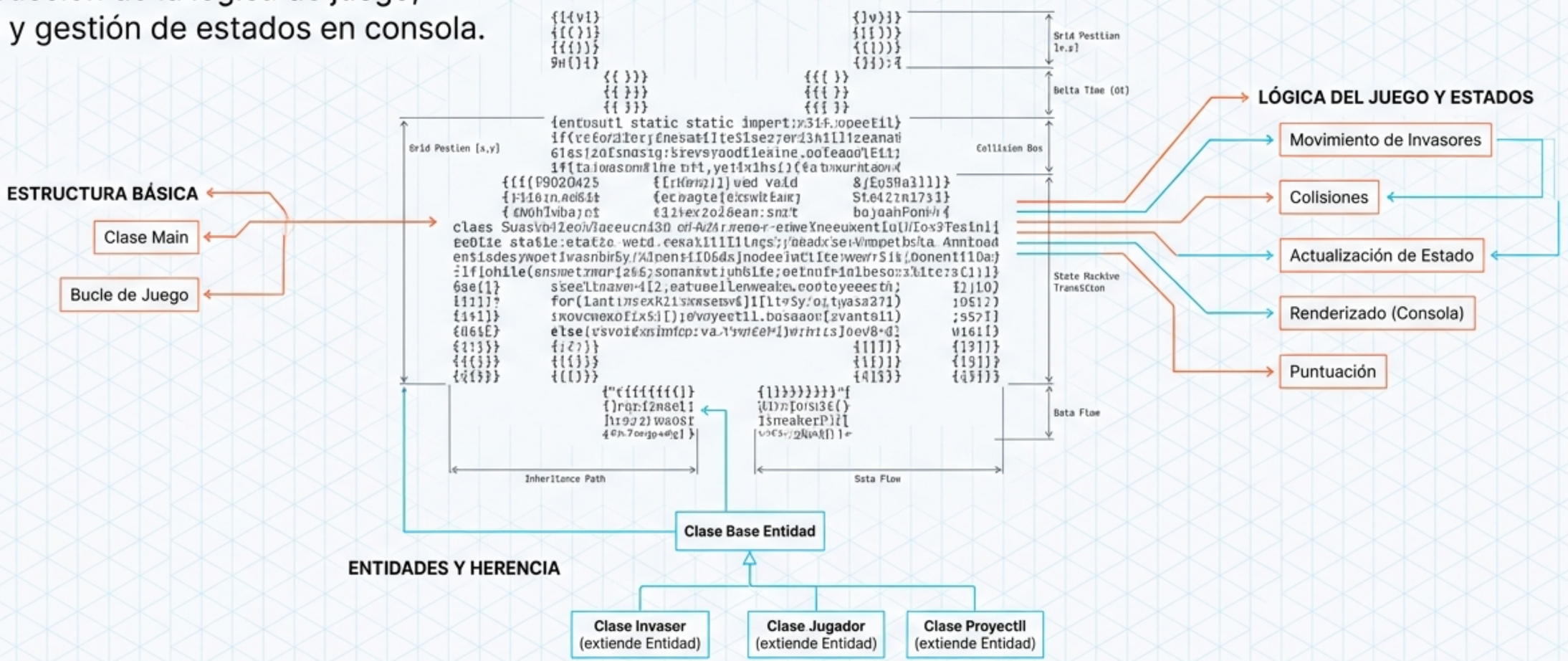
Java Space Invaders: Arquitectura de un Clásico

Deconstrucción de la lógica de juego, herencia y gestión de estados en consola.



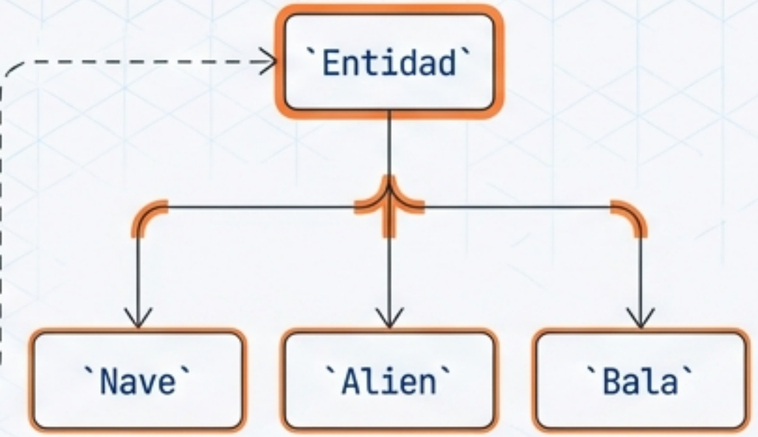
v1.0 // ANÁLISIS TÉCNICO // JAVA PURO

Visión General del Sistema

ENTRADA (INPUT)

Separación de responsabilidades:
Datos vs. Ejecución.

