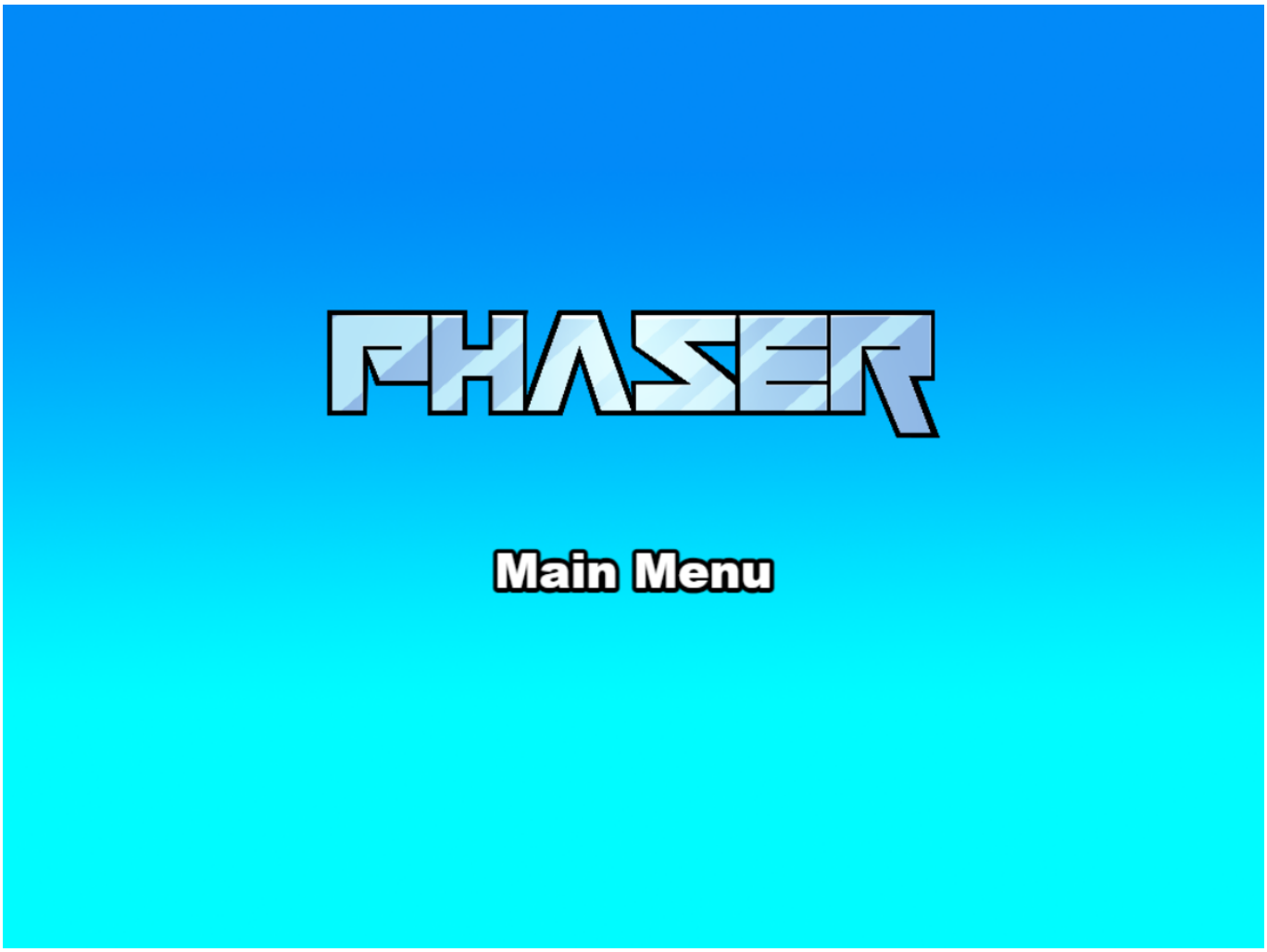
🎮 Nexa - Juego de Estrategia en Tiempo Real



Un juego de estrategia en tiempo real basado en grafos donde la gestión de energía y la planificación táctica son clave para la victoria.

