CIÊNCIAS DA COMPUTAÇÃO
TÉCNICAS E DESENVOLVIMENTO DE ALGORITMOS
PROF. WALACE SARTORI BONFIM
P1 - 2023.2

PROJETO DE TÉCNICAS E DESENVOLVIMENTO DE ALGORITMOS: JOGO DA VELHA

BEATRIZ RAMALHO
DANIEL FERREIRA
GUSTAVO HUDSON
JOSÉ GERALDO
PEDRO HENRIQUE PORTO

1. PARTICIPANTES

Beatriz da Costa Ramalho (
Daniel da Silva Ferreira (23203579)
Gustavo Hudson Marinho Dornelas (36159107)
José Geraldo Duarte de Oliveira (36211559)
Pedro Henrique Porto Ferreira (35926333)

2. REGRAS DO JOGO

As regras do jogo são as seguintes: o primeiro jogador que conseguir fazer uma sequência de símbolos iguais, seja o jogador “X” ou o jogador “O”, na vertical, horizontal ou na diagonal, vence. O adversário deve impedir o outro jogador de finalizar a sequência, enquanto pensa em uma forma estratégica de terminar a sua própria jogada.

A nossa aplicação permite ainda algumas funcionalidades, tais quais: jogar contra um colega ou contra o computador, visualizar o ranking de cada jogador, resetar o ranking, visualizar os créditos de criação do programa e finalizar a aplicação.

3. RESULTADOS

Ao iniciar o programa, um “Menu” é exibido oferecendo 5 opções: 1 - Iniciar jogo, 2 - Jogar sozinho, 3 - Exibir um ranking, 4 - Resetar o ranking, 5 - Créditos e 5 6 - Sair.

1. Iniciar o jogo: supõe-se que serão dois usuários jogando a aplicação. Queríamos guardar os dados de cada usuário (nome e pontuação) em uma struct, para resgatar essas informações a cada rodada, bem como a cada vez que o jogo fosse iniciado. Porém, encontramos dificuldades em aplicar essa mesma struct para o item 2, em que seria o jogadorXcomputador. Então precisamos restringir o usuário para apenas duas opções: “X” e “O”.

   O primeiro passo, tanto na opção multiplayer, como na opção contra o computador, é exibir o tabuleiro vazio, para que o primeiro jogador possa escolher o posicionamento. Para isso, criamos a função “criar_tabuleiro” (Imagem 1), que alocará memória suficiente para exibir e modificar a matriz, que é o tabuleiro, de acordo com o andamento do jogo.

```c
char** criar_tabuleiro(int tamanho) {
    char** tabuleiro = (char**)malloc(tamanho * sizeof(char*));
    for (int i = 0; i < tamanho; i++) {
        tabuleiro[i] = (char*)malloc(tamanho * sizeof(char));
        for (int j = 0; j < tamanho; j++) {
            tabuleiro[i][j] = ' ';
        }
    }
    return tabuleiro;
}
```

Imagem 1: função “criar_tabuleiro” aloca memória para a matriz (tabuleiro).

Tendo memória suficiente, criamos outra função para exibir o tabuleiro (Imagem 2), que estará vazio no primeiro momento, mas a cada rodada, será atualizada a partir da modificação em cada item específico da matriz.

```c
void exibir_tabuleiro(char** tabuleiro, int tamanho) {
    printf("\n  ");
    for (int i = 1; i <= tamanho; i++) {
        printf("%d ", i);
    }
    printf("\n");

    for (int i = 0; i < tamanho; i++) {
        printf("%d ", i + 1);
        for (int j = 0; j < tamanho; j++) {
            printf("%c ", tabuleiro[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

Imagem 2: função “exibir_tabuleiro”.

Em seguida, a aplicação segue perguntando a posição de cada jogada. A cada jogada, precisamos verificar se a posição está livre para ser marcada, se não estiver, o jogo deve avisar ao jogador e lhe dar uma outra chance para escolher um local correto.

Caso a jogada seja permitida, devemos verificar se a mesma configura uma vitória, através da função “verifica_vitoria” (Imagem 3), que percorrerá toda a matriz verificando se alguma linha, coluna ou diagonal está completamente preenchida por um símbolo só (X ou O).

