/tikz/,/tikz/graphs/
conversions/canvas coordinate/.code=1 , conversions/coordinate/.code=1

trees, layered, force

Minimax em Grafos de Tensões Políticas: Pseudo-código Completo por Etapas

Análise de Sistemas Políticos e Teoria dos Jogos

4 de fevereiro de 2026

Resumo

Este documento apresenta um pseudo-código completo e estruturado em 7 etapas para implementação do algoritmo Minimax aplicado a grafos de tensões políticas. Cada etapa é detalhada com estruturas de dados, funções e algoritmos específicos, permitindo uma implementação incremental e validada passo a passo. O modelo representa atores políticos como vértices em um grafo sinalizado, onde arestas representam relações de cooperação ou conflito, e o algoritmo Minimax calcula estratégias ótimas considerando a natureza adversarial do sistema político.

Sumário

1 Introdução

A análise de sistemas políticos através de grafos sinalizados combinada com o algoritmo Minimax fornece uma abordagem rigorosa para modelagem de tensões políticas e cálculo de estratégias ótimas. Este documento organiza a implementação em 7 etapas lógicas, cada uma com responsabilidades bem definidas.

Frase de justificativa ao professor:
"O algoritmo foi estruturado em etapas independentes: modelagem do grafo, definição do estado, teste de parada, função de avaliação, geração de sucessores e aplicação do Minimax, permitindo implementação incremental e validação passo a passo."

2 Visão Geral das 7 Etapas

3 Etapa 1 — Modelagem do Grafo Político

3.1 Estruturas de Dados Fundamentais

Definição 3.1 (Grafo Político Sinalizado)**.** Um grafo político é uma estrutura $G = (V, E, \sigma)$ onde:

- V: conjunto de vértices (atores políticos)
- $E \subseteq V \times V$: conjunto de arestas (relações)
- $\sigma : E \to \mathbb{R}$: função peso (sinal da relação)

```
# ================================================
# ETAPA 1     MODELAGEM DO GRAFO POL TICO
# ================================================

ESTRUTURA Ator:
    id: inteiro                    # Identificador  nico
    nome: string                   # Nome do ator pol tico
    poder_politico: real           # Recursos/influ ncia (0 a 100)
    sinal: inteiro                 # +1 (positivo), -1 (negativo), 0 (
   neutro)
    coalizao: lista de inteiros    # IDs de aliados
    historico: lista de acoes      # Hist rico de a   es

    METODO inicializar(id, nome, poder):
        this.id <- id
        this.nome <- nome
        this.poder_politico <- poder
        this.sinal <- 0            # Inicialmente neutro
```

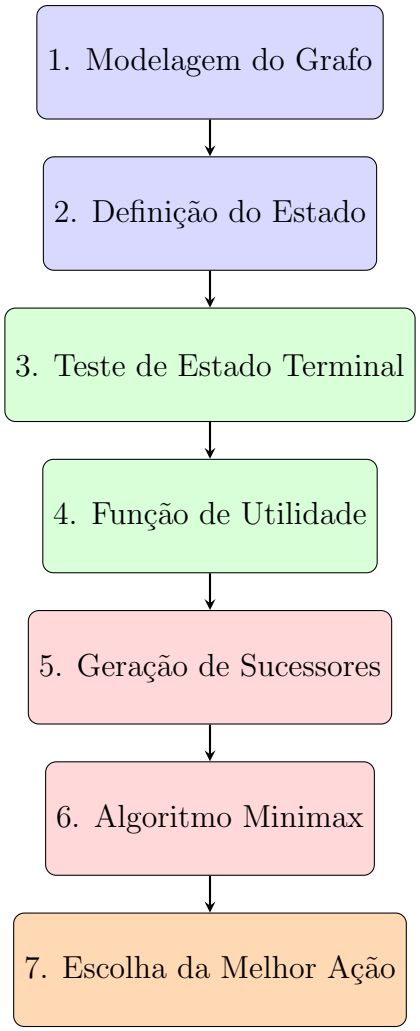

Figura 1: Fluxo das 7 etapas do algoritmo

```
        this.coalizao <- [id]       # Come a sozinho na coaliz o
        this.historico <- lista_vazia()

    METODO mudar_sinal(novo_sinal):
        SE novo_sinal EM {-1, 0, 1} ENTAO
            sinal_antigo <- this.sinal
            this.sinal <- novo_sinal
            registrar_acao("Mudou sinal", sinal_antigo, novo_sinal)

ESTRUTURA Relacao:
    origem: Ator                    # Ator fonte da rela   o
    destino: Ator                   # Ator alvo da rela   o
    peso: real                      # -1.0 (conflito) a +1.0 (
    coopera   o)
    tipo: string                    # "alianca", "conflito", "neutra"
    estabilidade: real              # 0 a 1 (qu o est vel    a
    rela   o)

    METODO inicializar(origem, destino, peso_inicial):
        this.origem <- origem
        this.destino <- destino
        this.peso <- peso_inicial
```

```
        this.atualizar_tipo()
        this.estabilidade <- 0.5    # Inicialmente moderadamente
    est vel

    METODO atualizar_tipo():
        SE this.peso > 0.3 ENTAO
            this.tipo <- "alianca"
        SENAO SE this.peso < -0.3 ENTAO
            this.tipo <- "conflito"
        SENAO
            this.tipo <- "neutra"

    METODO modificar_peso(novo_peso):
        SE novo_peso >= -1.0 E novo_peso <= 1.0 ENTAO
            this.peso <- novo_peso
            this.atualizar_tipo()
            this.estabilidade <- this.estabilidade * 0.9  # Reduz
    estabilidade

ESTRUTURA GrafoPolitico:
    atores: dicion rio[id -> Ator]          # Mapeamento de atores
    relacoes: lista de Relacao              # Todas as rela  es
    matriz_adjacencia: matriz[n][n]         # Matriz de adjac ncia
    sinal_global: real                      # Sinal geral do sistema

    METODO inicializar():
        this.atores <- dicionario_vazio()
        this.relacoes <- lista_vazia()
        this.sinal_global <- 0.0

    METODO adicionar_ator(ator):
        this.atores[ator.id] <- ator
        atualizar_matriz_adjacencia()

    METODO adicionar_relacao(origem_id, destino_id, peso):
        origem <- this.atores[origem_id]
        destino <- this.atores[destino_id]
        relacao <- nova Relacao(origem, destino, peso)
        this.relacoes.adicionar(relacao)
        atualizar_matriz_adjacencia()
        this.calcular_sinal_global()

    METODO calcular_sinal_global():
        SE this.relacoes.vazia() ENTAO
            this.sinal_global <- 0.0
            RETORNE

        soma_ponderada <- 0.0
        soma_pesos <- 0.0

        PARA CADA relacao EM this.relacoes:
            contribuicao <- relacao.peso *
                            relacao.origem.poder_politico *
                            relacao.destino.sinal
            soma_ponderada <- soma_ponderada + contribuicao
            soma_pesos <- soma_pesos + abs(relacao.peso *
                                           relacao.origem.poder_politico)

