

Teoria dos Grafos e Computabilidade Implementação 1

Fabio Antônio, Lucas Alkmim, Pedro Marçal, Pedro Ribeiro

11 de Agosto, 2024

Conteúdo

1	Introdução	3
1.1	Objetivo	3
2	Estrutura do Código	3
2.1	Structs	3
2.2	Constantes	3
2.3	Fluxo de Execução	4
3	Observações	5

1 Introdução

1.1 Objetivo

O objetivo deste projeto é de a partir do universo da franquia de jogos Pokemon, simular um sistema de batalha entre dois treinadores e seus respectivos pokemons, considerando as relações entre seus tipos e variedade de tamanho de equipe.

2 Estrutura do Código

2.1 Structs

As struct utilizadas para a implementação do código foram as seguintes:

```
1 typedef struct Pokemon {
2     char nome[NOME_MAX_TAMANHO];
3     float ataque;
4     float defesa;
5     float vida;
6     char tipo[TIPO_MAX_TAMANHO];
7 } Pokemon;
8
9 typedef struct Treinador {
10     int quantidade;
11     int pokemons_mortos;
12     Pokemon *pokemon;
13 } Treinador;
```

Que são inicializados pelas funções:

```
1 void lerPokemons(Treinador *treinador, FILE *arquivo) {}
2
3 Treinador *fazerTreinador(int num_pokemon) {}
```

2.2 Constantes

As constantes para tamanho utilizadas foram definidas ,quando não identificadas no enunciado, por meio de uma escolha, como a TIPO_MAX_TAMANHO, que segue a lógica da maior string de tipo existente nesse contexto ser 8 (elétrico) +1 (caractere nulo no final da string).

```
1 #define TIPO_MAX_TAMANHO 9
2 #define NOME_MAX_TAMANHO 100
3 #define POKEMON_MAX_QUANTIDADE 100
```

2.3 Fluxo de Execução

Respeitando o seguinte fluxo de execução:

- Leitura de dados do arquivo
- Definição e atribuição dos treinadores e seus respectivos pokemons
- Simulação da batalha
- Mostragem dos resultados

```
1
2 FILE *arquivo = fopen("teste.txt", "r");
3 fscanf(arquivo, "%d %d", &num_pokemon_1, &num_pokemon_2);
4
5 Treinador *treinador_1 = fazerTreinador(num_pokemon_1);
6
7 lerPokemons(treinador_1, arquivo);
8
9 batalhar(treinador_1, treinador_2);
10
11 imprimirResultados(treinador_1, treinador_2);
```

vale ressaltar que o procedimento

```
1 void batalhar(Treinador treinador_1, Treinador treinador_2) {}
```

utiliza na sua implementação para o calculo de dano a relação entre defesa - ataque, já considerando o ataque ajustado pela relação de fraqueza de tipos, no seguinte método:

```
1 int calcularDano(Pokemon *atacando, Pokemon *alvo) {
2     float fator = calcularFraqueza(atacando->tipo, alvo->tipo);
3     float ataque = atacando->ataque * fator;
4     float diferenca = (ataque - alvo->defesa);
```

fator de fraqueza calculado a partir do método a seguir:

```
1 float calcularFraqueza(char *tipo_atacando, char *tipo_alvo) {
2     float fator = 1;
3     if (strcmp(tipo_atacando, "pedra") == 0) {
4         if (strcmp(tipo_alvo, "eletrico") == 0 || strcmp(tipo_alvo, "el
5             étrico") == 0)
6             fator = 1.2;
7         else if (strcmp(tipo_alvo, "gelo") == 0)
8             fator = 0.8;
```

3 Observações

Foi utilizado o atributo `pokemons_mortos` da struct `Pokemon` para facilitar a implementação de algumas funções ao decorrer do código, como no procedimento `void imprimirResultados()`, para evitar a criação de mais parâmetros para contar as posições, como no trecho a seguir:

```
1 printf("Pokemons derrotados:\n");
2 for (int i = 0; i < treinador_1->pokemons_mortos; i++)
3     printf("%s ", treinador_1->pokemon[i].nome);
```

ou para saber corretamente em qual pokemon de cada treinador esta sendo utilizado no vetor:

```
1 if (fatalidade == 1) {
2     printf("%s venceu %s\n", treinador_1->pokemon[treinador_1->
3     pokemons_mortos].nome, treinador_2->pokemon[treinador_2->
4     pokemons_mortos].nome);
5     treinador_2->pokemons_mortos++;
6 }
```