**📚 Documentação Completa - RPG Turn-Based**

**🏗️ Visão Geral da Arquitetura**

Este é um RPG de turno desenvolvido em **C# + MonoGame** com arquitetura modular e extensível. O jogo segue o padrão **State Machine** para gerenciar diferentes telas e o padrão **Entity** para personagens.

**📁 Estrutura de Arquivos**

RPGTurnBased/

├── 🎮 Program.cs # Entry point

├── 🏛️ Core/

│ ├── Game1.cs # Classe principal do MonoGame

│ └── GameStateManager.cs # Gerenciador de estados

├── 🎭 States/

│ └── BattleState.cs # Estado de batalha

├── 👥 Entities/

│ ├── Entity.cs # Classe base para entidades

│ ├── Player.cs # Jogador

│ └── Enemy.cs # Inimigos

├── 🖥️ UI/

│ ├── TextBox.cs # Caixa de texto

│ └── MenuOption.cs # Sistema de menus

├── ⚙️ Managers/

│ └── InputManager.cs # Gerenciador de input

├── 📦 Content/

│ ├── Content.mgcb # Pipeline de conteúdo

│ └── DefaultFont.spritefont # Configuração da fonte

└── 📋 RPGTurnBased.csproj # Configuração do projeto

**🔍 Detalhamento dos Arquivos**

**🎮 Program.cs**

**Função**: Entry point da aplicação

namespace RPGTurnBased

**O que faz**:

* Cria uma instância de Game1
* Inicia o loop principal do jogo
* Configura [STAThread] para compatibilidade

**Quando é chamado**: Primeiro arquivo executado quando o jogo inicia

**🏛️ Core/Game1.cs**

**Função**: Classe principal do MonoGame (herda de Game)

namespace RPGTurnBased.Core

**Propriedades importantes**:

* SCREEN\_WIDTH = 1024, SCREEN\_HEIGHT = 768 - Resolução do jogo
* \_graphics - Gerenciador de gráficos
* \_spriteBatch - Para desenhar na tela
* \_gameStateManager - Controla diferentes estados
* \_inputManager - Gerencia entrada do usuário

**Métodos principais**:

* Initialize() - Configuração inicial
* LoadContent() - Carrega recursos e inicializa managers
* Update() - Loop de atualização (60 FPS)
* Draw() - Loop de renderização (60 FPS)

**Quando é importante**: Centro de tudo - controla o fluxo principal do jogo

**🏛️ Core/GameStateManager.cs**

**Função**: Gerencia diferentes estados do jogo (Menu, Batalha, Inventário, etc.)

namespace RPGTurnBased.Core

**Estados disponíveis**:

* MainMenu - Menu principal (futuro)
* Battle - Combate ativo
* Inventory - Tela de inventário (futuro)
* GameOver - Fim de jogo (futuro)

**Interface IGameState**:

* LoadContent() - Carregar recursos específicos do estado
* UnloadContent() - Limpar recursos
* Update() - Atualizar lógica do estado
* Draw() - Renderizar estado

**Método principal**:

* ChangeState() - Troca entre diferentes estados

**Quando é usado**: Sempre que você quer mudar de tela (batalha → menu → inventário)

**🎭 States/BattleState.cs**

**Função**: Implementa toda a lógica de combate turn-based

namespace RPGTurnBased.States

**Fases da batalha**:

* PlayerTurn - Jogador escolhe ação
* EnemyTurn - Inimigos atacam
* Victory - Jogador venceu
* Defeat - Jogador foi derrotado

**Componentes principais**:

* \_player - Instância do jogador
* \_enemies - Lista de inimigos na batalha
* \_messageBox - Caixa de texto inferior
* \_actionMenu - Menu de ações (Attack, Magic, etc.)
* \_targetMenu - Menu para selecionar alvos
* \_defaultFont - Fonte para renderização de texto

