

Documentação do trabalho Prático

Nome: Arthur Bernardo Alves Costa

Matrícula:

2024099100

Curso: Programação e Desenvolvimento de Software

1. INTRODUÇÃO

2. ESTRUTURA DO PROGRAMA

3. ESTRUTURA DE DADOS

4. FUNÇÕES

a. "ler_dados":

b. "calcular_dano":

c. "lutar":

d. "Jogar":

e. "main":

5. COMO RODAR

6. APRENDIZADOS E CONSIDERAÇÕES FINAIS

1. INTRODUÇÃO

O programa feito simula uma batalha pokémon entre dois jogadores. Onde cada jogador possui um determinado número de pokémons e eles se enfrentam até que um dos jogadores não tenha mais pokémons "vivos" naquela partida.

2. ESTRUTURA DO PROGRAMA

O programa foi estruturado utilizando várias funções para desempenhar um papel específico durante a execução do programa. Veja:

- Inclusão das bibliotecas:
 - **"stdlib.h"**: Biblioteca utilizada para alocação de memória para as variáveis estruturadas Pokémon.
 - **"string.h"**: Biblioteca utilizada para manipulação das strings que surgem durante a execução.
 - **"stdio.h"**: Biblioteca utilizada para pegar as entradas e mostrar as saídas para o usuário.

- Declaração da variável estruturada Pokémon com os atributos (Nome, ataque, defesa, vida e tipo)
- Função “**ler_dados**”: Função responsável por ler os dados dos pokémons do arquivo de entrada e colocar nas variáveis que serão utilizadas durante o programa.
- Função “**calcular_dano**”: Função responsável por calcular o dano de ataque de cada pokémon levando em consideração o tipo do pokémon que está atacando e o tipo do pokémon que está recebendo o ataque.
- Função “**lutar**”: Função responsável por simular as lutas entre os pokémons, levando em consideração a vida de cada um e retornando que jogador venceu cada batalha.
- Função “**jogar**”: Função principal do jogo onde foi criado a lógica principal do programa, ou seja, declaração de variáveis, chamadas das outras funções e as saídas para o usuário.
- Função “**main**”: Função principal do programa, onde apenas chamamos a função “**jogar**”

3. ESTRUTURA DE DADOS

Pokemon foi o nome dado a variável estruturada para guardamos os atributos dos pokémons, então declaramos o nome como uma string de tamanho 20, o ataque, a defesa e a vida declaramos como inteiro e por fim uma string de tamanho 10 para o tipo de cada pokémon.

```
1 // Aqui definimos a variável estruturada Pokémon
2 typedef struct {
3     char nome[20]; // Variável para o nome do pokémon
4     int ataque; // Variável para os pontos de ataque do pokémon
5     int defesa; // Variável para os pontos de defesa do pokémon
6     int vida; // Variável para os pontos de vida do pokémon
7     char tipo[10]; // Variável para o tipo do pokémon
8 } Pokemon;
```

Em seguida na função jogar foi alocado dinamicamente memória (“**malloc**”) para dois vetores de pokémons, onde seu tamanho é de acordo com a entrada do arquivo que indica o número de pokémons de cada jogador.

4. FUNÇÕES

a. “ler_dados”:

```
1 // Função para ler dados de Pokémon de um arquivo
2 void ler_dados(FILE *arquivo, Pokemon pokemons[], int n) {
3     // Loop para ler os dados de cada Pokémon do arquivo
4     for(int i = 0; i < n; i++){
5         fscanf(arquivo, "%s %d %d %d %s", pokemons[i].nome, &pokemons[i].ataque, &pokemons[i].defesa, &pokemons[i].vida, pokemons[i].tipo);
6     }
7 }
```