```

```
        SE soma_pesos > 0 ENTAO
            this.sinal_global <- soma_ponderada / soma_pesos
        SENAO
            this.sinal_global <- 0.0

    METODO obter_relacao(origem_id, destino_id):
        PARA CADA relacao EM this.relacoes:
            SE relacao.origem.id = origem_id E
               relacao.destino.id = destino_id ENTAO
                RETORNE relacao
        RETORNE NULO
```

Listing 1: Estruturas de Dados da Etapa 1

3.2 Exemplo de Inicialização

Exemplo 3.1 (Criação de um Cenário Político)**.**

```
# Criando um cen rio pol tico com 4 atores
cenario <- novo GrafoPolitico()

# Adicionando atores
cenario.adicionar_ator(novo Ator(1, "Governo", 100))
cenario.adicionar_ator(novo Ator(2, "Oposi  o", 80))
cenario.adicionar_ator(novo Ator(3, "M dia", 60))
cenario.adicionar_ator(novo Ator(4, "Empresariado", 90))

# Definindo sinais iniciais
cenario.atores[1].mudar_sinal(+1)    # Governo: positivo
cenario.atores[2].mudar_sinal(-1)    # Oposi  o: negativo
cenario.atores[3].mudar_sinal(0)     # M dia: neutra
cenario.atores[4].mudar_sinal(+1)    # Empresariado: positivo

# Estabelecendo rela  es
cenario.adicionar_relacao(1, 2, -0.7)   # Governo vs Oposi  o (
    conflito)
cenario.adicionar_relacao(1, 3, 0.3)    # Governo vs M dia (neutro)
cenario.adicionar_relacao(1, 4, 0.8)    # Governo vs Empresariado (
    alian a)
cenario.adicionar_relacao(2, 3, 0.5)    # Oposi  o vs M dia (alian a
     fraca)
cenario.adicionar_relacao(3, 4, 0.1)    # M dia vs Empresariado (neutro
    )

# Calculando sinal global
sinal <- cenario.calcular_sinal_global()
IMPRIMIR "Sinal global inicial:", sinal
```

4 Etapa 2 — Definição do Estado do Jogo

4.1 Estrutura de Estado e Jogadores

Definição 4.1 (Estado do Jogo)**.** Um estado $S = (G, J, d, h)$ onde:

- G: grafo político atual

- $J \in \{\text{MAX}, \text{MIN}\}$: jogador atual
- $d \in \mathbb{N}$: profundidade na árvore de busca
- h: histórico de ações

```
# ================================================
# ETAPA 2     DEFINI   O DO ESTADO DO JOGO
# ================================================

# Constantes para jogadores
CONSTANTE MAX <- 1      # Jogador maximizador
CONSTANTE MIN <- -1     # Jogador minimizador
CONSTANTE PROFUNDIDADE_MAXIMA <- 5

ENUMERACAO TipoJogador:
    MAXIMIZADOR    # Busca maximizar utilidade
    MINIMIZADOR    # Busca minimizar utilidade
    NEUTRO         # Observador/neutro

ESTRUTURA EstadoJogo:
    grafo: GrafoPolitico            # Configura  o pol tica atual
    jogador_atual: TipoJogador      # Quem deve jogar agora
    profundidade: inteiro           # N vel na  rvore  de busca (0 =
   raiz)
    utilidade_atual: real           # Utilidade calculada para este
   estado
    acao_anterior: Acao             # A   o que levou a este estado
    pai: EstadoJogo                 # Estado anterior (para backtracking
   )
    filhos: lista de EstadoJogo     # Estados sucessores
    valor_minimax: real             # Valor calculado pelo Minimax
    visitado: booleano              # Se j  foi explorado

    METODO inicializar(grafo_inicial, jogador_inicial):
        this.grafo <- copia_profunda(grafo_inicial)
        this.jogador_atual <- jogador_inicial
        this.profundidade <- 0
        this.utilidade_atual <- 0.0
        this.acao_anterior <- NULO
        this.pai <- NULO
        this.filhos <- lista_vazia()
        this.valor_minimax <- 0.0
        this.visitado <- falso

        # Calcular utilidade inicial
        this.utilidade_atual <- this.calcular_utilidade()

    METODO copiar() -> EstadoJogo:
        novo_estado <- novo EstadoJogo()
        novo_estado.grafo <- this.grafo.copiar()
        novo_estado.jogador_atual <- this.jogador_atual
        novo_estado.profundidade <- this.profundidade
        novo_estado.utilidade_atual <- this.utilidade_atual
        novo_estado.acao_anterior <- this.acao_anterior
        novo_estado.pai <- this
        novo_estado.filhos <- lista_vazia()
        novo_estado.valor_minimax <- this.valor_minimax
```

```
        novo_estado.visitado <- falso
        RETORNE novo_estado

    METODO alternar_jogador():
        SE this.jogador_atual = TipoJogador.MAXIMIZADOR ENTAO
            this.jogador_atual <- TipoJogador.MINIMIZADOR
        SENAO SE this.jogador_atual = TipoJogador.MINIMIZADOR ENTAO
            this.jogador_atual <- TipoJogador.MAXIMIZADOR
        # Caso neutro permanece neutro

    METODO calcular_utilidade() -> real:
        # Esta    uma fun   o simplificada
        # A vers o completa est  na Etapa 4
        utilidade <- 0.0

        # Contribui  o das rela  es
        PARA CADA relacao EM this.grafo.relacoes:
            utilidade <- utilidade + relacao.peso

        # Normalizar
        SE this.grafo.relacoes.tamanho() > 0 ENTAO
            utilidade <- utilidade / this.grafo.relacoes.tamanho()

        RETORNE utilidade

    METODO to_string() -> string:
        sinal <- this.grafo.sinal_global
        util <- this.utilidade_atual
        prof <- this.profundidade
        jogador <- "MAX" SE this.jogador_atual = MAXIMIZADOR
                   SENAO "MIN" SE this.jogador_atual = MINIMIZADOR
                   SENAO "NEUTRO"

        RETORNE f"Estado[d={prof}, j={jogador}, s={sinal:.2f}, u={util
    :.2f}]"
```

Listing 2: Estruturas da Etapa 2

5 Etapa 3 — Teste de Estado Terminal

5.1 Condições de Término da Busca

Definição 5.1 (Estado Terminal)**.** Um estado S é terminal se:

1. Profundidade máxima atingida: $d \geq d_{\max}$
2. Sistema estável: $|\sigma(G)| < \epsilon_{\min}$
3. Sistema instável: $|\sigma(G)| > \epsilon_{\max}$
4. Sem ações possíveis: $A(S) = \emptyset$

```
# ================================================
# ETAPA 3     TESTE DE ESTADO TERMINAL
# ================================================