```c
int verificar_vitoria(char** tabuleiro, int tamanho, char jogador) {
    for (int i = 0; i < tamanho; i++) {
        int linha = 1;
        int coluna = 1;
        for (int j = 0; j < tamanho; j++) {
            if (tabuleiro[i][j] != jogador) {
                linha = 0;
            }
            if (tabuleiro[j][i] != jogador) {
                coluna = 0;
            }
        }
        if (linha || coluna) {
            return 1;  // Jogador venceu
        }
    }

    int diagonal1 = 1;
    int diagonal2 = 1;
    for (int i = 0; i < tamanho; i++) {
        if (tabuleiro[i][i] != jogador) {
            diagonal1 = 0;
        }
        if (tabuleiro[i][tamanho - i - 1] != jogador) {
            diagonal2 = 0;
        }
    }

    if (diagonal1 || diagonal2) {
        return 1;  // Jogador venceu
    }

    return 0;  // Ningu�m venceu ainda
}
```

Imagem 3: função que percorre a matriz inteira para verificar a vitória.

Se sim, devemos exibir o tabuleiro conforme foi preenchido (para que os jogadores possam confirmar o placar), atualizar o ranking atual do programa, exibir o ranking atual, salvar esse ranking em um arquivo txt para que a sequência permaneça entre várias rodadas e entre várias inicializações do programa. Essa última ação é realizada a partir da função “salvar_ranking_em_arquivo” (Imagem 4), que, juntamente com a função para carregar este ranking de volta para a aplicação, foram as etapas mais difíceis do projeto. Como não tínhamos conhecimento necessário para tal, pesquisamos e tentamos utilizar o “fprintf” e “fscanf”, mas apenas essas ações não tiveram o efeito desejado. Procuramos auxílio com o professor, que nos mostrou o modo **w+**, que abre/cria um arquivo para leitura e escrita, e pode ser consultado posteriormente.

```c
void salvar_ranking_em_arquivo(Jogador ranking[], int num_jogadores) {
    FILE* arquivo = fopen("ranking.txt", "w+");
    fprintf(arquivo, "Ranking:\n");
    for (int i = 0; i < num_jogadores; i++) {
        fprintf(arquivo, "%s %d\n", ranking[i].nome, ranking[i].pontuacao);
    }
    fclose(arquivo);
}
```

Imagem 4: função para salvar placar em txt, implementando w+.

No caso da jogada não configurar uma vitória, verificamos se configura um empate através da função “verifica_empate”, a qual percorre a matriz

procurando por posições que estão vazias, se não houver nenhuma, então há empate.

Ao final, seja vitória ou empate, precisamos resetar o tabuleiro, para evitar que na próxima partida o mesmo seja exibido parcialmente preenchido. Para isso, utilizamos o comando “Free” para liberar a memória que está alocada pela matriz.

2. Jogar sozinho

Como dito anteriormente, tivemos dificuldades para implementar o bot, pois não sabíamos como registrar o usuário dele. Sendo assim, colocamos o jogador como sendo “X” e o computador como “O”.

A primeira jogada sempre será do jogador, e segue os mesmos passos narrados no item 1 descrito acima.

Para a jogada do computador, chamamos a função “jogada_bot” (Imagem 5), que nos retornará dois números aleatórios (entre os valores possíveis, de 1 a 3) indicando a posição da jogada. Após essas ações, as mesmas verificações realizadas no item 1 acima, são feitas aqui novamente.

```c
void jogada_bot(char** tabuleiro, int tamanho, char jogador) {
    int linha, coluna;

    printf("Jogador %c fazendo a jogada...\n", jogador);

    do {
        linha = rand() % tamanho;
        coluna = rand() % tamanho;
    } while (tabuleiro[linha][coluna] != ' ');

    tabuleiro[linha][coluna] = jogador;
}
```

Imagem 5: função que configura a jogada do bot.

3. Exibir um ranking:

Nessa opção, a função “carregar_ranking_do_arquivo” (Imagem 6) acessa o arquivo txt utilizando o modo **r** (também passado pelo professor), o qual abre o arquivo apenas para leitura de dados.

Se o arquivo informado não existir, retornará uma mensagem informando o erro. Caso contrário, sobrescreverá os dados do ranking com as informações armazenadas anteriormente.