ENTIDADES (DATOS)



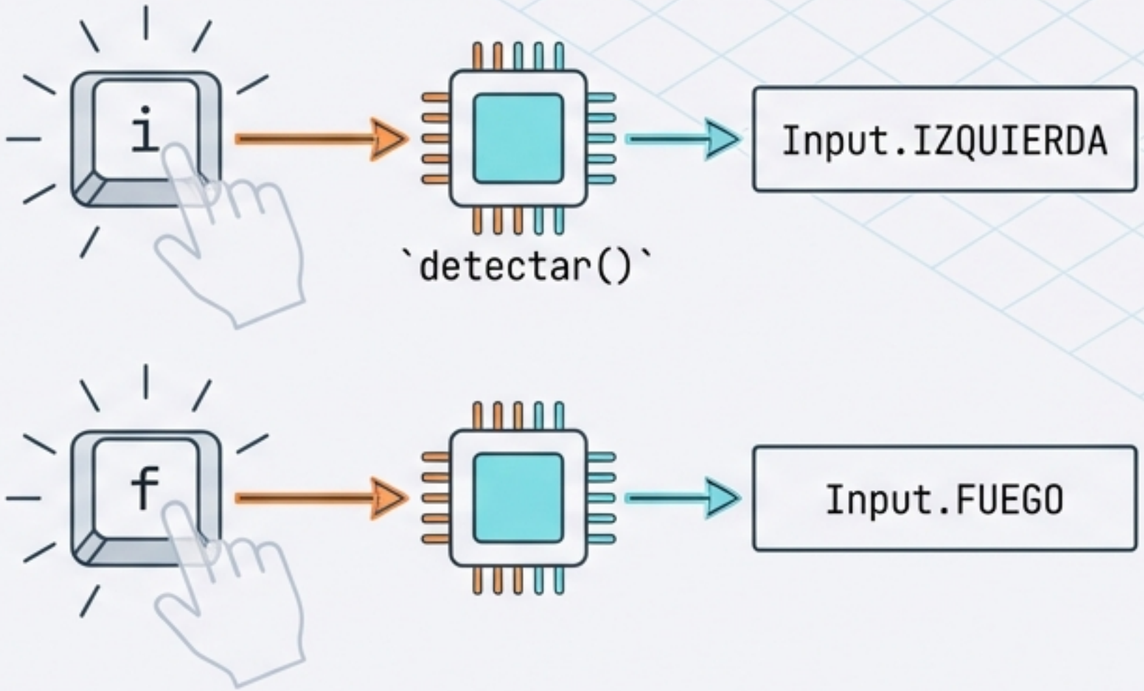
Estructura jerárquica de datos.

MOTOR (LÓGICA)

Bucle Principal.

La Interfaz de Control

```
enum Input {
  IZQUIERDA("i", "Izquierda"),
  DERECHA("d", "Derecha"),
  FUEGO("f", "FUEGO"),
  SALIR("x", "Salir");
  ...
  public static Input detectar(String texto) {
    ...
  }
}
```



- **Tipado Fuerte:** El uso de `enum` evita 'números mágicos' o cadenas de texto dispersas por el código.
- **Mapeo Directo:** El método `detectar(String texto)` traduce caracteres ASCII simples a intenciones de juego claras y legibles.