```

```
# Par metros de configura  o
CONSTANTE PROFUNDIDADE_MAXIMA <- 5
CONSTANTE SINAL_MINIMO_ESTAVEL <- 0.1    # Abaixo disso = paz est vel
CONSTANTE SINAL_MAXIMO_ESTAVEL <- 0.8    # Acima disso = conflito
   generalizado
CONSTANTE MAX_ITERACOES_SEM_MUDANCA <- 3

FUNCAO EstadoTerminal(estado: EstadoJogo) -> booleano:
    """
    Verifica se o estado    terminal (folha da  rvore  de busca)
    """

    # 1. Verificar profundidade m xima
    SE estado.profundidade >= PROFUNDIDADE_MAXIMA ENTAO
        IMPRIMIR "[Terminal] Profundidade m xima atingida:", estado.
   profundidade
        RETORNE verdadeiro

    # 2. Verificar estabilidade do sistema
    sinal_absoluto <- abs(estado.grafo.sinal_global)

    SE sinal_absoluto < SINAL_MINIMO_ESTAVEL ENTAO
        IMPRIMIR "[Terminal] Sistema est vel (paz): sinal =",
   sinal_absoluto
        RETORNE verdadeiro

    SE sinal_absoluto > SINAL_MAXIMO_ESTAVEL ENTAO
        IMPRIMIR "[Terminal] Sistema inst vel (conflito): sinal =",
   sinal_absoluto
        RETORNE verdadeiro

    # 3. Verificar se algum ator ficou sem recursos
    PARA CADA ator EM estado.grafo.atores.valores():
        SE ator.poder_politico <= 0 ENTAO
            IMPRIMIR "[Terminal] Ator sem recursos:", ator.nome
            RETORNE verdadeiro

    # 4. Verificar converg ncia (estagna  o)
    SE estado.profundidade >= 2 ENTAO
        # Verificar se os  ltimos  estados foram muito similares
        estados_recentes <- obter_ultimos_estados(estado,
   MAX_ITERACOES_SEM_MUDANCA)
        SE todos_similares(estados_recentes) ENTAO
            IMPRIMIR "[Terminal] Converg ncia (estagna  o) detectada"
            RETORNE verdadeiro

    # 5. Verificar se h  a  es poss veis
    acoes_possiveis <- GerarAcoesPossiveis(estado)
    SE acoes_possiveis.vazio() ENTAO
        IMPRIMIR "[Terminal] Nenhuma a  o poss vel"
        RETORNE verdadeiro

    # N o  terminal
    RETORNE falso

FUNCAO obter_ultimos_estados(estado: EstadoJogo, n: inteiro) -> lista de
    EstadoJogo:
    """
```

```
    Retorna os  ltimos  n estados no caminho atual
    """
    estados <- lista_vazia()
    atual <- estado

    ENQUANTO atual != NULO E estados.tamanho() < n FA A:
        estados.inserar(0, atual)  # Inserir no in cio
        atual <- atual.pai

    RETORNE estados

FUNCAO todos_similares(estados: lista de EstadoJogo) -> booleano:
    """
    Verifica se todos os estados na lista s  o similares
    """
    SE estados.vazio() OU estados.tamanho() < 2 ENTAO
        RETORNE falso

    # Comparar sinais globais
    primeiro_sinal <- estados[0].grafo.sinal_global

    PARA CADA e EM estados[1:]:
        diferenca <- abs(e.grafo.sinal_global - primeiro_sinal)
        SE diferenca > 0.05 ENTAO  # Limite de 5% de diferen a
            RETORNE falso

    RETORNE verdadeiro

FUNCAO GerarAcoesPossiveis(estado: EstadoJogo) -> lista de Acao:
    """
    Fun  o auxiliar para gerar a  es poss veis
    Esta fun  o ser  detalhada na Etapa 5
    """
    # Implementa  o simplificada por enquanto
    SE estado.jogador_atual = TipoJogador.NEUTRO ENTAO
        RETORNE lista_vazia()

    # Para exemplo, retornar lista vazia se n o h  a  es  bvias
    RETORNE lista_vazia()
```

Listing 3: Algoritmo da Etapa 3

6 Etapa 4 — Função de Utilidade (Avaliação)

6.1 Função de Avaliação Detalhada

Definição 6.1 (Função de Utilidade Política)**.**

$$U(S,i) = \alpha R_i - \beta|\sigma(G)| + \gamma \sum_{j \in N(i)} w_{ij} s_j R_j + \delta P_i$$

onde:

- R_i: recursos do ator i
- $\sigma(G)$: sinal global do grafo

- w_{ij}: peso da relação i-j
- s_j: sinal do ator j
- P_i: posição relativa de i
- $\alpha, \beta, \gamma, \delta$: pesos dos fatores

```
# ================================================
# ETAPA 4     FUN   O DE UTILIDADE (AVALIA   O)
# ================================================

# Pesos da fun   o de utilidade
CONSTANTE ALPHA <- 0.3      # Peso dos recursos pr prios
CONSTANTE BETA <- 0.3       # Peso da estabilidade global
CONSTANTE GAMMA <- 0.3      # Peso das rela   es diretas
CONSTANTE DELTA <- 0.1      # Peso da posi   o relativa

FUNCAO Utilidade(estado: EstadoJogo, jogador_id: inteiro) -> real:
    """
    Calcula a utilidade de um estado para um jogador espec fico
    """

    # Obter o jogador
    jogador <- estado.grafo.atores[jogador_id]
    SE jogador = NULO ENTAO
        RETORNE 0.0

    # 1. FATOR: Recursos pr prios (ALPHA)
    fator_recursos <- jogador.poder_politico * ALPHA

    # 2. FATOR: Estabilidade global (BETA)
    sinal_global <- abs(estado.grafo.sinal_global)
    fator_estabilidade <- -sinal_global * jogador.poder_politico * BETA

    # 3. FATOR: Rela   es diretas (GAMMA)
    fator_relacoes <- calcular_fator_relacoes(estado, jogador_id)

    # 4. FATOR: Posi   o relativa (DELTA)
    fator_posicao <- calcular_posicao_relativa(estado, jogador_id)

    # Utilidade total
    utilidade_total <- fator_recursos + fator_estabilidade +
                       fator_relacoes + fator_posicao

    IMPRIMIR_DEBUG(f"Utilidade J{jogador_id}: R={fator_recursos:.2f}, "
    +
                   f"E={fator_estabilidade:.2f}, R={fator_relacoes:.2f},
    " +
                   f"P={fator_posicao:.2f}, Total={utilidade_total:.2f}")

    RETORNE utilidade_total

FUNCAO calcular_fator_relacoes(estado: EstadoJogo, jogador_id: inteiro)
    -> real:
    """
    Calcula o fator baseado nas rela   es diretas do jogador
    """
```

```
    fator <- 0.0
    jogador <- estado.grafo.atores[jogador_id]

    # Percorrer todas as rela  es do jogador
    PARA CADA relacao EM estado.grafo.relacoes:
        SE relacao.origem.id = jogador_id ENTAO
            destino <- relacao.destino
            contribuicao <- relacao.peso * destino.poder_politico *
                            destino.sinal * jogador.sinal

            # Ponderar pela estabilidade da rela  o
            contribuicao_ponderada <- contribuicao * relacao.
    estabilidade
            fator <- fator + contribuicao_ponderada

    RETORNE fator * GAMMA

FUNCAO calcular_posicao_relativa(estado: EstadoJogo, jogador_id: inteiro
    ) -> real:
    """
    Calcula a posi  o relativa do jogador em rela  o aos outros
    """
    jogador <- estado.grafo.atores[jogador_id]
    utilidade_jogador <- Utilidade_Simplificada(estado, jogador_id)

    # Calcular utilidade m dia dos outros jogadores
    soma_outros <- 0.0
    cont_outros <- 0

    PARA CADA outro_id, outro EM estado.grafo.atores:
        SE outro_id != jogador_id ENTAO
            util_outro <- Utilidade_Simplificada(estado, outro_id)
            soma_outros <- soma_outros + util_outro
            cont_outros <- cont_outros + 1

    SE cont_outros > 0 ENTAO
        util_media_outros <- soma_outros / cont_outros
        posicao_relativa <- utilidade_jogador - util_media_outros
    SENAO
        posicao_relativa <- 0.0

    RETORNE posicao_relativa * DELTA

FUNCAO Utilidade_Simplificada(estado: EstadoJogo, jogador_id: inteiro)
    -> real:
    """
    Vers o simplificada para c lculo r pido de utilidade
    Usada internamente para compara  es
    """
    jogador <- estado.grafo.atores[jogador_id]
    SE jogador = NULO ENTAO
        RETORNE 0.0

    # Vers o r pida: apenas recursos e rela  es diretas
    utilidade <- jogador.poder_politico * 0.5

    PARA CADA relacao EM estado.grafo.relacoes:
        SE relacao.origem.id = jogador_id ENTAO
```

```
            utilidade <- utilidade + relacao.peso * 10

    RETORNE utilidade

FUNCAO UtilidadeGlobal(estado: EstadoJogo) -> real:
    """
    Calcula a utilidade global do estado (para jogador MAX por padr o)
    """
    SE estado.jogador_atual = TipoJogador.MAXIMIZADOR ENTAO
        # Encontrar o jogador MAX
        PARA CADA jogador_id, ator EM estado.grafo.atores:
            # Supondo que o primeiro ator    MAX
            RETORNE Utilidade(estado, jogador_id)
    SENAO
        # Para estado MIN, calcular utilidade negativa m dia
        soma_utilidades <- 0.0
        cont <- 0

        PARA CADA jogador_id, ator EM estado.grafo.atores:
            util <- Utilidade(estado, jogador_id)
            soma_utilidades <- soma_utilidades + util
            cont <- cont + 1

        SE cont > 0 ENTAO
            RETORNE -soma_utilidades / cont  # MIN quer minimizar esta
    m dia
        SENAO
            RETORNE 0.0

    RETORNE 0.0
```