```c
void carregar_ranking_do_arquivo(Jogador ranking[], int num_jogadores) {
    FILE* arquivo = fopen("ranking.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo de ranking.\n");
        return;
    }

    char buffer[100];
    fgets(buffer, sizeof(buffer), arquivo);

    for (int i = 0; i < num_jogadores; i++) {
        fscanf(arquivo, "%s %d", ranking[i].nome, &ranking[i].pontuacao);
    }

    fclose(arquivo);
}
```

Imagem 6: função para consultar o ranking armazenado.

4. APÊNDICE

a) Código-fonte:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

typedef struct {
    char nome[5];
    int pontuacao;
} Jogador;

//Exibe o tabuleiro
void exibir_tabuleiro(char** tabuleiro, int tamanho) {
    printf("\n  ");
    for (int i = 1; i <= tamanho; i++) {
        printf("%d ", i);
    }
    printf("\n");

    for (int i = 0; i < tamanho; i++) {
        printf("%d ", i + 1);
        for (int j = 0; j < tamanho; j++) {
            printf("%c ", tabuleiro[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

```c
//Utiliza condi��es para verificar a vit�ria
int verificar_vitoria(char** tabuleiro, int tamanho, char jogador) {
    for (int i = 0; i < tamanho; i++) {
        int linha = 1;
        int coluna = 1;
        for (int j = 0; j < tamanho; j++) {
            if (tabuleiro[i][j] != jogador) {
                linha = 0;
            }
            if (tabuleiro[j][i] != jogador) {
                coluna = 0;
            }
        }
        if (linha || coluna) {
            return 1;  // Jogador venceu
        }
    }

    int diagonal1 = 1;
    int diagonal2 = 1;
    for (int i = 0; i < tamanho; i++) {
        if (tabuleiro[i][i] != jogador) {
            diagonal1 = 0;
        }
        if (tabuleiro[i][tamanho - i - 1] != jogador) {
            diagonal2 = 0;
        }
    }

    if (diagonal1 || diagonal2) {
        return 1;  // Jogador venceu
    }

    return 0;  // Ningu�m venceu ainda
}

//Caso todas as casas estejam ocupadas e a vit�ria n�o aconte�a, considera empate
int verificar_empate(char** tabuleiro, int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        for (int j = 0; j < tamanho; j++) {
            if (tabuleiro[i][j] == ' ') {
```

```
                return 0;  // Ainda h espa os vazios, o jogo n o est
empatado
            }
        }
    }
      return 1;   // Todas as posi es est o ocupadas, o jogo est
empatado
}

//Ao ganhar, adiciona 1 ao ranking do jogador vencedor
void atualizar_ranking(Jogador ranking[], int num_jogadores, char
jogador) {
    for (int i = 0; i < num_jogadores; i++) {
        if (ranking[i].nome[0] == jogador) {
            ranking[i].pontuacao++;
            return;
        }
    }
}

//Controla as jogadas do bot, usando 33% de chance na vertical e 33% na
horizontal, apenas em casas vazias
void jogada_bot(char** tabuleiro, int tamanho, char jogador) {
    int linha, coluna;

    printf("Jogador %c fazendo a jogada...\n", jogador);

    do {
        linha = rand() % tamanho;
        coluna = rand() % tamanho;
    } while (tabuleiro[linha][coluna] != ' ');

    tabuleiro[linha][coluna] = jogador;
}

//Exibe o ranking
void exibir_ranking(Jogador ranking[], int num_jogadores) {
    printf("\nRanking:\n");
    for (int i = 0; i < num_jogadores; i++) {
        printf("%s: %d\n", ranking[i].nome, ranking[i].pontuacao);
    }
    printf("\n");
}
```

