Guia de Contribuição

👋 Bem-vindo!

Obrigado por se interessar em contribuir para o Motor Procedural Educacional! Este documento fornece diretrizes para contribuir ao projeto.

🎯 Visão Geral do Projeto

O Motor Procedural Educacional é um sistema modular para geração procedural de conteúdo em jogos educacionais. A arquitetura segue o **Protocolo Entropia Zero** com foco em:

- **Baixa entropia arquitetural**: Núcleo desacoplado de adaptadores
- **Baixa entropia didática**: APIs simples e auto-explicativas
- **Observabilidade**: Logging padronizado e reprodutibilidade

📁 Estrutura do Projeto

Plain Text

```
src/
├── core/              # Núcleo procedural (WFC, BSP, modelos)
├── 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
```

🚀 Como Começar

1. Clonar o Repositório

Bash

```
git clone https://github.com/seu-usuario/motor-procedural-edu.git
cd motor-procedural-edu
```

2. Instalar Dependências

```bash
pnpm install
```

3. Verificar Configuração

```bash
pnpm check      # Verificar TypeScript
pnpm test       # Rodar testes
pnpm tsx examples/example-dungeon.ts  # Testar exemplo
```

📝 Workflow de Contribuição

1. Criar Branch

```bash
git checkout -b feature/sua-feature
# ou
git checkout -b fix/seu-bug
```

2. Fazer Mudanças

Siga as convenções de código abaixo.

3. Adicionar Testes

Toda nova funcionalidade deve ter testes em `tests/`:

```bash
pnpm test
```

4. Verificar Tipos

```bash
pnpm check
```

5. Commit

Use mensagens descritivas:

```bash
git commit -m "feat: adicionar novo algoritmo de PCG"
git commit -m "fix: corrigir contradição em WFC"
git commit -m "docs: atualizar README"
```

6. Push e Pull Request

```bash
git push origin feature/sua-feature
```

Abra um Pull Request descrevendo suas mudanças.

🎨 Convenções de Código

TypeScript

- **Indentação**: 2 espaços
- **Linha máxima**: 100 caracteres (soft limit)
- **Tipos**: Sempre tipificar (sem `any`)
- **Nomes**: camelCase para variáveis, PascalCase para classes/interfaces

```typescript
// ✓ Bom
interface MapaGerado {
  id: string;
  seed: string;
  tiles: TileInstance[];
}

export function gerarMapa(config: ConfigWFC): MapaGerado {
  // ...
}

// ✗ Ruim
let mapa: any = {};
function gerar_mapa(cfg) {
```

```
  // ...
}
```

Documentação

- **JSDoc**: Documentar funções públicas
- **Comentários**: Explicar "por quê", não "o quê"
- **Exemplos**: Incluir exemplos de uso em comentários

TypeScript

```
/**
 * Executa um passo de colapso WFC
 *
 * @param grid - Grid atual com possibilidades
 * @param tiles - Mapa de tiles disponíveis
 * @param rng - Função de número aleatório
 * @returns Resultado do colapso com status
 *
 * @example
 * const resultado = stepCollapse(grid, tileMap, Math.random);
 * if (resultado.status === "ok") {
 *   console.log("Célula colapsada em", resultado.posicaoColapsada);
 * }
 */
export function stepCollapse(
  grid: GridWFC,
  tiles: Map<string, Tile>,
  rng: () => number
): ResultadoColapso {
  // ...
}
```

Testes

- **Nomes descritivos**: `deve gerar árvore BSP válida`
- **Arrange-Act-Assert**: Estrutura clara
- **Cobertura**: Testar sucesso e erro

TypeScript

```
describe("BSP", () => {
  it("deve gerar árvore BSP válida", () => {
    // Arrange
```

```
    const config: ConfigBSP = { /* ... */ };
    const rng = () => Math.random();

    // Act
    const tree = generateBspTree(config, rng);

    // Assert
    expect(tree).toBeDefined();
    expect(tree.bounds.largura).toBe(100);
  });
});
```

🔄 Pontos de Extensão

Adicionar Novo Algoritmo de PCG

1. Criar módulo em `src/core/{algoritmo}/`
2. Implementar interface padrão
3. Integrar em `intentCompiler.ts`
4. Adicionar testes
5. Documentar em `ARCHITECTURE.md`

Adicionar Nova Engine

1. Criar adaptador em `src/adapters/{engine}/`
2. Implementar geração de código
3. Documentar API esperada
4. Adicionar exemplos
5. Testar integração

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
5. Documentar uso

📚 Documentação

Arquivos Principais

- **README.md**: Visão geral e quick start
- **ARCHITECTURE.md**: Detalhes técnicos e design
- **ROBLOX_INTEGRATION.md**: Guia de integração com Roblox
- **CONTRIBUTING.md**: Este arquivo

Atualizar Documentação

Sempre que adicionar/modificar funcionalidade:

1. Atualizar README se é funcionalidade pública
2. Atualizar ARCHITECTURE.md se muda design
3. Adicionar exemplos em `examples/`
4. Adicionar JSDoc no código

🐛 Reportar Bugs

Use GitHub Issues com template:

Markdown

```
## Descrição
[Descrição clara do bug]

## Passos para Reproduzir
1. [Passo 1]
2. [Passo 2]
3. [Passo 3]

## Comportamento Esperado
[O que deveria acontecer]

## Comportamento Atual
[O que realmente acontece]

## Ambiente
- Node.js: [versão]
- OS: [sistema operacional]
```

💡 Sugerir Melhorias

Use GitHub Discussions ou Issues com label `enhancement`:

Markdown

```
## Descrição da Melhoria
[Descrição clara da melhoria]

## Motivação
[Por que isso seria útil]

## Possível Solução
[Como você implementaria]
```

✅ Checklist para Pull Request

- [ ] Código segue convenções do projeto
- [ ] Testes adicionados/atualizados
- [ ] `pnpm check` passa sem erros
- [ ] `pnpm test` passa
- [ ] Documentação atualizada
- [ ] Commit messages descritivas
- [ ] Sem conflitos com `main`

📞 Perguntas?

- Abra uma Discussion no GitHub
- Crie uma Issue com label `question`
- Consulte a documentação existente

📖 Referências

- Protocolo Entropia Zero
- Arquitetura do Motor
- Integração Roblox

🙏 Obrigado!

Suas contribuições ajudam a tornar o Motor Procedural Educacional melhor para todos!