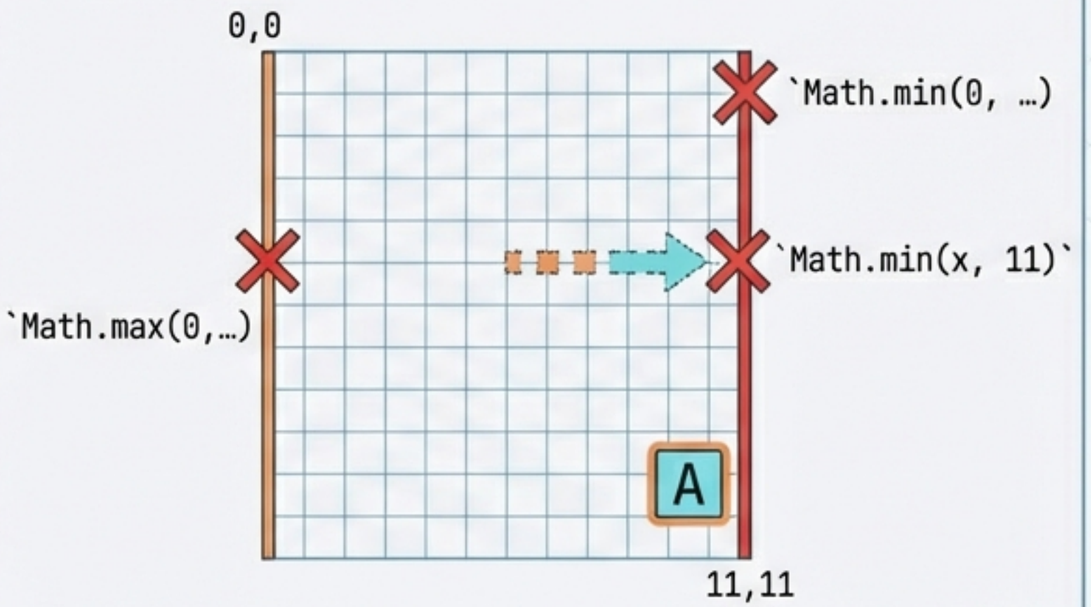
La Clase Madre: `Entidad`

CÓDIGO (JAVA)

```
class Entidad {
  protected int x, y;
  protected char icono;
  ...
  public void mover(Input accion) {
    x = Math.max(0, Math.min(x, ANCHO - 1));
    y = Math.max(-1, Math.min(y, ALTO));
  }
}
```

- **Encapsulamiento:** Las coordenadas `x` e `y` y el `icono` son estado protegido, accesible solo por la jerarquía de herencia.

DIAGRAMA DE LÍMITES



- **Límites del Mundo:** La lógica de `mover()` impone las reglas físicas del tablero. Ninguna entidad puede existir fuera de las coordenadas `[0, ANCHO-1]`.

Herencia y Polimorfismo

NAVE (Control Manual)

```
@Override
public void mover(Input accion) {
  if (accion == Input.IZQUIERDA) x--;
  if (accion == Input.DERECHA) x++;
  super.mover(accion);
}
```

e.mover()

e.mover()

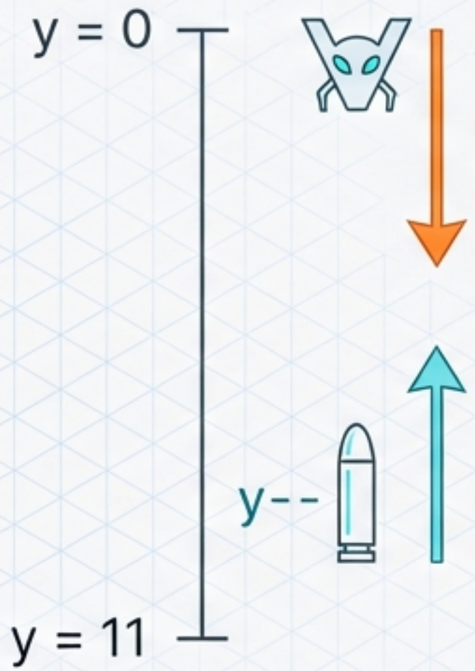
ALIEN (Autómata)

```
@Override
public void mover(Input accion) {
  y++; // Siempre baja
  super.mover(accion);
}
```

- Responde a la intención del usuario.
- Comportamiento determinista: desciende una fila en cada ciclo.

El motor invoca `e.mover()` de forma polimórfica, ignorando el tipo específico de objeto.

