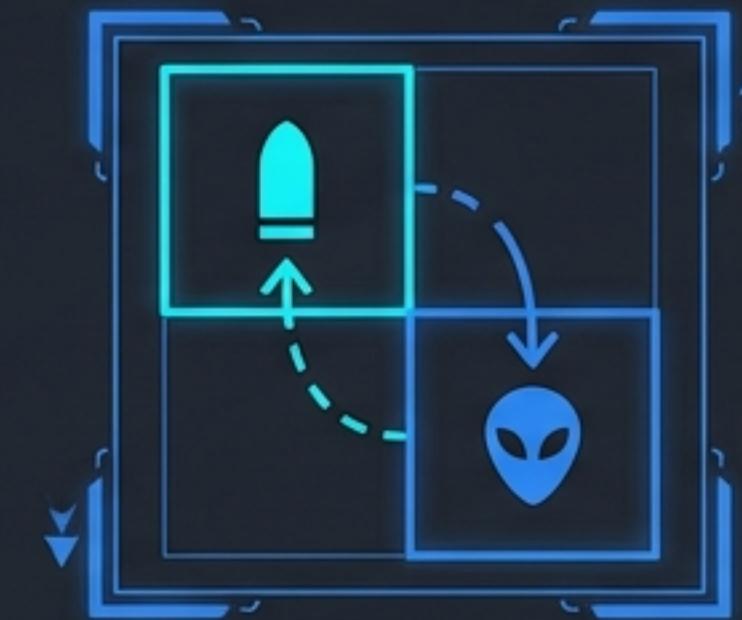
No basta con estar en la misma celda. Debemos detectar el 'Cruce en el aire' cuando un alien baja y una bala sube en el mismo turno.

Choque Directo



entidad.y == otra.y

Cruce en el Aire

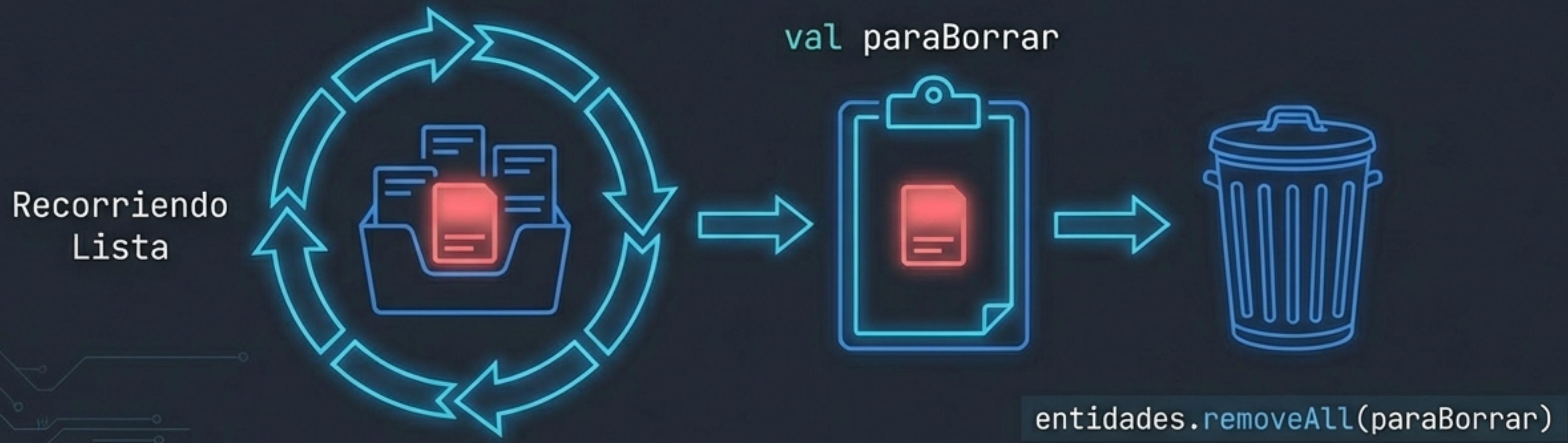


entidad.y == otra.y + 1

```
val mismaColumna = (entidad.x == otra.x)
if (mismaColumna && (choqueDirecto || cruceEnElAire))
```

El Problema de la Modificación Concurrente

“No puedes quitar el piso mientras caminas sobre él.”