- **Descrição:** Função lê os dados dos Pokémon de um arquivo e os armazena em um array de estruturas “**Pokemon**”
- **Parâmetros:**
 - “**FILE *arquivo**”: Ponteiro para o arquivo contendo os dados dos Pokémon.
 - “**Pokemon pokemons[]**”: Um vetor de variáveis estruturadas “**Pokemon**” para armazenar os dados lidos do arquivo.
 - “**int n**”: O número de Pokémon a serem lidos do arquivo.
- **Retorno:** A função não possui retorno, pois ela é do tipo “**void**”

b. “calcular_dano”:

```

1 float calcular_dano(Pokemon atacante, Pokemon defensor) {
2     float dano; // Usando float para precisão nos cálculos
3
4     if((strcmp(atacante.tipo, "eletrico") == 0 && strcmp(defensor.tipo, "agua") == 0) ||
5        (strcmp(atacante.tipo, "agua") == 0 && strcmp(defensor.tipo, "fogo") == 0) ||
6        (strcmp(atacante.tipo, "fogo") == 0 && strcmp(defensor.tipo, "gelo") == 0) ||
7        (strcmp(atacante.tipo, "gelo") == 0 && strcmp(defensor.tipo, "pedra") == 0) ||
8        (strcmp(atacante.tipo, "pedra") == 0 && strcmp(defensor.tipo, "eletrico") == 0)) {
9         dano = atacante.ataque * 1.2; // Aumenta o dano em 20% para relações de tipo favoráveis
10    } else if((strcmp(atacante.tipo, "eletrico") == 0 && strcmp(defensor.tipo, "pedra") == 0) ||
11              (strcmp(atacante.tipo, "agua") == 0 && strcmp(defensor.tipo, "eletrico") == 0) ||
12              (strcmp(atacante.tipo, "fogo") == 0 && strcmp(defensor.tipo, "agua") == 0) ||
13              (strcmp(atacante.tipo, "gelo") == 0 && strcmp(defensor.tipo, "fogo") == 0) ||
14              (strcmp(atacante.tipo, "pedra") == 0 && strcmp(defensor.tipo, "gelo") == 0)) {
15        dano = atacante.ataque * 0.8; // Reduz o dano em 20% para relações de tipo desfavoráveis
16    } else {
17        dano = atacante.ataque; // Dano padrão se os tipos forem iguais
18    }
19
20    return dano; // Converte o dano de volta para int antes de retornar
21 }

```

- **Descrição:** Esta função calcula o dano que um Pokémon que está atacando causa em outro Pokémon levando em consideração o tipo de cada um.
- **Parâmetros:**
 - **"Pokemon atacante":** O **"Pokemon"** que está atacando.
 - **"Pokemon defensor":** O **"Pokemon"** que está recebendo o ataque.
- **Retorno:** Retorna o dano calculado como inteiro para ser utilizado na função lutar.

c. “lutar”:

```
1 // Função para simular a luta entre dois pokemons
2 int lutar(Pokemon *p1, Pokemon *p2, int turno){
3     // Loop para a luta acontecer enquanto os dois pokemons tiverem vida > 0, ou seja vivos
4     while (p1->vida > 0 && p2->vida > 0) {
5
6         if(turno){ //se turno igual a 1 o jogador 1 começa atacando
7             // primeiro calculamos o dano do primeiro pokemon
8             float dano_p1 = calcular_dano(*p1, *p2);
9
10            // Se o dano do pokemon 1 for maior que a defesa do pokemon 2, entao o pokemon 2 perde vida igual a diferenca do ataque pela defesa
11            if (dano_p1 > p2->defesa) {
12                p2->vida = p2->vida - (dano_p1 - p2->defesa);
13            } else { // caso o contrario o pokemon 2 perde 1 de vida
14                p2->vida--;
15            }
16
17            // Verifica se p2 foi derrotado
18            if (p2->vida <= 0) {
19                printf("%s venceu %s\n", p1->nome, p2->nome);
20                return 1; // Retorna 1 indicando que pokemon 1 venceu
21            }
22            turno = 0;
23        } else { //se turno igual a zero jogador 2 que joga
24            // Ataque do p2 contra p1
25            float dano_p2 = calcular_dano(*p2, *p1);
26            // Se o dano do pokemon 2 for maior que a defesa do pokemon 1, entao o pokemon 1 perde vida igual a diferenca do ataque pela defesa
27            if (dano_p2 > p1->defesa) {
28                p1->vida = p1->vida - (dano_p2 - p1->defesa);
29            } else { // caso o contrario o pokemon 1 perde 1 de vida
30                p1->vida--;
31            }
32
33            // Verifica se p1 foi derrotado
34            if (p1->vida <= 0) {
35                printf("%s venceu %s\n", p2->nome, p1->nome);
36                return 2; // Retorna 2 indicando que pokemon 2 venceu
37            }
38            turno = 1;
39        }
40    }
41 }
```

