Programa e Desenvolvimento de Software I
Trabalho Prático

Laura de Magalhães Loiola Rezende

14 de Junho, 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:

```c
typedef struct Pokemon {
  char nome[NOME_MAX_TAMANHO];
  float ataque;
  float defesa;
  float vida;
  char tipo[TIPO_MAX_TAMANHO];
} Pokemon;

typedef struct Treinador {
  int quantidade;
  int pokemons_mortos;
  Pokemon *pokemon;
} Treinador;
```

Que são inicializados pelas funções:

```c
void lerPokemons(Treinador *treinador, FILE *arquivo) {}

Treinador *fazerTreinador(int num_pokemon) {}
```

2.2 Constantes

As contantes 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).

```c
#define TIPO_MAX_TAMANHO 9
#define NOME_MAX_TAMANHO 100
#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

```

  FILE *arquivo = fopen("teste.txt", "r");
  fscanf(arquivo, "%d %d", &num_pokemon_1, &num_pokemon_2);

  Treinador *treinador_1 = fazerTreinador(num_pokemon_1);

  lerPokemons(treinador_1, arquivo);

  batalhar(treinador_1, treinador_2);

  imprimirResultados(treinador_1, treinador_2);
```

vale ressaltar que o procedimento

```
  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:

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

fator de fraqueza calculado a partir do método a seguir:

```
float calcularFraqueza(char *tipo_atacando, char *tipo_alvo) {
  float fator = 1;
  if (strcmp(tipo_atacando, "pedra") == 0) {
    if (strcmp(tipo_alvo, "eletrico") == 0 || strcmp(tipo_alvo, "el
    étrico") == 0)
      fator = 1.2;
    else if (strcmp(tipo_alvo, "gelo") == 0)
      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:

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

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

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