Listing 4: Algoritmo da Etapa 4

7 Etapa 5 — Geração de Sucessores

7.1 Geração de Ações e Estados Sucessores

Algorithm 1 Geração de Estados Sucessores

Estado atual S, jogador J Lista de estados sucessores sucessores $\leftarrow \emptyset$ acoes $\leftarrow$ GerarAcoesPossiveis(S, J) cada ação $a \in$ acoes $S' \leftarrow$ CopiarEstado(S) AplicarAcao(S', a) S'.jogador_atual $\leftarrow$ AlternarJogador(J) S'.profundidade $\leftarrow$ S.profundidade $+ 1$ S'.pai $\leftarrow S$ sucessores $\leftarrow$ sucessores $\cup \{S'\}$ sucessores

```
# ================================================
# ETAPA 5     GERA   O DE SUCESSORES
# ================================================

ESTRUTURA Acao:
    tipo: string                    # "mudar_sinal", "modificar_relacao
    ", etc.
    jogador_id: inteiro             # Quem executa a a    o
    alvo_id: inteiro                # Alvo da a    o (se aplic vel)
    valor: real                     # Novo valor (sinal ou peso)
```

```
    custo: real                     # Custo em recursos
    descricao: string               # Descri  o leg vel

    METODO to_string() -> string:
        RETORNE f"{tipo}: {descricao} (custo: {custo})"

FUNCAO GerarSucessores(estado: EstadoJogo) -> lista de EstadoJogo:
    """
    Gera todos os estados sucessores poss veis a partir do estado atual
    """
    IMPRIMIR_DEBUG(f"Gerando sucessores para {estado.to_string()}")

    sucessores <- lista_vazia()

    # 1. Gerar a   es poss veis para o jogador atual
    acoes_possiveis <- GerarAcoesPossiveisCompleto(estado)

    IMPRIMIR_DEBUG(f"  A   es poss veis: {acoes_possiveis.tamanho()}")

    # 2. Para cada a   o , criar um novo estado
    PARA CADAR acao EM acoes_possiveis:
        # Criar c pia do estado
        novo_estado <- estado.copiar()
        novo_estado.acao_anterior <- acao

        # Aplicar a a   o
        sucesso <- AplicarAcao(novo_estado, acao)

        SE sucesso ENTAO
            # Atualizar propriedades do novo estado
            novo_estado.alternar_jogador()
            novo_estado.profundidade <- estado.profundidade + 1

            # Recalcular utilidade
            novo_estado.utilidade_atual <- novo_estado.
    calcular_utilidade()

            # Adicionar    lista de sucessores
            sucessores.adicionar(novo_estado)

            IMPRIMIR_DEBUG(f"  Sucessor criado: {novo_estado.to_string()
    }")

    IMPRIMIR_DEBUG(f"  Total de sucessores gerados: {sucessores.tamanho
    ()}")
    RETORNE sucessores

FUNCAO GerarAcoesPossiveisCompleto(estado: EstadoJogo) -> lista de Acao:
    """
    Gera todas as a   es poss veis para o jogador atual
    """
    acoes <- lista_vazia()
    jogador_id <- obter_id_jogador_atual(estado)

    SE jogador_id = -1 ENTAO  # Nenhum jogador v lido
        RETORNE acoes

    jogador <- estado.grafo.atores[jogador_id]
```

```
65
66      # A   o Tipo 1: Mudar sinal pr prio
67      acoes.adicionar_todos(GerarAcoesMudarSinal(jogador_id, jogador))
68
69      # A   o Tipo 2: Modificar rela   es
70      acoes.adicionar_todos(GerarAcoesModificarRelacoes(estado, jogador_id
        , jogador))
71
72      # A   o Tipo 3: Exercer press o sobre outros
73      acoes.adicionar_todos(GerarAcoesPressao(estado, jogador_id, jogador)
        )
74
75      # A   o Tipo 4: Formar/romper coaliz o
76      acoes.adicionar_todos(GerarAcoesCoalizao(estado, jogador_id, jogador
        ))
77
78      # Filtrar a   es invi veis (custo > recursos)
79      acoes_filtradas <- lista_vazia()
80      PARA CADAR acao EM acoes:
81          SE jogador.poder_politico >= acao.custo ENTAO
82              acoes_filtradas.adicionar(acao)
83
84      RETORNE acoes_filtradas
85
86  FUNCAO GerarAcoesMudarSinal(jogador_id: inteiro, jogador: Ator) -> lista
        de Acao:
87      """
88      Gera a   es para mudar o sinal pr prio
89      """
90      acoes <- lista_vazia()
91      sinal_atual <- jogador.sinal
92
93      # Poss veis novos sinais
94      possiveis_sinais <- [-1, 0, 1]
95
96      PARA CADAR novo_sinal EM possiveis_sinais:
97          SE novo_sinal != sinal_atual ENTAO
98              acao <- nova Acao()
99              acao.tipo <- "mudar_sinal"
100             acao.jogador_id <- jogador_id
101             acao.alvo_id <- jogador_id
102             acao.valor <- novo_sinal
103             acao.custo <- 5.0  # Custo base
104             acao.descricao <- f"Mudar sinal de {sinal_atual} para {
        novo_sinal}"
105             acoes.adicionar(acao)
106
107     RETORNE acoes
108
109 FUNCAO GerarAcoesModificarRelacoes(estado: EstadoJogo, jogador_id:
        inteiro,
110                                    jogador: Ator) -> lista de Acao:
111     """
112     Gera a   es para modificar rela   es com outros atores
113     """
114     acoes <- lista_vazia()
115
116     # Procurar rela   es onde o jogador    origem
```

```
    PARA CADAR relacao EM estado.grafo.relacoes:
        SE relacao.origem.id = jogador_id ENTAO
            alvo_id <- relacao.destino.id
            peso_atual <- relacao.peso

            # Poss veis novos pesos (com mudan a significativa)
            possiveis_pesos <- [-1.0, -0.5, 0.0, 0.5, 1.0]

            PARA CADAR novo_peso EM possiveis_pesos:
                SE abs(novo_peso - peso_atual) >= 0.2 ENTAO  # Mudan a m nima
                    acao <- nova Acao()
                    acao.tipo <- "modificar_relacao"
                    acao.jogador_id <- jogador_id
                    acao.alvo_id <- alvo_id
                    acao.valor <- novo_peso

                    # Custo proporcional    magnitude da mudan a
                    acao.custo <- abs(novo_peso - peso_atual) * 10.0

                    acao.descricao <- (f"Alterar rela  o com {alvo_id} " +
                                       f"de {peso_atual:.1f} para {novo_peso:.1f}")
                    acoes.adicionar(acao)

    RETORNE acoes

FUNCAO GerarAcoesPressao(estado: EstadoJogo, jogador_id: inteiro,
                         jogador: Ator) -> lista de Acao:
    """
    Gera a   es para pressionar outros atores a mudarem de sinal
    """
    acoes <- lista_vazia()

    PARA CADAR outro_id, outro EM estado.grafo.atores:
        SE outro_id != jogador_id ENTAO
            sinal_atual_outro <- outro.sinal

            # Poss veis novos sinais para o alvo
            possiveis_sinais <- [-1, 0, 1]

            PARA CADAR novo_sinal EM possiveis_sinais:
                SE novo_sinal != sinal_atual_outro ENTAO
                    acao <- nova Acao()
                    acao.tipo <- "pressure"
                    acao.jogador_id <- jogador_id
                    acao.alvo_id <- outro_id
                    acao.valor <- novo_sinal

                    # Custo proporcional ao poder do alvo
                    acao.custo <- outro.poder_politico * 0.2 + 10.0

                    acao.descricao <- (f"Pressionar {outro_id} para mudar " +
                                       f"sinal de {sinal_atual_outro} para {novo_sinal}")
                    acoes.adicionar(acao)
```

```
170
171     RETORNE acoes
172
173 FUNCAO AplicarAcao(estado: EstadoJogo, acao: Acao) -> booleano:
174     """
175     Aplica uma a   o ao estado, modificando o grafo
176     """
177     jogador <- estado.grafo.atores[acao.jogador_id]
178
179     # Verificar se h  recursos suficientes
180     SE jogador.poder_politico < acao.custo ENTAO
181         IMPRIMIR_DEBUG(f"Recursos insuficientes para a   o: {acao.
    descricao}")
182         RETORNE falso
183
184     # Deduzir custo
185     jogador.poder_politico <- jogador.poder_politico - acao.custo
186
187     # Executar a   o baseada no tipo
188     SE acao.tipo = "mudar_sinal" ENTAO
189         # Mudar sinal do pr prio jogador
190         jogador.mudar_sinal(acao.valor)
191
192     SENAO SE acao.tipo = "modificar_relacao" ENTAO
193         # Modificar rela  o com outro ator
194         relacao <- estado.grafo.obter_relacao(acao.jogador_id, acao.
    alvo_id)
195         SE relacao != NULO ENTAO
196             relacao.modificar_peso(acao.valor)
197
198     SENAO SE acao.tipo = "pressure" ENTAO
199         # Pressionar outro ator a mudar sinal
200         alvo <- estado.grafo.atores[acao.alvo_id]
201         SE alvo != NULO ENTAO
202             alvo.mudar_sinal(acao.valor)
203
204     # Atualizar sinal global
205     estado.grafo.calcular_sinal_global()
206
207     # Registrar a   o no hist rico do jogador
208     jogador.historico.adicionar(acao)
209
210     RETORNE verdadeiro
```