- **Descrição:** Função responsável por simular uma batalha entre dois Pokémon. Funciona da seguinte maneira, primeiro temos um if para verificar o turno, onde se turno é igual a 1 o jogador começa atacando e se turno é igual a 0 o jogador 2 começa atacando, ela executa enquanto os pokémons tem vida maior que 0.
- **Parâmetros:**
 - “**Pokemon *p1**”: Ponteiro para o Pokémon do jogador 1.
 - “**Pokemon *p2**”: Ponteiro para o Pokémon do jogador 2.
 - “**Int turno**” : Inteiro para sabermos qual jogador vai começar atacando, dessa forma foi possível implementar a lógica dos turnos.
- **Retorno:** Retorna um valor inteiro indicando qual foi o resultado da batalha (1 para vitória do jogador 1 e 2 para vitória do jogador 2.)

d. “Jogar”:

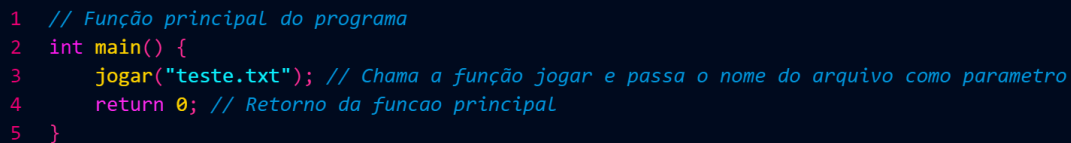
```
1 // Função principal para o jogo que pede como parametro um ponteiro para a posicao de memoria do arquivo
2 void jogar(char *arquivo){
3     //1 Etapa - Declaração de variáveis e leitura de dados
4
5     FILE *file = fopen(arquivo, "r"); // Abre o arquivo no modo de leitura
6     if(file == NULL){ // Aqui verificamos se houve algum erro na abertura do arquivo
7         perror("Erro ao abrir o arquivo.\n"); // Imprime a mensagem de erro indicando que não foi possível abrir o arquivo
8         return; // Retorna a função antes logo encerrando o programa
9     }
10
11     int numero_pokemons_p1, numero_pokemons_p2, n_vitorias_p2 = 0, n_vitorias_p1 = 0, vitoria_atual = 0; //Declaração das variáveis utilizadas durante o programa
12     fscanf(file, "%d %d", &numero_pokemons_p1, &numero_pokemons_p2); // Lê do arquivo o numero de pokemons do jogador 1 e 2 e atribui as respectivas variáveis
13
14     //verificação de limites de pokemons de cada jogador
15     if(numero_pokemons_p1 > 100 || numero_pokemons_p2 > 100){
16         perror("Erro: o numero maximo de pokemons para cada jogador e 100");
17         return;
18     }
19
20     // Alocamos dinamicamente a quantidade de memória necessária para armazenar os pokemons de cada jogador
21     Pokemon *P1 = malloc(numero_pokemons_p1 * sizeof(Pokemon));
22     //Verificação de erro de alocação
23     if(P1 == NULL){
24         printf("Erro ao alocar memoria para o jogador 1");
25         free(P1);
26         exit(1);
27     }
28     Pokemon *P2 = malloc(numero_pokemons_p2 * sizeof(Pokemon));
29     if(P2 == NULL){
30         printf("Erro ao alocar memoria para o jogador 2");
31         free(P2);
32         exit(1);
33     }
34     // Lê os dados dos pokemons de cada jogador do arquivo
35     ler_dados(file, P1, numero_pokemons_p1);
36     ler_dados(file, P2, numero_pokemons_p2);
37
38     fclose(file); // Agora que ja temos os dados que precisamos podemos fechar o arquivo
39
40     // 2 Etapa - Simulação das batalhas
41     //Loop para simular as batalhas entre os pokemons de cada jogador
42     int turno = 1;
43     while(n_vitorias_p2 < numero_pokemons_p1 && n_vitorias_p1 < numero_pokemons_p2){ // Para rodar enquanto numero de vitórias dos dois jogadores for menor que o numero de pokemons de cada jogador
44