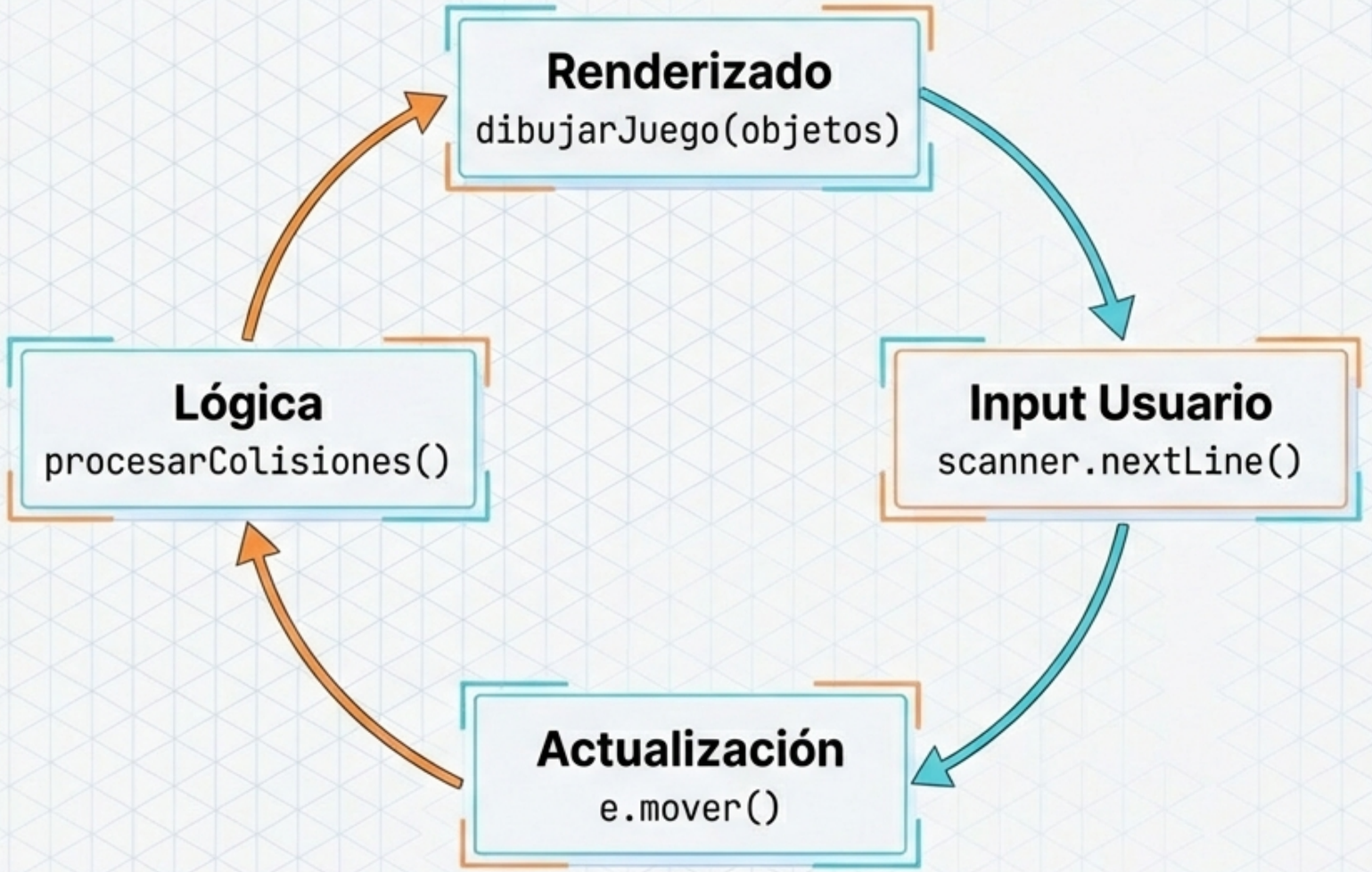
Balística Simple: `class Bala`




```
@Override
public void mover(Input accion) {
    y--; // La bala siempre sube
    super.mover(accion);
}
```

- **Comportamiento Inverso**: Mientras los enemigos descienden, la munición asciende.
- **Efimeridad**: Las balas están diseñadas para salir del tablero (`y < 0`), momento en el cual el recolector de basura manual las eliminará.

El Bucle Principal (The Game Loop)