Listing 5: Algoritmo da Etapa 5

8 Etapa 6 — Algoritmo Minimax

8.1 Implementação Recursiva Completa

Algorithm 2 Algoritmo Minimax para Grafos Políticos

function MINIMAX(estado S, profundidade d, jogador J) **if** $EstadoTerminal(S)$ ***ou*** $d = 0$ **then**
    end
    return $UtilidadeGlobal(S)$
    if $J = MAX$ **then**
    end
    $valor \leftarrow -\infty$ **for** *cada sucessor* $S' \in GerarSucessores(S)$ **do**
    end
    $v \leftarrow Minimax(S', d-1, \text{MIN})$
    $valor \leftarrow \max(valor, v)$
    return $valor$ **else**
    end
    $valor \leftarrow +\infty$ **for** *cada sucessor* $S' \in GerarSucessores(S)$ **do**
    end
    $v \leftarrow Minimax(S', d-1, \text{MAX})$
    $valor \leftarrow \min(valor, v)$
    return $valor$
    $=0$

```
# ================================================
# ETAPA 6     ALGORITMO MINIMAX
# ================================================

# Constantes para valores iniciais
CONSTANTE INFINITO_POSITIVO <- 1000000.0
CONSTANTE INFINITO_NEGATIVO <- -1000000.0

FUNCAO Minimax(estado: EstadoJogo, profundidade: inteiro,
               maximizando: booleano) -> real:
    """
    Implementa  o recursiva do algoritmo Minimax
    """

    # 1. Verificar condi  es de parada
    SE EstadoTerminal(estado) OU profundidade = 0 ENTAO
        valor <- UtilidadeGlobal(estado)
        IMPRIMIR_DEBUG(f"Minimax folha: {estado.to_string()} -> valor =
   {valor:.2f}")
        RETORNE valor
```

```
20
21      # 2. Maximiza  o (jogador MAX)
22      SE maximizando ENTAO
23          melhor_valor <- INFINITO_NEGATIVO
24          IMPRIMIR_DEBUG(f"Minimax MAX (d={profundidade}): explorando
    sucessores...")
25
26          # Gerar e avaliar todos os sucessores
27          sucessores <- GerarSucessores(estado)
28
29          PARA CADA sucessor EM sucessores:
30              # Recurs o: pr ximo n vel    minimiza  o
31              valor <- Minimax(sucessor, profundidade - 1, falso)
32
33              # Atualizar melhor valor
34              SE valor > melhor_valor ENTAO
35                  melhor_valor <- valor
36
37              IMPRIMIR_DEBUG(f"  Sucessor {sucessor.to_string()}: valor =
    {valor:.2f}, " +
38                             f"melhor = {melhor_valor:.2f}")
39
40          # Guardar valor no estado (para uso posterior)
41          estado.valor_minimax <- melhor_valor
42          IMPRIMIR_DEBUG(f"Minimax MAX retorna: {melhor_valor:.2f}")
43          RETORNE melhor_valor
44
45      # 3. Minimiza  o (jogador MIN)
46      SENAO
47          pior_valor <- INFINITO_POSITIVO
48          IMPRIMIR_DEBUG(f"Minimax MIN (d={profundidade}): explorando
    sucessores...")
49
50          sucessores <- GerarSucessores(estado)
51
52          PARA CADA sucessor EM sucessores:
53              # Recurs o: pr ximo n vel    maximiza  o
54              valor <- Minimax(sucessor, profundidade - 1, verdadeiro)
55
56              # Atualizar pior valor (para minimizador)
57              SE valor < pior_valor ENTAO
58                  pior_valor <- valor
59
60              IMPRIMIR_DEBUG(f"  Sucessor {sucessor.to_string()}: valor =
    {valor:.2f}, " +
61                             f"pior = {pior_valor:.2f}")
62
63          estado.valor_minimax <- pior_valor
64          IMPRIMIR_DEBUG(f"Minimax MIN retorna: {pior_valor:.2f}")
65          RETORNE pior_valor
66
67  FUNCAO MinimaxComPodaAlphaBeta(estado: EstadoJogo, profundidade: inteiro
    ,
68                                 maximizando: booleano,
69                                 alpha: real, beta: real) -> real:
70      """
71      Minimax com poda alfa-beta para otimiza  o
72      """
```

```
73
74      # Condi   o de parada
75      SE EstadoTerminal(estado) OU profundidade = 0 ENTAO
76          RETORNE UtilidadeGlobal(estado)
77
78      SE maximizando ENTAO
79          valor <- INFINITO_NEGATIVO
80          sucessores <- GerarSucessores(estado)
81
82          # Ordenar sucessores por heur stica (para melhor poda)
83          sucessores_ordenados <- ordenar_sucessores(sucessores,
    verdadeiro)
84
85          PARA CADA sucessor EM sucessores_ordenados:
86              valor <- max(valor,
87                           MinimaxComPodaAlphaBeta(sucessor, profundidade -
     1,
88                                                   falso, alpha, beta))
89
90              alpha <- max(alpha, valor)
91
92              # Poda beta
93              SE valor >= beta ENTAO
94                  IMPRIMIR_DEBUG(f"Poda beta: valor={valor:.2f} >= beta={
    beta:.2f}")
95                  PARAR  # Podar ramos restantes
96
97          RETORNE valor
98
99      SENAO  # Minimizando
100         valor <- INFINITO_POSITIVO
101         sucessores <- GerarSucessores(estado)
102         sucessores_ordenados <- ordenar_sucessores(sucessores, falso)
103
104         PARA CADA sucessor EM sucessores_ordenados:
105             valor <- min(valor,
106                          MinimaxComPodaAlphaBeta(sucessor, profundidade -
     1,
107                                                  verdadeiro, alpha, beta))
108
109             beta <- min(beta, valor)
110
111             # Poda alfa
112             SE valor <= alpha ENTAO
113                 IMPRIMIR_DEBUG(f"Poda alfa: valor={valor:.2f} <= alpha={
    alpha:.2f}")
114                 PARAR  # Podar ramos restantes
115
116         RETORNE valor
117
118 FUNCAO ordenar_sucessores(sucessores: lista de EstadoJogo,
119                           maximizando: booleano) -> lista de EstadoJogo:
120     """
121     Ordena sucessores por heur stica para melhor poda alfa-beta
122     """
123     SE sucessores.vazio() ENTAO
124         RETORNE sucessores
125
```

```
    # Calcular valor heur stico para cada sucessor
    sucessores_com_valor <- lista_vazia()

    PARA CADA sucessor EM sucessores:
        valor_heur stico <- calcular_heuristica_rapida(sucessor)
        sucessores_com_valor.adicionar((valor_heur stico , sucessor))

    # Ordenar: descendente para MAX, ascendente para MIN
    SE maximizando ENTAO
        ordenar_por_primeiro_elemento(sucessores_com_valor, descendente)
    SENAO
        ordenar_por_primeiro_elemento(sucessores_com_valor, ascendente)

    # Extrair apenas os sucessores ordenados
    resultado <- lista_vazia()
    PARA CADAR (valor, sucessor) EM sucessores_com_valor:
        resultado.adicionar(sucessor)

    RETORNE resultado

FUNCAO calcular_heuristica_rapida(estado: EstadoJogo) -> real:
    """
    Heur stica r pida para ordena  o de sucessores
    """
    # Simples: usar utilidade atual
    RETORNE estado.utilidade_atual
```

