Motor Procedural Educacional - V1

Um motor procedural agnóstico de engine para geração de conteúdo em jogos educacionais. Implementa o **Protocolo Entropia Zero** com foco em arquitetura modular, reprodutibilidade e observabilidade.

🎯 O Que É

O Motor Procedural Educacional permite que alunos e professores gerem mapas de jogos proceduralmente usando **intenções de alto nível**. Converte descrições naturais em mapas completos com código pronto para usar em Roblox (V1) e preparado para outras engines.

Exemplo de Uso

TypeScript

```typescript
import { generateDungeonForStudent } from "./src/edu/api/educationalApi";

// Gerar dungeon padrão
const resultado = generateDungeonForStudent("aluno_001");

// Resultado contém:
// - mapa: Estrutura completa com setores e tiles
// - codigoGerado: Script Luau pronto para Roblox
// - logs: Histórico completo da geração
```

🏗️ Arquitetura

Plain Text

```
Intenção (JSON)
    ↓
Compilador (Regras + Config)
    ↓
BSP (Setores) + WFC (Tiles)
    ↓
MapaGerado (JSON)
    ↓
Adaptador Roblox/Luau
    ↓
Código Luau + Build
```

Três Camadas Principais:

1. **Núcleo Procedural** (agnóstico de engine)
    - Wave Function Collapse 2D com entropia de Shannon
    - Binary Space Partitioning para divisão de espaço
    - Serialização JSON para portabilidade
2. **Compilador de Intenção**
    - Converte intenções estruturadas em regras
    - Configura algoritmos de PCG
    - Gera código específico de engine
3. **Adaptador Roblox/Luau**
    - Traduz estruturas do núcleo em Roblox
    - Gera scripts Luau
    - Respeita limites de performance

Veja ARCHITECTURE.md para detalhes completos.

🚀 Quick Start

Instalação

Bash

```
cd motor-procedural-edu
pnpm install
```

Gerar Dungeon

Bash

```
pnpm tsx examples/example-dungeon.ts
```

Rodar Testes

Bash

```
pnpm test
```

📦 Estrutura de Arquivos

Plain Text

```
src/
├── core/                    # Núcleo procedural
│   ├── models/             # Tipos e serialização
│   ├── wfc/                # Wave Function Collapse
│   └── bsp/                # Binary Space Partitioning
├── compiler/               # Compilador de intenções
├── adapters/roblox/        # Adaptador Roblox/Luau
├── edu/api/                # API educacional
└── infra/logging/          # Sistema de logging

tests/                       # Testes automatizados
examples/                    # Exemplos de uso
```

🎓 Conceitos Principais

Intenção

Descrição de alto nível do que o aluno quer gerar:

TypeScript

```typescript
interface Intencao {
  id: string;
  categoria: "Mapa" | "Progressao" | "Economia" | "Social";
  descricaoNatural: string;
  parametros: Record<string, any>;
}
```

Exemplo:

TypeScript

```typescript
{
  id: "intencao_001",
  categoria: "Mapa",
  descricaoNatural: "Quero uma dungeon com 5 áreas e uma sala de boss",
  parametros: {
    quantidadeAreas: 5,
    temBossRoom: true,
    dificuldade: "normal"
```

```
  }
}
```

Mapa Gerado

Resultado completo da geração procedural:

TypeScript

```typescript
interface MapaGerado {
  id: string;
  seed: string;
  dimensoes: { largura: number; altura: number };
  setores: Setor[];        // De BSP
  tiles: TileInstance[];   // De WFC
  metadados: {
    autorId?: string;
    criadoEm: string;
    stats?: {
      numSetores: number;
      numTiles: number;
      densidade?: number;
      tempoGeracaoMs?: number;
    };
  };
}
```

Logging Padronizado

Todos os builds são registrados em JSON:

JSON

```json
{
  "timestamp": "2026-01-10T20:10:00Z",
  "studentId": "aluno_123",
  "intentId": "intencao_001",
  "categoria": "Mapa",
  "engineAlvo": "Roblox",
  "seed": "abc123",
  "mapStats": {
    "numSetores": 5,
    "numTiles": 250
  },
  "buildStatus": "success",
```

```
  "duracao": 245
}
```

🔧 API Educacional

Funções Principais

generateDungeonForStudent(studentId, intencao?)

Gera uma dungeon padrão ou customizada:

TypeScript

```typescript
const resultado = generateDungeonForStudent("aluno_001", {
  id: "dungeon_custom",
  categoria: "Mapa",
  descricaoNatural: "Dungeon com 8 áreas",
  parametros: {
    quantidadeAreas: 8,
    temBossRoom: true,
    dificuldade: "hard"
  }
});
```

generateArenaForStudent(studentId, intencao?)