45         vitoria_atual = lutar(&P1[n_vitorias_p2], &P2[n_vitorias_p1], turno); // Realizamos a batalha entre dois pokemons na posicao em que cada jogador venceu, pois se ele venceu e o n_vitorias aumenta ele ja passa o proximo pokemon
46
47         // Estrutura para atualizar as vitórias de cada jogador
48         if (vitoria_atual == 1) { // Se o retorno da função lutar foi 1 então o jogador 1 venceu uma batalha
49             n_vitorias_p1++; // Se o jogador numero 1 ganha aumenta o ndevitorias1
50             turno = 0; //variavel utilizada para controlar qual jogador vai atacar se o turno é zero quem começa atacando é o jogador 2
51         } else if (vitoria_atual == 2) { // Se não se o retorno da função lutar foi 2 então o jogador 2 venceu uma batalha
52             n_vitorias_p2++; // Se o jogador numero 2 ganha aumenta o ndevitorias2
53             turno = 1; // variavel utilizada para controlar qual jogador vai atacar se o turno é 1 quem começa é o jogador 1
54         }
55     }
56
57     // 3 Etapa - Verificar quem venceu a maioria das batalhas e assim quem venceu o jogo
58     if(n_vitorias_p2 > n_vitorias_p1){
59         printf("Jogador 2 venceu\n");
60     } else {
61         printf("Jogador 1 venceu\n");
62     }
63
64     // 4 Etapa - Mostrar os Pokemons que sobreviveram as lutas
65     printf("Pokemons Sobreviventes\n");
66
67     // Loop para mostrar os pokemons que sobreviveram do jogador 1
68     for(int y = 0; y < numero_pokemons_p1; y++){
69         if(P1[y].vida > 0){
70             printf("%s\n", P1[y].nome);
71         }
72     }
73     // Loop para mostrar os pokemons que sobreviveram do jogador 2
74     for(int y = 0; y < numero_pokemons_p2; y++){
75         if(P2[y].vida > 0){
76             printf("%s\n", P2[y].nome);
77         }
78     }
79
80     // 5 Etapa - Mostrar os pokemons que foram derrotados
81     printf("Pokemons Derrotados\n");
82
83     //Loop para mostrar os pokemons que foram derrotados do jogador 1
84     for(int z = 0; z < numero_pokemons_p1; z++){
85         if(P1[z].vida <= 0){
86             printf("%s\n", P1[z].nome);
87         }
88     }
89     //Loop para mostrar os pokemons que foram derrotados do jogador 2
90     for(int z = 0; z < numero_pokemons_p2; z++){
91         if(P2[z].vida <= 0){
92             printf("%s\n", P2[z].nome);
93         }
94     }
95
96     //Liberar a memória alocada para os Pokemons
97     free(P1);
98     free(P2);
99 }
```

- **Descrição:** Função principal responsável por coordenar o jogo, lendo os dados do arquivo, simulando batalhas, determinando o vencedor, os pokémons sobreviventes e os pokémons derrotados. Para simular as batalhas, criei um loop que executa enquanto um jogador tem menos vitórias do que o número de pokémons do outro. Dentro do loop, quando o retorno da função lutar é 1, o número de vitórias do jogador 1 aumenta, e a próxima luta é simulada com o vetor de pokémons do jogador 2 na

posição do número de vitórias e vice-versa. Dessa forma, foi possível implementar uma lógica que percorre os vetores à medida que cada jogador vence.

