Motor Procedural Educacional - Arquitetura V1

Visão Geral

O Motor Procedural Educacional é um sistema modular e agnóstico de engine para geração procedural de conteúdo em jogos educacionais. Implementa o **Protocolo Entropia Zero**, focando em baixa entropia arquitetural, didática e observabilidade.

Fluxo Principal

Plain Text

```
Intenção (JSON)
    ↓
Compilador de Intenção
    ↓
Regras + Configurações
    ↓
BSP (Setores) + WFC (Tiles)
    ↓
MapaGerado (JSON)
    ↓
Adaptador Roblox/Luau
    ↓
Código Luau + Build no Workspace
```

Estrutura de Módulos

Plain Text

```
src/
├── core/
│   ├── models/
│   │   ├── types.ts           # Tipos e interfaces principais
│   │   └── serialization.ts   # Serialização JSON
│   ├── wfc/
│   │   └── wfc.ts             # Wave Function Collapse 2D
│   └── bsp/
│       └── bsp.ts             # Binary Space Partitioning
├── compiler/
│   └── intentCompiler.ts      # Compilador Intenção → Regras → Mapa → Código
├── adapters/
```

```
│   └── roblox/
│       └── (adaptador Luau - em desenvolvimento)
├── edu/
│   └── api/
│       └── educationalApi.ts  # API de alto nível para alunos/professores
└── infra/
    └── logging/
        └── logger.ts          # Sistema de logging padronizado
```

Componentes Principais

1. Núcleo Procedural (`core/`)

Responsabilidade: Lógica de geração procedural, agnóstica de engine.

1.1 Modelos de Dados (`core/models/types.ts`)

Define os tipos principais:

- **Tile**: Unidade básica do mapa com regras de adjacência
- **TileInstance**: Instância de um tile em uma posição específica
- **Setor**: Divisão retangular do espaço (resultado de BSP)
- **Regra**: Define comportamento procedural
- **Intenção**: Descrição de alto nível do que gerar
- **MapaGerado**: Resultado completo da geração

Contrato de Borda: Todos os tipos são JSON-serializáveis para portabilidade.

1.2 Wave Function Collapse (`core/wfc/wfc.ts`)

Implementação simplificada de WFC 2D com entropia de Shannon:

- `initializeGrid()` : Cria grid com todas as possibilidades abertas
- `stepCollapse()` : Executa um passo de colapso
- `runToCompletion()` : Loop até conclusão ou contradição

Características:

- Entropia de Shannon para seleção de células
- Propagação de restrições após colapso
- Tratamento de contradições com contexto

1.3 Binary Space Partitioning (`core/bsp/bsp.ts`)

Divisão recursiva de espaço em setores:

- `generateBspTree()` : Cria árvore BSP
- `flattenToSectors()` : Converte folhas em setores
- `validarSetores()` : Verifica tamanhos mínimos

Características:

- Divisão horizontal/vertical aleatória
- Respeita tamanhos mínimos de salas
- Determinístico com seed

1.4 Serialização (`core/models/serialization.ts`)

Contrato JSON para portabilidade:

- `serializeMapa()` : Converte para JSON compacto
- `deserializeMapa()` : Reconstrói a partir de JSON
- `validarMapaDesserializado()` : Verifica integridade

2. Compilador de Intenção (`compiler/`)

Responsabilidade: Converter intenções em mapas gerados.

Fluxo de Compilação

1. **Mapeamento Intenção → Regras**: Tabela que converte categorias e parâmetros em regras
2. **Configuração de BSP**: Derivar profundidade e tamanhos baseado em regras
3. **Configuração de WFC**: Derivar distribuição de tiles baseado em regras
4. **Geração**: Executar BSP + WFC com seed reproduzível
5. **Geração de Código**: Produzir Luau para Roblox
6. **Logging**: Registrar sucesso/erro com contexto

Saída Estruturada

TypeScript

```
PlanoDeGeracao {
  intencao: Intencao
  regras: Regra[]
```

```
  mapa: MapaGerado
  codigoGerado: string  // Luau
  logs: LogEntrada[]
}
```

3. Adaptador Roblox/Luau (`adapters/roblox/`)

Responsabilidade: Traduzir estruturas do núcleo em Roblox.