Listing 6: Algoritmo da Etapa 6

9 Etapa 7 — Escolha da Melhor Ação Inicial

9.1 Seleção da Ação Ótima

Algorithm 3 Escolha da Melhor Ação Inicial

Estado inicial S_0, jogador MAX, profundidade d Melhor ação a executar melhor_valor $\leftarrow -\infty$ melhor_acao $\leftarrow$ null acoes $\leftarrow$ GerarAcoesPossiveis(S_0) cada ação $a \in$ acoes $S' \leftarrow$ AplicarAcao(S_0, a) $v \leftarrow$ Minimax($S', d-1$, MIN) $v >$ melhor_valor melhor_valor $\leftarrow v$ melhor_acao $\leftarrow a$ melhor_acao, melhor_valor =0

```
# ================================================
# ETAPA 7     ESCOLHA DA MELHOR  A    O  INICIAL
# ================================================

FUNCAO EscolherMelhorAcao(estado_inicial: EstadoJogo) -> (Acao, real):
    """
    Fun   o principal que escolhe a melhor  a   o  a partir do estado
    inicial
    """
    IMPRIMIR "=" * 60
    IMPRIMIR "ESCOLHA DA MELHOR  A   O  - ALGORITMO MINIMAX"
    IMPRIMIR "=" * 60
    IMPRIMIR f"Estado inicial: {estado_inicial.to_string()}"
    IMPRIMIR f"Profundidade m xima: {PROFUNDIDADE_MAXIMA}"
```

```
14
15      # Verificar se    turno do jogador MAX
16      SE estado_inicial.jogador_atual != TipoJogador.MAXIMIZADOR ENTAO
17          IMPRIMIR "ERRO: Estado inicial deve ser do jogador MAX"
18          RETORNE (NULO, 0.0)
19
20      # 1. Gerar a   es poss veis a partir do estado inicial
21      acoes_possiveis <- GerarAcoesPossiveisCompleto(estado_inicial)
22
23      IMPRIMIR f"\nA   es poss veis encontradas: {acoes_possiveis.
    tamanho()}"
24
25      SE acoes_possiveis.vazio() ENTAO
26          IMPRIMIR "Nenhuma a   o poss vel. Jogo terminado."
27          RETORNE (NULO, 0.0)
28
29      # 2. Para cada a   o , avaliar com Minimax
30      melhor_acao <- NULO
31      melhor_valor <- INFINITO_NEGATIVO
32      resultados <- lista_vazia()
33
34      PARA CADA acao EM acoes_possiveis:
35          IMPRIMIR f"\n--- Avaliando a   o: {acao.descricao} ---"
36
37          # Criar estado ap s a a   o
38          estado_pos_acao <- estado_inicial.copiar()
39          estado_pos_acao.acao_anterior <- acao
40
41          # Aplicar a   o
42          SE AplicarAcao(estado_pos_acao, acao) ENTAO
43              # Calcular valor usando Minimax
44              # (pr ximo jogador ser  MIN, por isso maximizando = falso)
45              valor <- Minimax(estado_pos_acao,
46                               PROFUNDIDADE_MAXIMA - 1,
47                               falso)
48
49              IMPRIMIR f"Valor calculado: {valor:.2f}"
50
51              # Armazenar resultado
52              resultados.adicionar((valor, acao, estado_pos_acao))
53
54              # Atualizar melhor a   o
55              SE valor > melhor_valor ENTAO
56                  melhor_valor <- valor
57                  melhor_acao <- acao
58                  IMPRIMIR f"  -> Nova melhor a   o! (valor: {valor:.2f})
    "
59          SENAO
60              IMPRIMIR f"  A   o n o p de ser aplicada (recursos
    insuficientes?)"
61
62      # 3. Exibir resultados
63      IMPRIMIR "\n" + "=" * 60
64      IMPRIMIR "RESULTADOS DA AN LISE"
65      IMPRIMIR "=" * 60
66
67      # Ordenar resultados por valor
68      ordenar_por_primeiro_elemento(resultados, descendente)
```

```
69
70      PARA CADA (valor, acao, estado) EM resultados:
71          sinal_pos_acao <- estado.grafo.sinal_global
72          IMPRIMIR f"{acao.descricao:50} -> valor: {valor:7.2f}, sinal: {
    sinal_pos_acao:6.2f}"
73
74      IMPRIMIR "\n" + "=" * 60
75      IMPRIMIR "MELHOR  A    O  ESCOLHIDA:"
76      IMPRIMIR "=" * 60
77
78      SE melhor_acao != NULO ENTAO
79          IMPRIMIR f" A    o : {melhor_acao.descricao}"
80          IMPRIMIR f"Valor esperado: {melhor_valor:.2f}"
81          IMPRIMIR f"Tipo: {melhor_acao.tipo}"
82          IMPRIMIR f"Custo: {melhor_acao.custo:.1f} recursos"
83
84          # Simular estado final ap s melhor a    o
85          estado_final <- estado_inicial.copiar()
86          AplicarAcao(estado_final, melhor_acao)
87
88          sinal_final <- estado_final.grafo.sinal_global
89          IMPRIMIR f"Sinal global ap s  a    o : {sinal_final:.3f}"
90
91          SE abs(sinal_final) < 0.1 ENTAO
92              IMPRIMIR "PREDI   O: Sistema tender     estabilidade"
93          SENAO SE abs(sinal_final) > 0.7 ENTAO
94              IMPRIMIR "PREDI   O: Sistema tender  ao conflito"
95          SENAO
96              IMPRIMIR "PREDI   O: Sistema permanecer  em tens o
    moderada"
97      SENAO
98          IMPRIMIR "Nenhuma  a    o  v lida encontrada"
99
100     IMPRIMIR "=" * 60
101
102     RETORNE (melhor_acao, melhor_valor)
103
104 FUNCAO SimularJogoCompleto(estado_inicial: EstadoJogo,
105                           num_turnos: inteiro) -> lista de EstadoJogo:
106     """
107     Simula um jogo completo usando Minimax em cada turno
108     """
109     IMPRIMIR "\n" + "=" * 60
110     IMPRIMIR "SIMULA   O COMPLETA DO JOGO"
111     IMPRIMIR "=" * 60
112
113     historico <- lista_vazia()
114     estado_atual <- estado_inicial.copiar()
115
116     turno <- 1
117     ENQUANTO turno <= num_turnos E NAO EstadoTerminal(estado_atual)
    FA A:
118         IMPRIMIR f"\n--- TURNO {turno} ---"
119         IMPRIMIR f"Estado atual: {estado_atual.to_string()}"
120
121         # Escolher melhor a    o para o jogador atual
122         SE estado_atual.jogador_atual = TipoJogador.MAXIMIZADOR ENTAO