**Posições dos inimigos**:

Vector2[] \_enemyPositions = {

new Vector2(300, 200), // Esquerda

new Vector2(500, 150), // Centro

new Vector2(700, 200), // Direita

new Vector2(400, 280) // Frente

};

**Fluxo de batalha**:

1. StartPlayerTurn() - Mostra menu de ações
2. StartTargetSelection() - Escolher alvo (se necessário)
3. ExecutePlayerAction() - Executa ação do jogador
4. CheckBattleEnd() - Verifica vitória/derrota
5. StartEnemyTurn() - Inimigos atacam
6. Volta ao passo 1

**Quando é ativo**: Durante todas as batalhas do jogo

**👥 Entities/Entity.cs**

**Função**: Classe base abstrata para todas as entidades (jogador, inimigos, NPCs futuros)

namespace RPGTurnBased.Entities

**Propriedades base**:

* Name - Nome da entidade
* Health/MaxHealth - Vida atual/máxima
* Attack - Poder de ataque
* Defense - Defesa
* Speed - Velocidade (determina ordem dos turnos)
* Position - Posição na tela
* Color - Cor do retângulo (temporário)
* Bounds - Área de colisão/clique
* IsAlive - Se está vivo (Health > 0)
* IsSelected - Se está selecionado

**Métodos importantes**:

* TakeDamage(int damage) - Receber dano
* Heal(int amount) - Curar
* CalculateDamage(Entity target) - Calcular dano contra alvo
* Update() - Atualizar posição/estado
* Draw() - Desenhar na tela (retângulo + barra de vida)

**Preparado para sprites**:

// Estrutura comentada para futuras implementações

// public Texture2D Sprite { get; protected set; }

// public Rectangle SourceRectangle { get; protected set; }

**Quando é usado**: Base para Player, Enemy e futuras entidades

**👥 Entities/Player.cs**

**Função**: Representa o jogador, herda de Entity

namespace RPGTurnBased.Entities

**Propriedades específicas**:

* Level - Nível atual
* Experience - Experiência atual
* Mana/MaxMana - Mana para magias
* DefaultPosition - Posição fixa (sempre de costas na parte inferior)

**Stats iniciais**:

* Health: 100, Attack: 15, Defense: 8, Speed: 12
* Level: 1, Mana: 20
* Cor: Azul
* Tamanho: 64x80 pixels

**Métodos específicos**:

* GainExperience(int exp) - Ganhar experiência
* CanCastSpell(int manaCost) - Verificar se pode usar magia
* UseMana(int amount) - Gastar mana
* RestoreMana(int amount) - Recuperar mana

**Renderização especial**:

* Barra de vida (vermelha)
* Barra de mana (azul)
* Sempre posicionado na parte inferior da tela

**Quando é usado**: Durante toda a gameplay - é o avatar do jogador

**👥 Entities/Enemy.cs**

**Função**: Representa inimigos, herda de Entity

namespace RPGTurnBased.Entities

**Tipos de inimigos**:

enum EnemyType { Goblin, Orc, Skeleton, Dragon }

**Stats por tipo**:

* **Goblin**: 30 HP, 8 ATK, 3 DEF, 15 SPD - Verde, 48x64px, 15 EXP
* **Orc**: 60 HP, 15 ATK, 8 DEF, 8 SPD - Verde escuro, 64x80px, 35 EXP
* **Skeleton**: 45 HP, 12 ATK, 5 DEF, 10 SPD - Cinza claro, 56x72px, 25 EXP
* **Dragon**: 150 HP, 25 ATK, 12 DEF, 6 SPD - Vermelho escuro, 96x120px, 100 EXP

**Propriedades específicas**:

* Type - Tipo do inimigo
* ExperienceReward - XP que dá quando derrotado

**Método estático**:

* CreateRandomEnemy(Vector2 position) - Cria inimigo aleatório

**Renderização**:

* Retângulo colorido baseado no tipo
* Barra de vida vermelha acima
* Indicador de seleção (borda amarela)