Secuencialidad

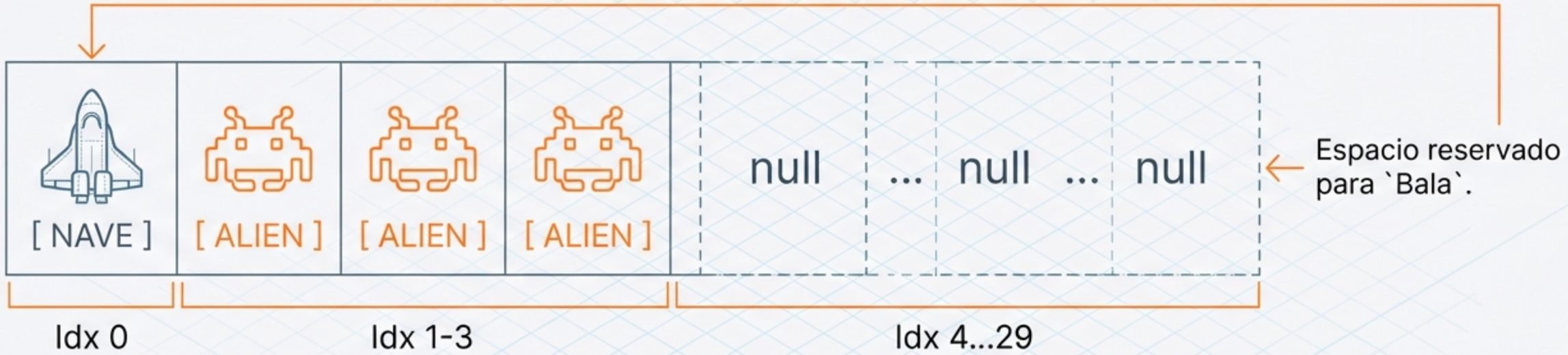
En consola, el juego es por turnos. El bucle espera el ‘Intro’ del usuario.

Gestión de Trama

Cada iteración del `while` representa un ‘frame’ de animación.

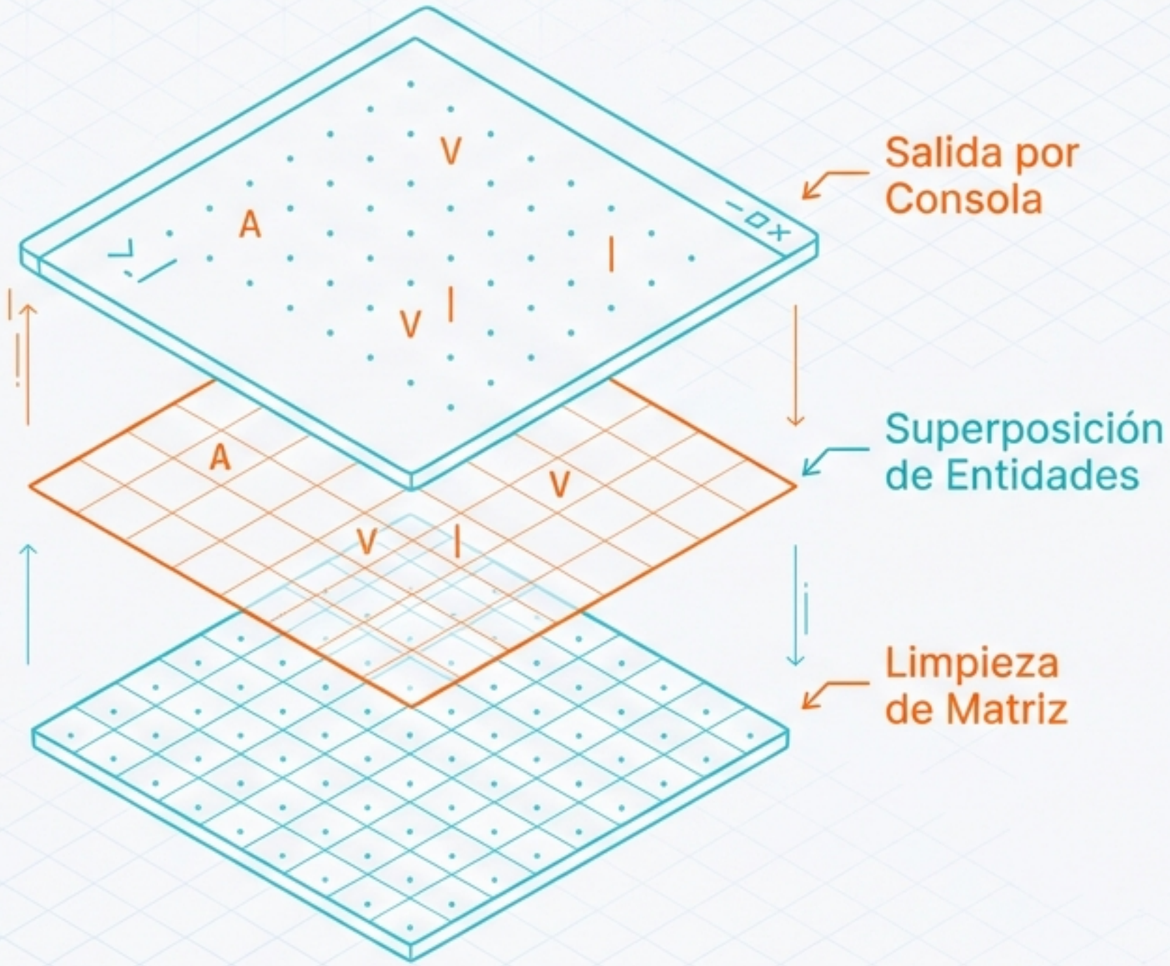
Gestión de Memoria Estática

`MAX_ENTIDADES = 30`. Sin `ArrayList` ni estructuras dinámicas.