Solución: Usamos una lista temporal. Marcamos las víctimas durante la lógica, y las borramos de forma segura al final del turno.

Condiciones de Victoria y Derrota

El bucle solo se rompe cuando se cumple una condición terminal.



`if (alien.y >= alto - 1)`



`return
(Programa termina)`



`if (aliensRestantes == 0)`



`jugando = false
(Bucle termina)`

Resumen de Conceptos Kotlin



→ **Class/Objects:** El plano vs la construcción.



→ **MutableList:** Una colección dinámica que crece y decrece.



→ **When:** Alternativa limpia a múltiples if/else.



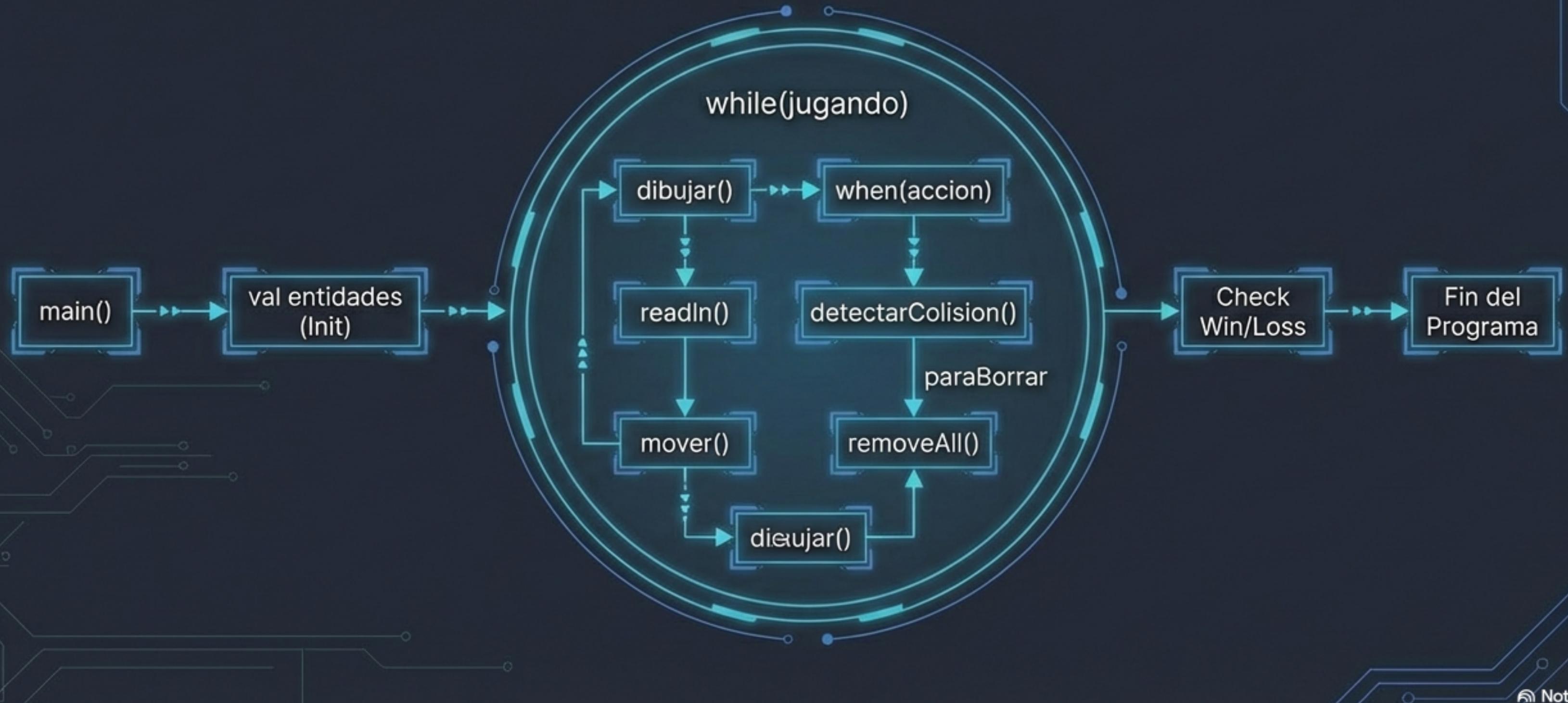
→ **Loops (While/For):** El motor de repetición y escaneo.