Características • Instalación • Cómo Jugar • Mecánicas • Desarrollo

📖 Descripción General

Nexa es un título de estrategia en tiempo real donde los jugadores compiten por controlar la mayor cantidad de **nodos** dentro de un campo representado como un **grafo**. Cada jugador comienza con un nodo inicial desde el cual administra la distribución de **energía** hacia nodos vecinos a través de aristas conectadas.

🎯 Objetivo del Juego

- **Victoria primaria:** Controlar el **70% de los nodos** durante **10 segundos continuos**
- **Victoria alternativa:** Mantener la **mayor cantidad de nodos** al finalizar el límite de tiempo de **3 minutos**

- **Derrota automática:** Perder el nodo inicial (base central)

✨ Características

- 🕹️ **Estrategia en tiempo real** con mecánicas de gestión de recursos
- 🔋 **Sistema de energía dinámico** con ataque y defensa
- 🎨 **6 tipos de nodos especiales** con habilidades únicas
- ⚔️ **Sistema de conflictos** con resolución en tiempo real
- 🏆 **Múltiples condiciones de victoria**
- 🎮 **Interfaz intuitiva** construida con Phaser 3
- ⚡ **Rendimiento optimizado** con Vite y TypeScript

🚀 Instalación

Requisitos Previos

- **Node.js** 18.x o superior
- **pnpm** 8.x o superior (gestor de paquetes recomendado)

Pasos de Instalación

```
# 1. Clonar el repositorio
git clone https://github.com/gustadev24/nexa.git
cd nexa

# 2. Instalar dependencias
pnpm install

# 3. Iniciar servidor de desarrollo
pnpm run dev

# 4. Abrir en el navegador
# El juego estará disponible en http://localhost:8080
```

Scripts Disponibles

```
# Desarrollo (con auto-reload)
pnpm run dev

# Desarrollo sin logs
pnpm run dev-nolog

# Build de producción
pnpm run build

# Build sin logs
pnpm run build-nolog
```

🎮 Cómo Jugar

Controles Básicos

1. **Seleccionar nodo:** Click en un nodo de tu propiedad
2. **Asignar energía de ataque:** Arrastra desde tu nodo hacia un nodo vecino
3. **Ajustar defensa:** La energía no asignada permanece como defensa
4. **Capturar nodos:** Envía suficiente energía de ataque para superar la defensa enemiga

Conceptos Clave

- **Energía Total:** Recurso conservativo que se distribuye entre nodos y aristas
- **Energía de Ataque:** Se asigna a aristas y viaja hacia nodos enemigos (intervalos de 20ms)
- **Energía de Defensa:** Permanece en nodos para protegerlos (actualización cada 30ms)
- **Captura:** Ocurre cuando la energía de ataque supera la defensa del nodo enemigo

---

⚙️ Mecánicas del Juego

🔋 Sistema de Energía

Energía Total

- Recurso **conservativo** compartido entre todos tus nodos
- Se **incrementa** al capturar **Nodos de Energía**
- No tiene límite máximo de concentración en un solo nodo

Distribución de Energía

Tipo	Ubicación	Intervalo	Función
Ataque	Aristas	20ms	Capturar nodos enemigos
Defensa	Nodos	30ms	Proteger nodos propios

Regla de Oro: La defensa siempre se actualiza **antes** que los ataques en cada tick.

🎯 Tipos de Nodos

Icono	Tipo	Efecto
⚪	**Básico**	Funcionalidad estándar de ataque/defensa
🟢	**Energía**	Aumenta energía total al capturarlo
🔴	**Ataque**	Duplica energía de aristas salientes
🔵	**Defensa**	Duplica defensa contra ataques

Icono	Tipo	Efecto
🟢	**Super Energía**	Gran aumento de energía + efectos especiales
🌐	**Neutral**	Sin dueño inicial, capturable por cualquiera

Importante: Los efectos de nodos especiales solo se aplican **mientras los controlas**.