- **Lógica de Disparo:** Al pulsar FUEGO, el motor itera buscando el primer `null`. Si el array está lleno, el arma se encasquilla.

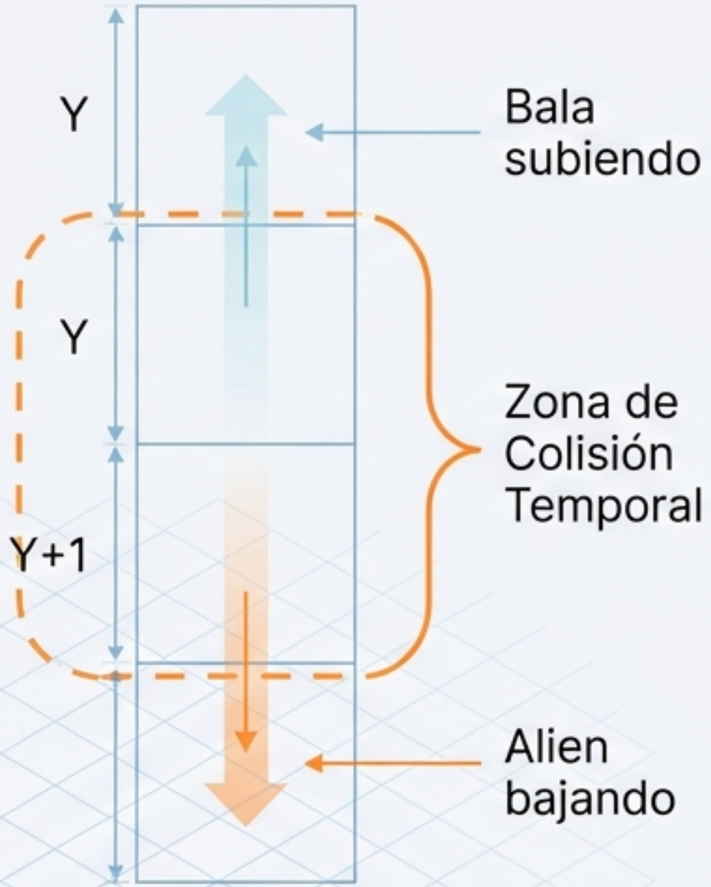
El Sistema de Renderizado




```
// Limpiar
matriz[y][x] = '.';
// Pintar Entidades
matriz[e.y][e.x] = e.icono;
// Imprimir
System.out.print(matriz[y][x] + " ");
```

En cada frame, se reconstruye la escena desde cero.