```c
//Salva o ranking em um arquivo "ranking.txt" externo
void salvar_ranking_em_arquivo(Jogador ranking[], int num_jogadores) {
    FILE* arquivo = fopen("ranking.txt", "w+");
    fprintf(arquivo, "Ranking:\n");
    for (int i = 0; i < num_jogadores; i++) {
                        fprintf(arquivo,   "%s   %d\n",   ranking[i].nome,
ranking[i].pontuacao);
    }
    fclose(arquivo);
}

//reseta o ranking para zero
void resetar_ranking(Jogador ranking[], int num_jogadores) {
    for (int i = 0; i < num_jogadores; i++) {
        ranking[i].pontuacao = 0;
    }

    FILE* arquivo = fopen("ranking.txt", "w");
    fprintf(arquivo, "Ranking:\n");
    for (int i = 0; i < num_jogadores; i++) {
                        fprintf(arquivo,   "%s   %d\n",   ranking[i].nome,
ranking[i].pontuacao);
    }
    fclose(arquivo);

      printf("---------\nRanking  resetado.\n----------\n"); //isso  foi
dif?cil de conseguir fazer
}


//Ao exibir o ranking, puxa as informa??es do "ranking.txt" e exibe no
"exibir_ranking"
void  carregar_ranking_do_arquivo(Jogador  ranking[],  int  num_jogadores)
{
    FILE* arquivo = fopen("ranking.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo de ranking.\n");
        return;
    }

    char buffer[100];
    fgets(buffer, sizeof(buffer), arquivo);
```

```
    for (int i = 0; i < num_jogadores; i++) {
                        fscanf(arquivo,   "%s   %d",   ranking[i].nome,
&ranking[i].pontuacao);
    }

    fclose(arquivo);
}

//Usa aloca��o de mem�ria para criar o tabuleiro
char** criar_tabuleiro(int tamanho) {
    char** tabuleiro = (char**)malloc(tamanho * sizeof(char*));
    for (int i = 0; i < tamanho; i++) {
        tabuleiro[i] = (char*)malloc(tamanho * sizeof(char));
        for (int j = 0; j < tamanho; j++) {
            tabuleiro[i][j] = ' ';
        }
    }
    return tabuleiro;
}

//Ao encerrar qualquer partida, reinicia o tabuleiro
void reiniciar_tabuleiro(char** tabuleiro, int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        for (int j = 0; j < tamanho; j++) {
            tabuleiro[i][j] = ' ';
        }
    }
}

//Ao reiniciar o tabuleiro, limpa a mem�ria alocada e libera as casas
void liberar_tabuleiro(char** tabuleiro, int tamanho) {
    for (int i = 0; i < tamanho; i++) {
        free(tabuleiro[i]);
    }
    free(tabuleiro);
}

//Fun��o principal de funcionamento do jogo, determinando os jogadores
e seus comandos
int main() {
    setlocale(0, "Portuguese");
    int tamanho = 3; // Tamanho do tabuleiro
```

```
    char** tabuleiro = criar_tabuleiro(tamanho);
    int linha, coluna;
    char jogador = 'X';

    Jogador ranking[2] = {{"X", 0}, {"O", 0}};
    int num_jogadores = 2;

    //Mostra as op��es do menu
    int opcao;
    do {
        printf("Menu:\n");
        printf("1. Iniciar jogo\n");
        printf("2. Jogar sozinho\n");
        printf("3. Exibir o ranking\n");
        printf("4. Resetar ranking\n");
        printf("5. Cr�ditos\n");
        printf("6. Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);

         //Usa um Switch case para escolher o menu, cada case determina
qual o c�digo que ir� entrar em funcionamento
        switch (opcao) {
            case 1:
                do {
                    exibir_tabuleiro(tabuleiro, tamanho);

                     printf("Jogador %c, informe a linha (1-%d) e coluna
(1-%d) separadas por espaco: ", jogador, tamanho, tamanho);
                    scanf("%d %d", &linha, &coluna);

                      if (linha >= 1 && linha <= tamanho && coluna >= 1
&& coluna <= tamanho && tabuleiro[linha - 1][coluna - 1] == ' ') {
                        tabuleiro[linha - 1][coluna - 1] = jogador;