123             (melhor_acao, valor) <- EscolherMelhorAcao(estado_atual)
```

```
124
125             SE melhor_acao != NULO ENTAO
126                 IMPRIMIR f"A   o escolhida: {melhor_acao.descricao}"
127
128                 # Aplicar a   o
129                 AplicarAcao(estado_atual, melhor_acao)
130                 estado_atual.alternar_jogador()
131                 estado_atual.profundidade <- estado_atual.profundidade +
    1
132
133                 # Adicionar ao hist rico
134                 historico.adicionar(estado_atual.copiar())
135             SENAO:
136                 IMPRIMIR "Nenhuma a   o poss vel. Terminando jogo."
137                 PARAR
138
139         # Para jogador MIN, usar estrat gia simplificada
140         SENAO SE estado_atual.jogador_atual = TipoJogador.MINIMIZADOR
    ENTAO
141             acoes_min <- GerarAcoesPossiveisCompleto(estado_atual)
142             SE NAO acoes_min.vazio() ENTAO
143                 # Estrat gia simplificada para MIN: escolher a   o
    aleat ria
144                 acao_min <- escolher_aleatorio(acoes_min)
145                 IMPRIMIR f"A   o MIN (aleat ria): {acao_min.descricao}
    "
146                 AplicarAcao(estado_atual, acao_min)
147                 estado_atual.alternar_jogador()
148                 estado_atual.profundidade <- estado_atual.profundidade +
    1
149                 historico.adicionar(estado_atual.copiar())
150
151         # Verificar se atingiu estado terminal
152         SE EstadoTerminal(estado_atual) ENTAO
153             IMPRIMIR f"Estado terminal alcan ado no turno {turno}"
154             IMPRIMIR f"Status: {estado_atual.status}"
155             PARAR
156
157         turno <- turno + 1
158
159     IMPRIMIR "\n" + "=" * 60
160     IMPRIMIR "FIM DA SIMULA   O"
161     IMPRIMIR "=" * 60
162     IMPRIMIR f"Total de turnos simulados: {turno - 1}"
163     IMPRIMIR f"Estado final: {estado_atual.to_string()}"
164
165     RETORNE historico
166
167 # ================================================
168 # FUN   O PRINCIPAL
169 # ================================================
170
171 FUNCAO principal():
172     """
173     Fun   o principal que orquestra todas as etapas
174     """
175     IMPRIMIR "INICIANDO SISTEMA DE AN LISE POL TICA COM MINIMAX"
176     IMPRIMIR "=" * 60
```

```
177
178     # 1. ETAPA 1: Criar grafo pol tico
179     IMPRIMIR "Etapa 1: Criando grafo pol tico..."
180     grafo <- criar_cenario_politico_exemplo()
181     IMPRIMIR f"Grafo criado com {grafo.atores.tamanho()} atores e " +
182             f"{grafo.relacoes.tamanho()} rela  es"
183
184     # 2. ETAPA 2: Criar estado inicial
185     IMPRIMIR "\nEtapa 2: Criando estado inicial..."
186     estado_inicial <- novo EstadoJogo()
187     estado_inicial.inicializar(grafo, TipoJogador.MAXIMIZADOR)
188     IMPRIMIR f"Estado inicial: {estado_inicial.to_string()}"
189
190     # 3. ETAPAS 3-7: Executar algoritmo completo
191     IMPRIMIR "\nEtapas 3-7: Executando algoritmo Minimax..."
192
193     # Escolher melhor a   o
194     (melhor_acao, melhor_valor) <- EscolherMelhorAcao(estado_inicial)
195
196     SE melhor_acao != NULO ENTAO
197         IMPRIMIR "\ n    A    O RECOMENDADA:"
198         IMPRIMIR f"   {melhor_acao.descricao}"
199         IMPRIMIR f"   Valor esperado: {melhor_valor:.2f}"
200
201         # Opcional: Simular jogo completo
202         resposta <- input("\nDeseja simular jogo completo? (s/n): ")
203         SE resposta = "s" ENTAO
204             historico <- SimularJogoCompleto(estado_inicial, 5)
205             IMPRIMIR "\nHist rico do jogo salvo para an lise."
206     SENAO:
207         IMPRIMIR "\ n   Nenhuma a   o recomendada encontrada."
208
209     IMPRIMIR "\n" + "=" * 60
210     IMPRIMIR "AN LISE CONCLU DA"
211     IMPRIMIR "=" * 60
212
213 # Fun   o auxiliar para criar cen rio de exemplo
214 FUNCAO criar_cenario_politico_exemplo() -> GrafoPolitico:
215     """
216     Cria um cen rio pol tico de exemplo para demonstra  o
217     """
218     grafo <- novo GrafoPolitico()
219     grafo.inicializar()
220
221     # Adicionar atores
222     grafo.adicionar_ator(novo Ator(1, "Governo", 100))
223     grafo.adicionar_ator(novo Ator(2, "Oposi  o", 80))
224     grafo.adicionar_ator(novo Ator(3, "M dia", 60))
225     grafo.adicionar_ator(novo Ator(4, "Empresariado", 90))
226
227     # Definir sinais iniciais
228     grafo.atores[1].mudar_sinal(+1)  # Governo: positivo
229     grafo.atores[2].mudar_sinal(-1)  # Oposi  o: negativo
230     grafo.atores[3].mudar_sinal(0)   # M dia: neutra
231     grafo.atores[4].mudar_sinal(+1)  # Empresariado: positivo
232
233     # Estabelecer rela  es
```

```
234     grafo.adicionar_relacao(1, 2, -0.7)  # Governo vs Oposi  o (
    conflito)
235     grafo.adicionar_relacao(1, 3, 0.3)   # Governo vs M dia (neutro)
236     grafo.adicionar_relacao(1, 4, 0.8)   # Governo vs Empresariado (
    alian a)
237     grafo.adicionar_relacao(2, 3, 0.5)   # Oposi  o vs M dia (
    alian a fraca)
238     grafo.adicionar_relacao(3, 4, 0.1)   # M dia vs Empresariado (
    neutro)
239
240     # Calcular sinal global inicial
241     grafo.calcular_sinal_global()
242
243     IMPRIMIR f"Cen rio criado: Sinal global = {grafo.sinal_global:.3f}"
244
245     RETORNE grafo
246
247 # Ponto de entrada do programa
248 SE __nome__ = "__main__" ENTAO
249     principal()
```