API Esperada (lado Luau):

Lua

```lua
local MapaModule = require(path.to.MapaModule)
local Builder = require(path.to.RobloxAdapter)

local mapaJson = --[[ string JSON ]]
local mapa = MapaModule.FromJSON(mapaJson)

Builder.BuildFromMapa(workspace, mapa, {
    baseFolderName = "GeneratedMaps",
    maxParts = 5000
})
```

Responsabilidades:

- Criar pasta raiz para mapas gerados
- Instanciar Parts/Models baseado em TileInstances
- Respeitar limite de partes
- Preparar para paralelismo futuro

4. API Educacional (`edu/api/`)

Responsabilidade: Interface amigável para alunos e professores.

Funções Principais:

- `generateDungeonForStudent()` : Gera dungeon padrão
- `generateArenaForStudent()` : Gera arena
- `logIntentAndBuild()` : Compila com logging
- `obterLogsDoAluno()` : Recupera histórico
- `obterEstatisticas()` : Estatísticas gerais

Tiles Padrão Inclusos:

- `chao_normal` : Piso walkable
- `parede_pedra` : Parede sólida
- `porta_madeira` : Transição

5. Sistema de Logging (`infra/logging/`)

Responsabilidade: Observabilidade e reprodutibilidade.

Schema Padronizado:

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",
  "errorType": null,
  "errorMessage": null,
  "duracao": 245
}
```

Métodos:

- `registrarBuild()` : Registra evento
- `registrarSucesso()` : Sucesso com stats
- `registrarErro()` : Erro com tipo e mensagem
- `filtrarPorStatus()` : Query por status
- `filtrarPorEstudante()` : Query por aluno

Pontos de Extensão

Adicionar Novo Algoritmo de PCG

1. Criar módulo em `src/core/`
2. Implementar interface padrão (entrada config, saída estruturada)

3. Integrar em `intentCompiler.ts`
4. Adicionar testes em `tests/`

Adicionar Nova Engine

1. Criar adaptador em `src/adapters/{engine}/`
2. Implementar geração de código específico
3. Documentar API esperada (lado engine)
4. Adicionar suporte em `intentCompiler.ts`

Adicionar Nova Categoria de Intenção

1. Estender enum `Intencao.categoria`
2. Adicionar mapeamento em `mapearIntencaoParaRegras()`
3. Adicionar função wrapper em `educationalApi.ts`
4. Adicionar testes

Decisões de Design

1. Agnóstico de Engine

- **Núcleo não referencia APIs específicas** de Roblox, Unity, etc.
- **Contrato JSON** permite portabilidade
- **Adaptadores isolados** facilitam adição de novas engines

2. Reprodutibilidade

- **RNG seeded**: Mesmo seed = mesmo mapa
- **Logs estruturados**: Rastreabilidade completa
- **Serialização determinística**: JSON é idempotente

3. Baixa Entropia Didática

- **APIs simples**: Um aluno lê exemplo e entende
- **Nomes claros**: `generateDungeonForStudent` é auto-explicativo
- **Erros nomeados**: `ContradictionError` com contexto

4. Observabilidade

- **Logging padronizado**: Todos os builds registram
- **Métricas**: Tempo, número de tiles, densidade
- **Queries**: Filtrar por aluno, status, etc.

Testes

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

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

Executar:

```bash
pnpm test
```

Exemplos

Veja `examples/example-dungeon.ts` para:

1. Usar função padrão
2. Customizar intenção
3. Testar reprodutibilidade

Executar:

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

Limitações da V1

- ✗ Geração em tempo real in-game
- ✗ Técnicas avançadas (search-based, 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