                            if (verificar_vitoria(tabuleiro, tamanho,
jogador)) {
                            exibir_tabuleiro(tabuleiro, tamanho);
                            printf("Jogador %c venceu!\n", jogador);

                               atualizar_ranking(ranking, num_jogadores,
jogador);
                            exibir_ranking(ranking, num_jogadores);
```

```
                                    salvar_ranking_em_arquivo(ranking,
num_jogadores);

                                reiniciar_tabuleiro(tabuleiro, tamanho);
                                break;
                                     } else if (verificar_empate(tabuleiro,
tamanho)) {
                                exibir_tabuleiro(tabuleiro, tamanho);
                                printf("O jogo terminou em empate!\n");

                                reiniciar_tabuleiro(tabuleiro, tamanho);
                                break;
                            }

                            jogador = (jogador == 'X') ? 'O' : 'X';
                        } else {
                            printf("Posicao invalida. Escolha outra.\n");
                        }
                    } while (1);
                    break;

                case 2:
                    do {
                        exibir_tabuleiro(tabuleiro, tamanho);

                         printf("Jogador %c, informe a linha (1-%d) e coluna
(1-%d) separadas por espaco: ", jogador, tamanho, tamanho);
                        scanf("%d %d", &linha, &coluna);

                          if (linha >= 1 && linha <= tamanho && coluna >= 1
&& coluna <= tamanho && tabuleiro[linha - 1][coluna - 1] == ' ') {
                            tabuleiro[linha - 1][coluna - 1] = jogador;

                                  if (verificar_vitoria(tabuleiro, tamanho,
jogador)) {
                                exibir_tabuleiro(tabuleiro, tamanho);
                                printf("Jogador %c venceu!\n", jogador);

                                   atualizar_ranking(ranking, num_jogadores,
jogador);
                                exibir_ranking(ranking, num_jogadores);
                                         salvar_ranking_em_arquivo(ranking,
num_jogadores);
```

```
                            reiniciar_tabuleiro(tabuleiro, tamanho);
                            break;
                                 } else if (verificar_empate(tabuleiro,
tamanho)) {
                            exibir_tabuleiro(tabuleiro, tamanho);
                            printf("O jogo terminou em empate!\n");

                            reiniciar_tabuleiro(tabuleiro, tamanho);
                            break;
                        }

                        jogador = (jogador == 'X') ? 'O' : 'X';
                    } else {
                        printf("Posicao invalida. Escolha outra.\n");
                    }

                    jogada_bot(tabuleiro, tamanho, jogador);

                     if (verificar_vitoria(tabuleiro, tamanho, jogador))
{
                        exibir_tabuleiro(tabuleiro, tamanho);
                        printf("Jogador %c venceu!\n", jogador);

                              atualizar_ranking(ranking, num_jogadores,
jogador);
                        exibir_ranking(ranking, num_jogadores);
                                     salvar_ranking_em_arquivo(ranking,
num_jogadores);

                        reiniciar_tabuleiro(tabuleiro, tamanho);
                        break;
                    } else if (verificar_empate(tabuleiro, tamanho)) {
                        exibir_tabuleiro(tabuleiro, tamanho);
                        printf("O jogo terminou em empate!\n");

                        reiniciar_tabuleiro(tabuleiro, tamanho);
                        break;
                    }

                    jogador = (jogador == 'X') ? 'O' : 'X';

                } while (1);
```

```
                break;

            case 3:
                carregar_ranking_do_arquivo(ranking, num_jogadores);
                exibir_ranking(ranking, num_jogadores);
                break;

            case 4:
            resetar_ranking(ranking, num_jogadores);
            break;

            case 5:
                            printf("----------------\nCr⌐ditos:\nGustavo
Hudson\nJos⌐ Geraldo\nBeatriz Ramalho\nDaniel Ferreira\nPedro Henrique
Porto\n-----------------\n");
                break;

            case 6:
                printf("-----\nSaindo do jogo.\n-----\n");
                break;
        }

    } while (opcao != 6);

    liberar_tabuleiro(tabuleiro, tamanho);
//Retornar 0 significa exito, enquanto outros valores ir⌐o significar
erro no funcionamento, por isso o uso do "return 0"
    return 0;
}
```