- **Parâmetros:**
 - “**char *arquivos**”: O nome do arquivo que contém os dados dos Pokémons que serão utilizados no programa.
- **Retorno:**
 - Não possui retorno.

e. “**main**”:



```
1 // Função principal do programa
2 int main() {
3     jogar("teste.txt"); // Chama a função jogar e passa o nome do arquivo como parametro
4     return 0; // Retorno da funcao principal
5 }
```

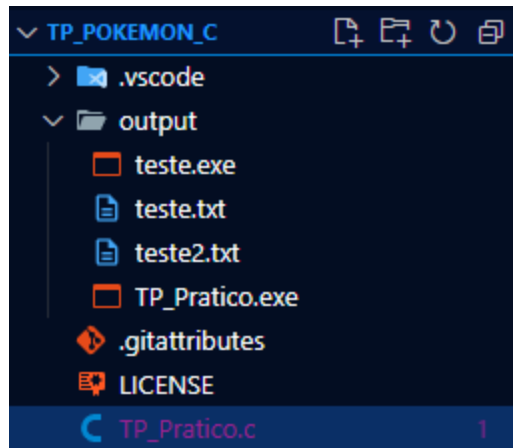
- **Descrição:** Função principal do programa, onde chamamos a função jogar e informamos o nome do arquivo com os dados.
- **Parâmetros:**
 - Não possui parâmetros.
- **Retorno:**
 - 0.

5. COMO RODAR

Primeiro é necessário um arquivo .txt com entradas padronizadas para serem lidas durante o programa e utilizadas nas variáveis. Em seguida deve-se colocar o arquivo de extensão “.txt” na mesma pasta que o executável gerado ao compilar o código c. Seguindo os passos descritos, basta apenas executar o código e realizar os testes.

Exemplo de teste no meu Computador:

- Diretório:



- txt de teste 1:



- saída:


```
Squirtle venceu Golem
Charmander venceu Squirtle
Vulpix venceu Charmander
Jogador 1 venceu
Pokemons Sobreviventes
Vulpix
Onix
Pokemons Derrotados
Squirtle Golem Charmander
```

6. APRENDIZADOS E CONSIDERAÇÕES FINAIS

De forma geral o trabalho teve sim alguns momentos em que agarrei para resolver algumas coisas, mas no fim foi bem tranquilo. Por exemplo:

- A leitura dos dados do arquivo eu demorei bastante para entender como faria o programar ler o arquivo e extrair perfeitamente cada dado e depois atribuir a suas respectivas variáveis e propriedades. Logo, foi preciso pesquisar mais sobre o assunto e enfim conseguir suprir essa parte do trabalho.
- Além disso houve erros de lógica para as batalhas, pois diversas vezes chegava em resultados errados e na minha cabeça estava tudo certo, mas novamente com calma foi possível chegar as saídas esperadas.
- Por fim, a última parte que tive bastante trabalho foi na lógica dos turnos que demorei para conseguir pensar em algo, porém depois de diversas horas pensando consegui fazer funcionar tudo certinho.

Em relação à lógica desenvolvida, foi criada a estrutura Pokémon com os atributos necessários conforme as instruções. Em seguida, foi alocado dinamicamente um vetor de Pokémon, levando em consideração a quantidade de Pokémon de cada jogador. As funções foram divididas para garantir a organização e implementadas de acordo com seus objetivos específicos. Vale ressaltar que o código está totalmente comentado para facilitar ao máximo a compreensão.