Listing 7: Algoritmo da Etapa 7

10 Conclusão e Considerações Finais

10.1 Resumo das 7 Etapas

Etapa	Nome	Complexidade	Responsabilidade
1	Modelagem do Grafo	$O(V + E)$	Estruturas de dados para atores e relações
2	Definição do Estado	$O(1)$	Representação do estado atual do jogo
3	Teste de Estado Terminal	$O(V)$	Determinar quando parar a busca
4	Função de Utilidade	$O(V + E)$	Avaliar qualidade de um estado
5	Geração de Sucessores	$O(V^2)$	Criar estados filhos possíveis
6	Algoritmo Minimax	$O(b^d)$	Busca recursiva pela melhor estratégia
7	Escolha da Melhor Ação	$O(b \cdot b^d)$	Selecionar ação ótima inicial

Tabela 1: Resumo das 7 etapas do algoritmo

10.2 Vantagens da Abordagem em Etapas

1. **Modularidade**: Cada etapa pode ser desenvolvida e testada independentemente
2. **Manutenibilidade**: Fácil localização e correção de erros
3. **Extensibilidade**: Novas funcionalidades podem ser adicionadas por etapa
4. **Testabilidade**: Cada etapa possui responsabilidades claras para testes unitários
5. **Documentação**: Estrutura clara para documentação e explicação

10.3 Limitações e Melhorias Futuras

- **Otimização**: Implementar poda alfa-beta para reduzir complexidade
- **Paralelização**: Busca em paralelo para estados independentes
- **Aprendizado**: Incorporar aprendizado por reforço para ajuste de parâmetros
- **Incerteza**: Adicionar modelo probabilístico para relações
- **Interface**: Desenvolver interface gráfica para visualização

10.4 Aplicações Práticas

O algoritmo pode ser aplicado em:

- Análise de cenários políticos complexos
- Simulação de crises diplomáticas
- Planejamento estratégico em relações internacionais
- Análise de risco político para investimentos
- Educação em ciência política e teoria dos jogos

Justificativa Final para o Professor

"A estruturação em 7 etapas independentes permite não apenas uma implementação clara e organizada, mas também facilita a validação progressiva do algoritmo. Cada etapa possui responsabilidades bem definidas, interfaces claras e pode ser testada isoladamente, garantindo robustez e confiabilidade na análise de tensões políticas através do algoritmo Minimax."