Universidade Federal do Rio Grande do Norte
Bacharelado em tecnologia da informação

Matheus Ronaldo de Souza

Projeto: Jogo da Velha com IA básica

Objetivo

Desenvolver uma implementação em C do clássico jogo da velha com interface de linha de comando, incluindo um oponente controlado por IA capaz de tomar decisões estratégicas básicas. O projeto traz um ambiente interativo para demonstração prática dos conceitos de programação estruturada, oferecendo uma aplicação que exercita lógica, tomada de decisão e implementação de algoritmos contra um oponente computacional.

O jogo da velha foi selecionado por apresentar adequação aos requisitos da Unidade 1, permitindo a aplicação dos tópicos estudados (variáveis, condicionais, repetições, funções e vetores) em um contexto intuitivo e compreensivo.

Metodologia

Para o desenvolvimento do projeto, foram utilizadas as seguintes ferramentas:

- **Linguagem de programação:** C (padrão C11)
- **Compilador:** GCC 12.2.0 por meio do MSYS2
- **Editor de código:** Visual Studio Code
- **Controle de versão:** Git integrado ao GitHub
- **Sistema operacional:** Windows 10

➔ **Como foram usadas as estruturas condicionais?**

As estruturas condicionais foram empregadas em múltiplas camadas do sistema. Na validação de entrada, a função posicaoValida() utiliza condições para verificar se as coordenadas estão dentro dos limites do tabuleiro (0-2) e se a célula está desocupada. Para a lógica de detecção de vitória, verificarVencedor() implementa condições que analisam sequências horizontais, verticais e diagonais. O sistema de controle de fluxo em main() emprega switch-case para gerenciar o menu de opções, enquanto condições aninhadas na IA (podeVencer()) permitem a tomada de decisão estratégica.

➔ **Qual a lógica das estruturas de repetição implementadas?**

As estruturas de repetição foram estrategicamente distribuídas: main() utiliza um loop while(1) para manter a execução do jogo até condições de término. A IA em obterJogadaIA() emprega um loop for para percorrer o vetor tentativas de prioridades pré-definidas. Para simulação de jogadas, podeVencer() utiliza loops for aninhados para varrer sistematicamente o tabuleiro tabuleiro[3][3], testando cada posição vazia.

➔ **Como os vetores foram aplicados no projeto?**

Os vetores podem ser flexíveis dependendo do uso, então o tabuleiro[3][3] (matriz char) representa o estado do jogo, mapeando diretamente a interface visual. Para a estratégia da IA, tentativas[9][2] (matriz int) codifica heurísticas de jogo com coordenadas pré-calculadas. Nos algoritmos de análise, o acesso sequencial aos elementos via índices permite verificações eficientes de padrões vencedores.

➔ **Organização e função das funções criadas**

A arquitetura segue separação de concerns através de 13 funções especializadas:

- **Gestão de estado:** inicializarTabuleiro(), fazerJogada(), posicaoValida(), verificarVencedor(), tabuleiroCheio()
- **Interface:** exibirTabuleiro(), obterJogadaHumana(), exibirMenu(), exibirVitoria(), exibirEmpate()
- **Inteligência Artificial:** podeVencer(), obterJogadaIA()
- **Arranjo:** main()

Estruturas de Dados:

➔ **Vetores Principais:**

1. tabuleiro[3][3] - Matriz do Estado do Jogo

- Tipo: char (caractere)
- Dimensão: 3x3 (9 elementos)
- Propósito: Representa o estado atual do tabuleiro do jogo da velha
- Valores possíveis:
  - ' ' (espaço): posição vazia
  - 'X': jogada do jogador humano
  - 'O': jogada da IA/computador
- Acesso: Coordenadas [linha][coluna] onde linha e coluna variam de 0 a 2
- Importância: É a estrutura de dados central que define todo o estado do jogo