→ **Null Safety:** Uso de ? y verificaciones null en el renderizado.

Anatomía Completa del Sistema

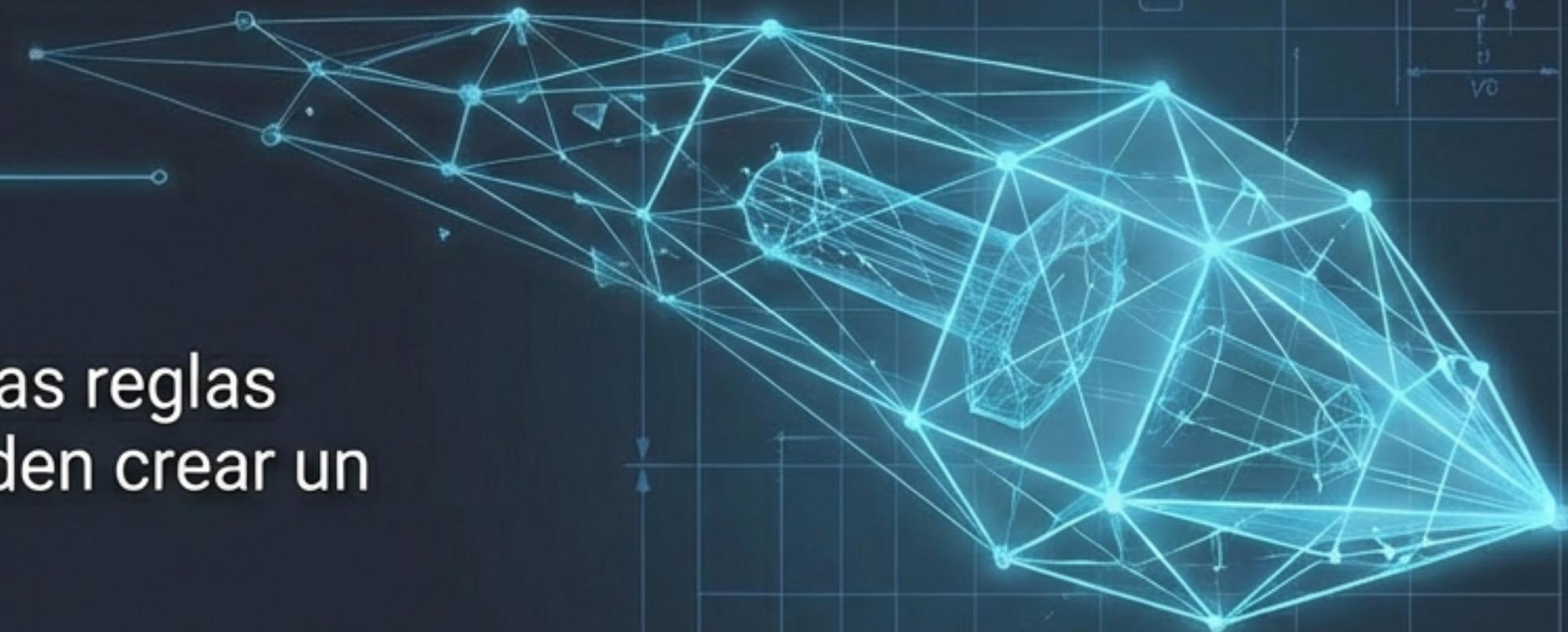
Todo el código conectado en un solo flujo lógico.



De la Lógica Simple a Mundos Complejos

A

- |



Has visto cómo unas pocas reglas matemáticas y listas pueden crear un universo interactivo.

¿Qué sigue? Añade escudos, velocidades variables o una nave nodriza. El código es tuyo para modificarlo.