⚔️ Resolución de Conflictos

Orden de Resolución (cada tick)

1. **Actualización de defensa** en todos los nodos
2. **Envío de energía** por aristas
3. **Resolución de conflictos** en aristas
4. **Resolución de ataques** en nodos
5. **Captura de nodos** y aplicación de efectos

Reglas de Conflicto

```
Energías enemigas en arista:
├─ Valores iguales → Ambas destruidas
└─ Valores diferentes → La mayor continúa con diferencia

Ataque vs Defensa:
├─ Ataque > Defensa → Nodo capturado
├─ Ataque = Defensa → Nodo queda neutral
└─ Ataque < Defensa → Ataque destruido

Energía enemiga en nodo aliado:
└─ Se suma a la defensa del nodo

Energías aliadas opuestas:
└─ Se anulan (genera advertencia de desperdicio)
```

🏆 Condiciones de Victoria

1. **Victoria por Dominación:** Controlar ≥70% de nodos durante 10 segundos continuos
2. **Victoria por Tiempo:** Mayor cantidad de nodos al acabar los 3 minutos
3. **Derrota Automática:** Pérdida del nodo inicial (base)
4. **Empate:** Cantidad igual de nodos al finalizar el tiempo

---

🎲 Casos Especiales

- **Sin límites:** Puedes concentrar toda tu energía en un solo nodo
- **Capturas en cascada:** Un nodo capturado puede generar efectos en nodos vecinos

- **Energía en tránsito:** Continúa su curso aunque el nodo cambie de dueño
- **Nodos de articulación:** Su captura puede dividir tu grafo (solo conservas el subgrafo conectado a tu base)

🛠️ Desarrollo

Estructura del Proyecto

```
nexa/
├── src/
│   ├── core/           # Lógica del juego (GameManager, AIController)
│   │   ├── managers/   # Gestores del juego
│   │   └── types/      # Definiciones TypeScript
│   ├── scenes/         # Escenas de Phaser (Boot, Game, MainMenu, etc.)
│   ├── entities/       # Entidades del juego (Nodos, Aristas, Jugadores)
│   ├── ui/             # Componentes de interfaz
│   └── game/           # Punto de entrada del juego
├── public/             # Assets estáticos
├── vite/               # Configuración de Vite (dev/prod)
└── docs/               # Documentación adicional
```

Tecnologías Utilizadas

- **Phaser 3** - Framework de juegos HTML5
- **TypeScript** - Tipado estático
- **Vite** - Build tool y servidor de desarrollo
- **pnpm** - Gestor de paquetes eficiente

Configuración de Alias de Imports

El proyecto usa alias para imports más limpios:

```
// ❌ Antes
import { Game } from '../../../game/scenes/Game';

// ✅ Ahora
import { Game } from '@/game/scenes/Game';
import { Utils } from '@/core/utils';
```

Alias disponibles:

- `@/` → `src/`
- `@/core` → `src/core/`

📚 Documentación Adicional

- **Guía de Inicio** - Primeros pasos y configuración
- **Game Manager** - Arquitectura del gestor del juego
- **Sistema de Tipos** - Definiciones TypeScript
- **Escenas** - Estructura de escenas de Phaser
- **Configuración Vite** - Detalles de configuración

🤝 Contribuir

¡Las contribuciones son bienvenidas! Por favor lee CONTRIBUTING.md para conocer las normas de colaboración.

Flujo de Trabajo Rápido

```
# 1. Crear rama de feature
git checkout -b feature/nueva-funcionalidad

# 2. Hacer cambios y commits
git add .
git commit -m "feat: agregar nueva funcionalidad"

# 3. Push y crear Pull Request
git push origin feature/nueva-funcionalidad
```

📄 Licencia

Este proyecto está bajo la licencia MIT.

👥 Equipo

Desarrollado por: Equipo Nexa
Repositorio: github.com/gustadev24/nexa

¿Te gusta Nexa? ¡Dale una ☆ al repositorio!