**Quando é usado**: Oponentes em todas as batalhas

**🖥️ UI/TextBox.cs**

**Função**: Caixa de texto estilo RPG clássico

namespace RPGTurnBased.UI

**Propriedades visuais**:

* BackgroundColor - Fundo semi-transparente preto
* BorderColor - Borda branca
* TextColor - Texto branco
* Font - SpriteFont para renderização

**Funcionalidades**:

* **Quebra de linha inteligente** - Baseada na largura real da fonte
* **Múltiplas linhas** - Suporta textos longos
* **Padding** - Espaçamento interno (10px)
* **Borda** - 2px de largura
* **Fallback** - Retângulos coloridos se fonte falhar

**Métodos principais**:

* SetText(string text) - Mudar texto e recalcular linhas
* SetFont(SpriteFont font) - Definir fonte
* UpdateTextLines() - Quebrar texto em linhas
* Draw() - Renderizar caixa + texto

**Onde aparece**: Parte inferior da tela durante batalhas

**🖥️ UI/MenuOption.cs**

**Função**: Sistema completo de menus navegáveis

namespace RPGTurnBased.UI

**Classe MenuOption**:

* Text - Texto da opção
* OnSelect - Ação executada quando selecionada
* Bounds - Área clicável
* IsEnabled - Se está habilitada

**Classe MenuSystem**:

* Lista de opções
* Índice da opção selecionada
* Cores configuráveis

**Cores padrão**:

* Fundo: Azul escuro semi-transparente
* Texto: Branco
* Selecionado: Amarelo
* Desabilitado: Cinza

**Navegação**:

* NavigateUp()/NavigateDown() - Mover seleção
* SelectCurrentOption() - Executar ação
* Pula opções desabilitadas automaticamente

**Tipos de menu no jogo**:

* **Menu de ações**: Attack, Magic, Defend, Run
* **Menu de alvos**: Lista de inimigos vivos

**Renderização**:

* Fundo com borda
* Highlight da opção selecionada
* Texto real com SpriteFont
* Fallback para retângulos se necessário

**⚙️ Managers/InputManager.cs**

**Função**: Gerencia toda entrada do usuário (teclado/mouse)

namespace RPGTurnBased.Managers

**Estados de input**:

* \_currentKeyboardState - Estado atual do teclado
* \_previousKeyboardState - Estado anterior (para detectar "press")
* \_currentMouseState - Estado atual do mouse
* \_previousMouseState - Estado anterior

**Métodos de detecção**:

* IsKeyPressed() - Tecla foi pressionada (não segurada)
* IsKeyDown() - Tecla está sendo segurada
* IsLeftMouseButtonPressed() - Clique do mouse

**Controles mapeados**:

* **Navegação**: Setas OU WASD
* **Confirmar**: Enter OU Space OU Z
* **Cancelar**: Escape OU X
* **Sair**: Escape (na tela principal)

**Métodos de conveniência**:

* IsUpPressed(), IsDownPressed() - Navegação vertical
* IsLeftPressed(), IsRightPressed() - Navegação horizontal
* IsConfirmPressed(), IsCancelPressed() - Ações principais

**Atualização**: Chamado a cada frame para capturar mudanças

**📦 Content/DefaultFont.spritefont**

**Função**: Configuração da fonte para renderização de texto

XnaContent xmlns:Graphics="Microsoft.Xna.Framework.Content.Pipeline.Graphics"

**Configurações**:

* **FontName**: Arial
* **Size**: 14 pontos
* **Style**: Regular
* **UseKerning**: true (espaçamento melhorado)
* **CharacterRegions**: ASCII 32-126 (letras, números, símbolos básicos)

**O que inclui**:

* Letras maiúsculas e minúsculas
* Números 0-9
* Símbolos de pontuação
* Caracteres especiais básicos