Gera uma arena de combate:

TypeScript

```typescript
const resultado = generateArenaForStudent("aluno_002");
```

logIntentAndBuild(studentId, intencao)

Compila com logging detalhado:

TypeScript

```typescript
const resultado = logIntentAndBuild("aluno_003", intencao);
console.log(resultado.mapa.metadados.stats);
```

obterLogsDoAluno(studentId)

Recupera histórico de um aluno:

```typescript
const logs = obterLogsDoAluno("aluno_001");
console.log(logs);
```

obterEstatisticas()

Estatísticas gerais:

```typescript
const stats = obterEstatisticas();
console.log(`Total: ${stats.totalBuilds}, Sucessos: ${stats.sucessos}`);
```

🧪 Testes

Cobertura completa em `tests/core.test.ts`:

- **BSP**: Geração, conversão, validação
- **WFC**: Inicialização, colapso, conclusão
- **Serialização**: Idempotência, validação
- **Tipos**: Erros com contexto

```bash
pnpm test
```

📊 Exemplo Completo

```typescript
import { logIntentAndBuild } from "./src/edu/api/educationalApi";
import { Intencao } from "./src/core/models/types";

// 1. Definir intenção
const intencao: Intencao = {
  id: "intencao_exemplo",
  categoria: "Mapa",
  descricaoNatural: "Uma dungeon com 5 áreas principais",
  parametros: {
```

```
    quantidadeAreas: 5,
    temBossRoom: true,
    dificuldade: "normal"
  }
};

// 2. Compilar
const resultado = logIntentAndBuild("aluno_001", intencao);

// 3. Acessar resultados
console.log("Mapa ID:", resultado.mapa.id);
console.log("Seed:", resultado.mapa.seed);
console.log("Setores:", resultado.mapa.setores.length);
console.log("Tiles:", resultado.mapa.tiles.length);
console.log("Tempo:", resultado.mapa.metadados.stats?.tempoGeracaoMs, "ms");

// 4. Usar código Luau em Roblox
console.log("Código Luau:");
console.log(resultado.codigoGerado);

// 5. Verificar logs
console.log("Logs:", resultado.logs);
```

🔄 Reprodutibilidade

Mapas são 100% reproduzíveis com a mesma seed:

TypeScript

```
const intencao = { /* ... */ };

// Primeira geração
const resultado1 = compilarIntencao(intencao, tiles, "seed_123");

// Segunda geração com mesma seed
const resultado2 = compilarIntencao(intencao, tiles, "seed_123");

// Mesmos tiles!
console.log(resultado1.mapa.tiles.length === resultado2.mapa.tiles.length);
```

🎯 Escopo V1

✅ Incluído

- Wave Function Collapse 2D simplificado

- Binary Space Partitioning
- Serialização JSON
- Compilador de Intenção → Regras → Código
- Adaptador Roblox/Luau
- API educacional
- Sistema de logging padronizado
- Testes automatizados

❌ Fora de Escopo

- Geração em tempo real in-game
- Técnicas avançadas (search-based PCG, RL, IA generativa)
- Editor visual completo
- Suporte a múltiplas engines (apenas preparado)

🚀 Roadmap V2+

- [ ] Suporte a Unity e Godot
- [ ] Geração em tempo real (Parallel Luau)
- [ ] Search-based PCG para otimização
- [ ] Editor visual com node graph
- [ ] Integração com IA generativa
- [ ] Análise de dificuldade procedural

📚 Referências

[1] Procedural content generation for games: A survey
https://atlarge-research.com/pdfs/2013-hendrikx-procedural.pdf
[2] Procedural Content Generation in Games: A Survey with Insights on ...
https://arxiv.org/html/2410.15644v1
[3] Software Architecture Requirements for Procedural Content ...
https://sol.sbc.org.br/index.php/sbgames/article/view/37376
[4] A Survey of Programmatic Procedural Content Generation Using Wave Collapse Functions

https://shaankhan.dev/blog/wfc-and-bsp-for-procedural-dungeons-2021/paper-1
[5] Combining Constructive Procedural Dungeon Generation Methods
https://www.sbgames.org/proceedings2020/ComputacaoShort/207911.pdf

📝 Licença

MIT

👨‍💻 Desenvolvido por

Manus AI - Motor Procedural Educacional V1 (Protocolo Entropia Zero)