2. tentativas[9][2] - Vetor de Estratégia da IA

- Tipo: int (inteiro)
- Dimensão: 9x2 (9 pares de coordenadas)
- Propósito: Codifica a estratégia de prioridade da IA para seleção de jogadas
- Sequência de prioridade:
  - {1,1}: Centro (maior prioridade - posição mais vantajosa)
  - {0,0}, {0,2}, {2,0}, {2,2}: Cantos (segunda prioridade)
  - {0,1}, {1,0}, {1,2}, {2,1}: Bordas (menor prioridade)
- Funcionamento: A IA percorre este vetor em ordem até encontrar uma posição válida

➔ **VARIÁVEIS DE CONTROLE PRINCIPAIS:**

3. Variáveis de Estado do Jogo

- jogadorAtual (char): Controla qual jogador tem a vez ('X' ou 'O')
- modoIA (int): Define o modo de operação (0 = dois jogadores, 1 = jogador vs IA)

- vencedor (char): Armazena o resultado da verificação ('X', 'O' ou ' ' para empate)

4. Variáveis de Coordenadas e Entrada

- linha, coluna (int): Armazenam temporariamente as coordenadas da jogada atual
- opcao (int): Armazena a seleção do usuário no menu principal

5. Variáveis de Iteração

- i, j (int): Contadores utilizados em loops para navegar pelas matrizes
- linhaVencedora, colunaVencedora (int): Parâmetros de saída da função podeVencer

Implementação e reflexão

➔ **Dificuldades Encontradas**

A implementação do projeto apresentou desafios na lógica de detecção de vitória, que exigiu verificações simultâneas em oito direções distintas (três horizontais, três verticais e duas diagonais). Outra complexidade foi o desenvolvimento da estratégia da IA, particularmente na implementação do mecanismo de simulação de jogadas que permite antecipar vitórias iminentes e bloquear adversários. A validação de entradas do usuário também demandou atenção para garantir estabilidade em cenários de entrada inválida.

➔ **Soluções Implementadas**

Para superar esses desafios, foi adotada uma abordagem algorítmica sistemática. O sistema de verificação de vitória foi otimizado para realizar verificações em tempo constante através de padrões predefinidos. A IA foi equipada com um sistema de priorização que combina estratégias ofensivas (busca de vitória) e defensivas (bloqueio), seguindo uma sequência lógica de decisões. Implementou-se ainda um sistema de validação em múltiplas camadas que previne erros de execução e fornece feedback contextual ao usuário.

➔ **Organização do Código**

A arquitetura do código segue princípios de separação de concerns, com divisão clara entre lógica de negócio (gestão do tabuleiro e regras do jogo), interface de usuário (entrada e saída de dados) e inteligência artificial (tomada de decisão autônoma). Essa organização facilitou o desenvolvimento incremental, permitindo testar e refinar cada componente isoladamente antes da integração final.

➔ **Conclusão**

O projeto demonstra a aplicação prática dos conceitos da Unidade 1, transformando componentes teóricos em um sistema funcional. A implementação da IA adiciona uma camada de complexidade apropriada que enriquece a experiência do usuário enquanto serve como demonstração tangível dos

algoritmos estudados. A estrutura modular adotada prova a viabilidade do desenvolvimento incremental para projetos de programação estruturada.

Respostas às Perguntas Orientadoras

➔ **Quais conceitos da Unidade 1 foram aplicados e onde?**

Todos os conceitos foram aplicados: variáveis para controle de estado, condicionais para validações e decisões, repetições para fluxo do jogo e análise do tabuleiro, funções para modularização, e vetores para representação de dados e estratégias.

➔ **Como a organização em funções facilita a manutenção do código?**

A divisão em funções especializadas permite modificações localizadas, teste individual de componentes e reutilização de código..

➔ **Quais foram os principais desafios na implementação das estruturas de repetição?**

O balanceamento entre eficiência e legibilidade, garantindo que os loops terminem corretamente em todos os cenários possíveis do jogo.

➔ **Como os vetores foram utilizados para resolver o problema proposto?**

Como estruturas fundamentais para modelagem do tabuleiro (tabuleiro[3][3]) e codificação de conhecimento estratégico (tentativas[9][2]).

➔ **Que melhorias poderiam ser implementadas nas próximas unidades?**

Persistência de histórico de partidas, tabuleiros de tamanho variável, diferentes níveis de dificuldade para a IA, e interface gráfica simples.