RELATÓRIO: JOGO DE DADOS

NOME E RGM DOS INTEGRANTES DA EQUIPE

- Gabriel Lemos Vilar- RGM 37429329
- Danton Martins – RGM 38393999
- Diego Braga Fong- RGM 36898821
- Francisco serafim da silva- RGM 39572684

1. INTRODUÇÃO

Este relatório apresenta o desenvolvimento de um jogo de dados simples, implementado na linguagem C, para a disciplina de Técnicas de Desenvolvimento de Algoritmos.

No jogo, dois jogadores participam de uma disputa onde cada um lança um dado de seis faces. O jogador que obtiver o maior número no dado vence a rodada. Caso os dois jogadores lancem o mesmo valor, a rodada é considerada um empate e não conta para nenhum jogador.

O jogo continua até que ambos decidam parar. No final, o programa exibe o placar final, indicando o número total de vitórias de cada jogador.

2. RESULTADOS

2.1 Funcionalidades do Jogo

O jogo de dados implementado apresenta as seguintes funcionalidades principais:

1. **Cadastro de jogadores:**
   - Solicita os nomes dos dois jogadores no início do programa e os armazena dinamicamente utilizando a função criarJogador().
2. **Lançamento de dados:**
   - A função jogarDado() gera números aleatórios entre 1 e 6, simulando o lançamento de um dado de seis faces.
3. **Comparação de resultados:**
   - A função compararDados() compara os valores lançados pelos dois jogadores e determina o vencedor da rodada. Em caso de empate, nenhuma vitória é contabilizada.
4. **Controle de jogabilidade:**
   - Após cada rodada, o programa pergunta se os jogadores desejam continuar jogando, permitindo múltiplas rodadas.
5. **Placar final:**
   - No término do jogo, o programa exibe o número total de vitórias de cada jogador com a função exibirResultados().
6. **Gerenciamento de memória:**

- A memória alocada para os dados dos jogadores é liberada ao final do programa usando a função liberarMemoria().

2.1 Funcionalidades do Jogo

Função criarJogador():

Esta função cria dinamicamente um jogador, inicializando o nome e o número de vitórias:

```
Jogador* criarJogador(const char* nome) {
  Jogador* jogador = (Jogador*) malloc(sizeof(Jogador));
  if (!jogador) {
    printf("Erro ao alocar memória!\n");
    exit(1);
  }
  strcpy(jogador->nome, nome);
  jogador->vitorias = 0;
  return jogador;
}
```

3. DIFICULDADES ENCONTRADAS E SOLUÇÕES IMPLEMENTADAS

1. Geração de números aleatórios

- **Dificuldade:** Inicialmente, os resultados do dado não eram suficientemente aleatórios.
- **Solução:** Foi utilizada a função srand(time(NULL)) para inicializar a semente do gerador de números aleatórios com base no tempo do sistema, garantindo maior variabilidade nos resultados.

2. Alocação dinâmica de memória

- **Dificuldade:** O armazenamento dos dados dos jogadores exigiu o uso de alocação dinâmica de memória. Isso gerou a necessidade de monitorar corretamente a alocação e a liberação da memória.
- **Solução:** Utilizamos a função malloc() para alocar memória e free() para liberá-la no final do programa, prevenindo vazamentos de memória.

3. Comparação dos resultados

- **Dificuldade:** A implementação da lógica para comparar os resultados dos dados e atualizar o placar precisou ser precisa para evitar inconsistências.
- **Solução:** A função compararDados() foi cuidadosamente projetada para determinar corretamente o vencedor ou identificar empates.

4. APÊNDICE: CÓDIGO FONTE COMPLETO

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <locale.h>



// Definição da estrutura do jogador

typedef struct {

  char nome[50];

  int vitorias;

} Jogador;



// Função para criar um jogador dinamicamente

Jogador* criarJogador(const char* nome) {

  Jogador* jogador = (Jogador*) malloc(sizeof(Jogador));

  if (!jogador) {

    printf("Erro ao alocar memória!\n");

    exit(1);

  }

  strcpy(jogador->nome, nome);  // acessa o campo 'nome' apontado pela estrutura 'jogador'

  jogador->vitorias = 0;

  return jogador;
```

```
}

// Função para jogar o dado (gera número aleatório entre 1 e 6)
int jogarDado() {
    return (rand() % 6) + 1;
}

// Função para comparar os valores dos dados e atualizar o número de vitórias
void compararDados(Jogador* jogador1, Jogador* jogador2, int dado1, int dado2) {
    printf("%s jogou: %d\n", jogador1->nome, dado1);
    printf("%s jogou: %d\n", jogador2->nome, dado2);

    if (dado1 > dado2) {
        printf("%s venceu esta rodada!\n\n", jogador1->nome);
        jogador1->vitorias++;
    } else if (dado2 > dado1) {
        printf("%s venceu esta rodada!\n\n", jogador2->nome);
        jogador2->vitorias++;
    } else {
        printf("Empate nesta rodada!\n\n");
    }
}

// Função para exibir os resultados finais
void exibirResultados(Jogador* jogador1, Jogador* jogador2) {
    printf("\nResultados finais:\n");
    printf("%s: %d vitória(s)\n", jogador1->nome, jogador1->vitorias);
```

```
    printf("%s: %d vitória(s)\n", jogador2->nome, jogador2->vitorias);
}

// Função para liberar a memória alocada para os jogadores
void liberarMemoria(Jogador* jogador1, Jogador* jogador2) {
    free(jogador1);
    free(jogador2);
}

int main() {
    setlocale(LC_ALL, "Portuguese");
    srand(time(NULL)); // Inicializa a semente para números aleatórios

    char nome1[50], nome2[50];
    printf("Digite o nome do jogador 1: ");
    scanf("%s", nome1);
    printf("Digite o nome do jogador 2: ");
    scanf("%s", nome2);

    // Criar jogadores dinamicamente
    Jogador* jogador1 = criarJogador(nome1);
    Jogador* jogador2 = criarJogador(nome2);

    char continuar;
    do {
        // Jogar os dados
        int dado1 = jogarDado();
```

```
        int dado2 = jogarDado();

        // Comparar resultados
        compararDados(jogador1, jogador2, dado1, dado2);

        // Perguntar se os jogadores desejam continuar
        printf("Desejam jogar novamente? (s/n): ");
        scanf(" %c", &continuar);

    } while (continuar == 's' || continuar == 'S');

    // Exibir resultados finais
    exibirResultados(jogador1, jogador2);

    // Liberar memória alocada
    liberarMemoria(jogador1, jogador2);

    return 0;
}
```