Detección de Impactos y Hitboxes

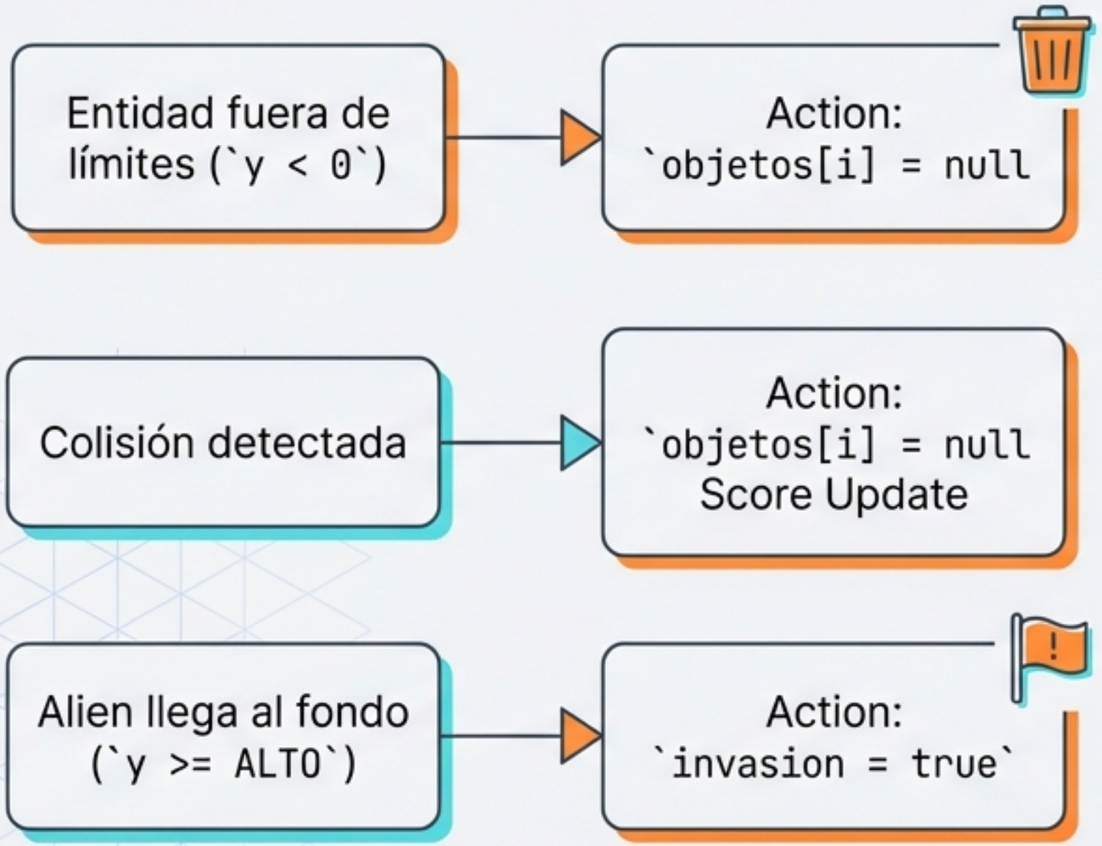



```
if (posibleBala.x == alien.x &&
    (posibleBala.y == alien.y ||
     posibleBala.y == alien.y - 1))
}
```

El Problema del Túnel: Como la bala sube y el alien baja en el mismo turno, podrían cruzarse sin compartir coordenada exacta.

Solución: La comprobación extendida (`y-1`) detecta el cruce en el tiempo.

Gestión de Estado y Limpieza



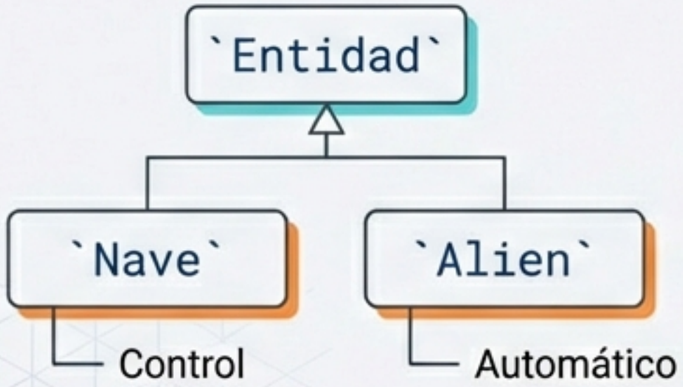

```
if (impactado) {
    objetos[i] = null; // Liberar memoria
    System.out.println("¡ALIEN DESTRUIDO!");
}
```

Derrota: Invasión detectada en la última fila.

Victoria: Contador `aliensVivos` llega a 0.

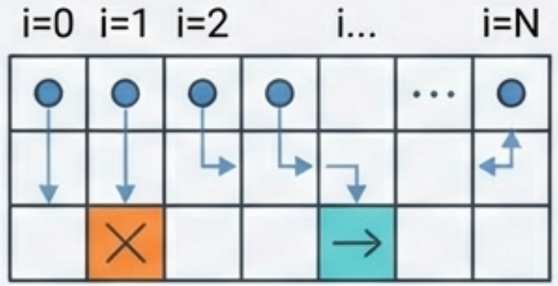
Resumen Técnico

Arquitectura POO



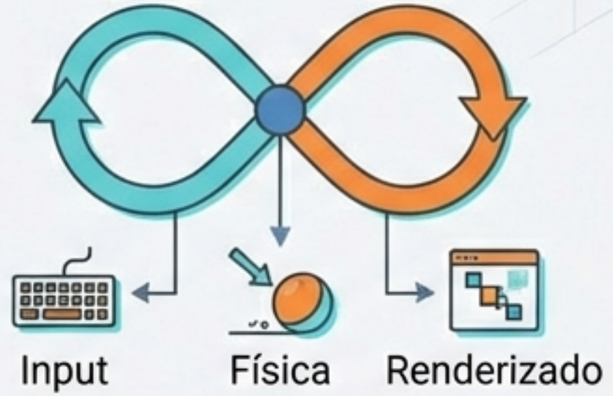
`Entidad` como contrato base. Polimorfismo para separar lógica de control (Nave) de lógica automática (Alien).