**Processamento**: MonoGame converte para textura bitmap otimizada

**📦 Content/Content.mgcb**

**Função**: Pipeline de conteúdo do MonoGame

#begin DefaultFont.spritefont

/importer:FontDescriptionImporter

/processor:FontDescriptionProcessor

**O que faz**:

* Define como processar cada arquivo de conteúdo
* Converte .spritefont para formato binário otimizado
* Configura compressão e formatos de textura
* Gerencia dependências entre recursos

**Quando é usado**: Durante a compilação (dotnet build)

**📋 RPGTurnBased.csproj**

**Função**: Configuração do projeto .NET

<TargetFramework>net8.0-windows</TargetFramework>

**Dependências**:

* MonoGame.Framework.DesktopGL - Engine principal
* MonoGame.Content.Builder.Task - Pipeline de conteúdo

**Configurações**:

* Target: .NET 8.0 para Windows
* Tipo: Aplicação Windows (WinExe)
* Referência ao Content.mgcb para processamento automático

**🔄 Fluxo de Execução**

**Inicialização:**

1. Program.Main() → Game1.Run()
2. Game1.Initialize() → Game1.LoadContent()
3. GameStateManager criado → BattleState ativado
4. BattleState.LoadContent() → Carrega fonte, cria entidades, setup UI

**Loop Principal (60 FPS):**

1. Game1.Update() → GameStateManager.Update() → BattleState.Update()
2. InputManager.Update() → Captura entrada do usuário
3. Estado processa input → Atualiza entidades → Verifica condições
4. Game1.Draw() → GameStateManager.Draw() → BattleState.Draw()
5. Entidades se desenham → UI se desenha → Frame completo

**Turno de Batalha:**

1. **Player Turn**: Mostra menu → Input → Seleção → Execução
2. **Damage Calculation**: Attack - Defense = Damage
3. **Check Victory**: Todos inimigos mortos? Jogador morto?
4. **Enemy Turn**: IA simples → Sempre ataca jogador
5. **Repeat**: Volta para Player Turn

**🎯 Pontos de Extensão**

**Para adicionar novos inimigos:**

* Modificar EnemyType enum em Enemy.cs
* Adicionar cases nos métodos GetEnemyXXX()

**Para novos estados de jogo:**

* Criar classe implementando IGameState
* Adicionar enum em GameStateManager.GameStates
* Implementar case em ChangeState()

**Para novas ações de batalha:**

* Adicionar opção em SetupActionMenu() no BattleState
* Implementar método de execução
* Adicionar case em ExecutePlayerAction()

**Para sprites reais:**

* Descomentar propriedades Sprite nas entidades
* Carregar texturas no LoadContent()
* Modificar métodos Draw() para usar sprites

**🏗️ Padrões Arquiteturais Utilizados**

**State Pattern:**

* GameStateManager + IGameState
* Permite trocar facilmente entre diferentes telas

**Entity Pattern:**

* Entity base + Player/Enemy derivados
* Facilita adição de novos tipos de personagens

**Manager Pattern:**

* InputManager centraliza controle de entrada
* Permite trocar facilmente esquemas de controle

**Composition over Inheritance:**

* UI como componentes separados (TextBox, MenuSystem)
* Podem ser reutilizados em diferentes estados

**🚀 Estado Atual vs Potencial Futuro**

**✅ Implementado:**

* Sistema de turnos completo
* Interface funcional com texto legível
* Múltiplos tipos de inimigos
* Sistema de dano e vida
* Navegação por menus
* Arquitetura extensível

**🔮 Próximas Expansões Possíveis:**

* Sistema de sprites reais
* Múltiplas magias e habilidades
* Inventário e equipamentos
* Sistema de levels e progressão
* Save/Load de progresso
* Diferentes tipos de batalha
* Mapa mundial (overworld)
* NPCs e diálogos
* Sistema de quests
* Música e efeitos sonoros

Esta documentação serve como guia completo para entender, modificar e expandir o projeto! 🎮