Guía Técnica

José Alfredo Quevedo Suarez, Santiago Herrera Parra, Antonio Garay Pinzón

1. Estructuras de Datos Principales

El juego utiliza varias estructuras de datos, principalmente **objetos**, **arrays** y algunos **objetos literales** para almacenar configuraciones y recursos.

1.1. Objetos Literales

- cameraOffset, panStart: { x, y } → representan coordenadas en el lienzo para el movimiento de cámara.
- recursos: { hojas, agua } → guarda la cantidad de recursos disponibles.
- CONFIG_OLEADAS: Array de números → define la cantidad de enemigos por oleada.
- juego: { camaras: [], hormigas: [], depredadores: [] } → objeto principal que contiene los arrays de entidades del juego.

Uso principal: Agrupar variables relacionadas en un solo objeto para facilitar acceso y manipulación.

1.2. Arrays

- juego.camaras: Array de instancias de la clase Camara.
- juego.hormigas: Array de instancias de la clase Hormiga.
- juego.depredadores: Array de instancias de la clase Depredador.
- animacionesPiedra: Array de animaciones de piedras (vacío en este fragmento, pero preparado para agregar animaciones).
- SPRITE_*: Matrices (Array de Array) de 0 y 1 para representar pixel art.

Notas sobre su uso:

- Los arrays se recorren usando forEach o filter para actualizar el estado de cada entidad y para dibujarlas.
- Las **conexiones entre cámaras** también se implementan como arrays:
- this.conexiones = [];

Cada elemento apunta a otra instancia de Camara. Esto crea un **grafo no dirigido**, donde cada nodo conoce sus vecinos.

1.3. Clases y Objetos

El código define tres clases principales, cada una con su propio estado interno:

1. **Camara**
    - Representa una sala del hormiguero (reina, pasillo, comida o agua).
    - **Propiedades clave**:
        - x, y: coordenadas en el mundo.
        - tipo: determina el tipo de cámara y su comportamiento.
        - conexiones: Array de nodos vecinos (Camara), formando un grafo.
        - radio, vida, vidaMax: propiedades de tamaño y salud.
2. **Hormiga**
    - Representa hormigas obreras, luchadoras o larvas.
    - **Propiedades clave**:
        - tipo, tipoFuturo: control de comportamiento y evolución de larvas.
        - nodoActual, destino: referencias a instancias de Camara.
        - x, y, angulo: posición y orientación.
        - carga: recurso que transporta ('hoja' o 'agua').
3. **Depredador**
    - Representa enemigos como arañas.
    - **Propiedades clave**:
        - x, y, vida, vidaMax, velocidad, daño, radio.
        - animFrame: controla animación de movimiento.

Observaciones de estructuras:

- Cada clase utiliza **referencias a otros objetos** (Hormiga a Camara, Depredador a Hormiga o Camara).
- Esto crea una **red de relaciones** entre entidades, útil para pathfinding simple y ataques.

1.4. Matrices para Pixel Art

Los sprites se representan como **arrays bidimensionales de 0 y 1**:

```
const SPRITE_HORMIGA = [
 [0,1,0,1,0],
 [0,0,1,0,0],
 [1,0,1,0,1],
 ...
];
```

- **Filas** → eje Y.
- **Columnas** → eje X.
- 1 → dibujar pixel.
- 0 → transparencia.
- Función dibujarSprite recorre la matriz y dibuja un fillRect por cada 1.

Beneficio: permite definir sprites pixelados fácilmente y rotarlos o escalar con el canvas.

2. Interacciones de Datos

1. **Hormiga ↔ Cámara**
    - Cada Hormiga tiene nodoActual y destino que apuntan a Camara.
    - Movimiento: se calcula distancia y ángulo entre Hormiga y destino.
2. **Depredador ↔ Hormiga / Reina**
    - Cada Depredador busca instancias de Hormiga de tipo luchadora y la reina en juego.camaras.
    - Ataque: se basa en distancia y cooldownAtaque.
3. **Conexiones entre cámaras (grafo)**
    - Camara.conexiones mantiene referencias a otras cámaras.
    - Se utiliza para mover hormigas aleatoriamente entre nodos conectados.

3. Sistemas de Juego

- **Ciclo de juego principal (loop)**:
    1. Limpiar canvas.
    2. Aplicar transformaciones de cámara.
    3. Dibujar túneles (Camara.dibujarTuneles()).
    4. Dibujar cámaras (Camara.dibujarNodo()).
    5. Actualizar y dibujar hormigas.
    6. Actualizar y dibujar depredadores.
    7. Revisar estado de la reina.
    8. Actualizar UI con estadísticas y recursos.
    9. Llamar requestAnimationFrame(loop) para siguiente frame.
- **Eventos de entrada** (mousedown, mousemove, wheel) manipulan:
    - Cámara (cameraOffset, cameraZoom).
    - Selección o movimiento de nodos (draggedNode).
    - Crear/borrar/conectar cámaras (herramientaActual).
- **Sistema de oleadas**:
    - CONFIG_OLEADAS define la cantidad de enemigos por oleada.
    - tiempoParaHorda cuenta regresivamente y dispara lanzarHorda().
    - Los depredadores se almacenan en juego.depredadores.
- **Guardado de partida**:
    - localStorage guarda cámaras (solo IDs de conexiones), hormigas, depredadores y recursos.

4. Resumen de Estructuras de Datos Clave

Estructura	Tipo	Contenido	Uso principal
juego.camaras	Array	Camara	Grafo de nodos (hormiguero)
Camara.conexiones	Array	Camara	Conexiones entre salas
juego.hormigas	Array	Hormiga	Entidades móviles, transportan recursos o luchan
juego.depredadores	Array	Depredador	Enemigos que atacan hormigas y reina
SPRITE_*	Array 2D	0/1	Pixel art para dibujar sprites
recursos	Objeto	hojas, agua	Contador de recursos del jugador
cameraOffset, panStart	Objeto	x, y	Coordenadas para cámara y paneo

5. Conclusiones

- El juego combina **arrays** para almacenar entidades y **objetos** para atributos individuales.
- Las **referencias cruzadas** (Hormiga → Camara, Depredador → Hormiga) permiten simular un mundo interactivo sin estructuras más complejas como grafos explícitos.
- Los **sprites como matrices** facilitan la animación pixelada y permiten rotaciones, escalado y efectos de “wiggle”.
- El diseño es modular: cada clase maneja su lógica y renderizado, mientras que el loop central orquesta las actualizaciones.