Estado Estático



Gestión manual de memoria con Array fijo. CRUD básico sobre índices para disparos y muertes.

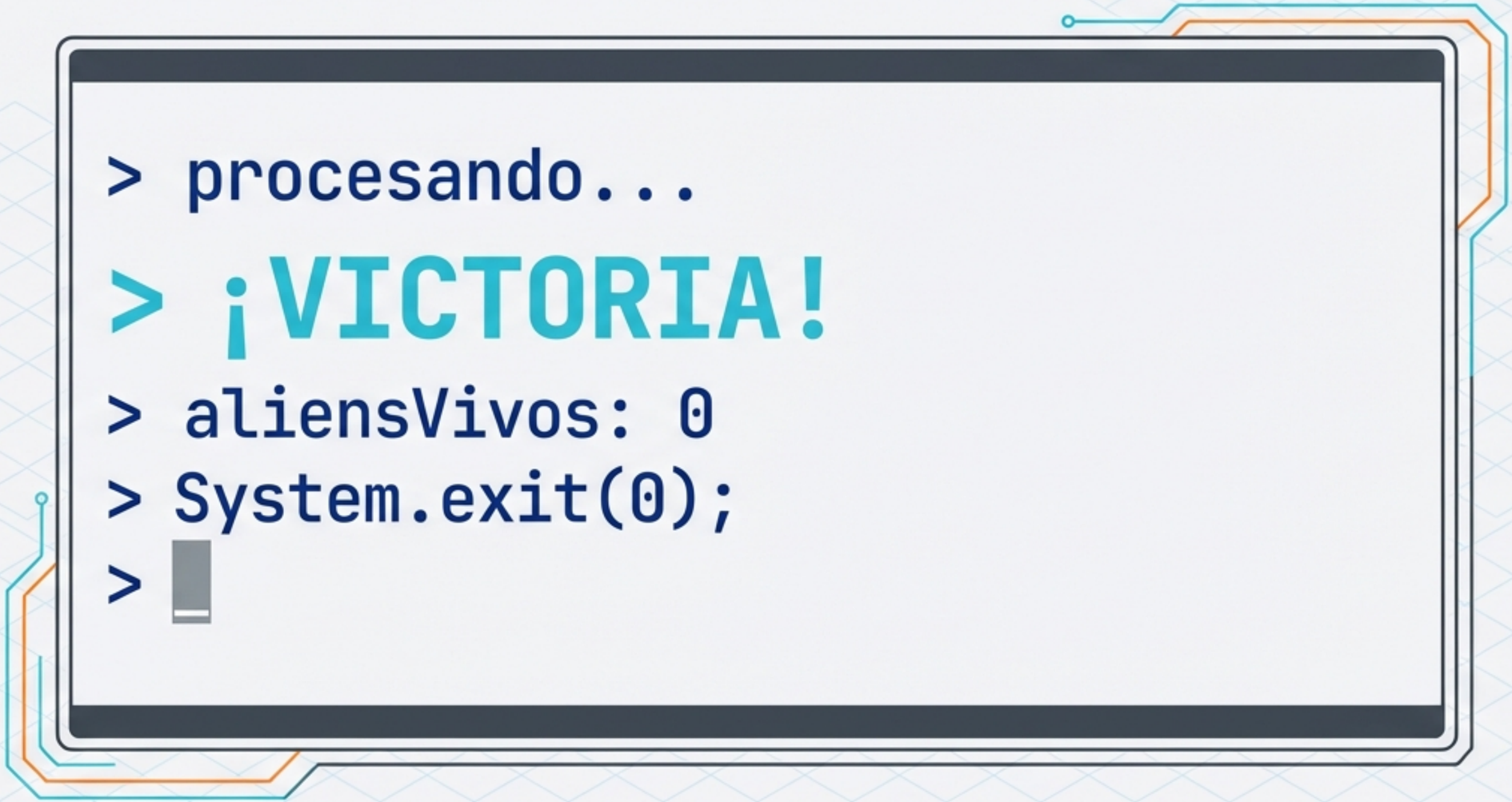
El Bucle



Coordinación centralizada de Input, Física y Renderizado en un solo hilo de ejecución.

Complejidad emergente a partir de estructuras simples.

Resultado de Juego: Victoria



Diseño basado en la implementación Java de Space Invaders.