Documentação do código

Rodrigo Aroeira

Sumário

1 Introdução

Este projeto desenvolve um simulador de batalhas Pokémon em linguagem C. O objetivo é criar um sistema onde dois treinadores, cada um com um grupo de Pokemons, se enfrentam até que um dos treinadores fique sem Pokémons vivos.

1.1 Mecanismo de Batalha

Durante a batalha, os Pokémons dos dois treinadores atacam alternadamente. O dano causado por cada ataque considera os valores de ataque e defesa dos Pokemons, além de um multiplicador baseado na efetividade dos tipos, simulando as vantagens de tipo. Por exemplo, um Pokémon do tipo fogo causará mais dano a um Pokemon do tipo gelo.

1.2 Entradas e Saídas

A entrada do programa é um arquivo de texto que especifica os detalhes dos treinadores e seus Pokemons. Este arquivo inclui a quantidade de Pokemons de cada treinador e os atributos de cada Pokemon, como nome, ataque, defesa, vida e tipo. O formato da entrada permite que o programa carregue dinamicamente os dados dos treinadores e Pokemons. O arquivo de texto é escolhido por um nome padrão, ou alterado pela linha de comando, antes da execução do programa.

Como saída, o programa imprime uma descrição detalhada da batalha, indicando quais Pokémons atacam, os resultados dos ataques e, eventualmente, quais Pokemons são derrotados. Ao final, o programa declara o vencedor da batalha e fornece um resumo dos Pokemons sobreviventes e derrotados.

1.3 Objetivo Final

O objetivo final do programa é determinar e declarar o vencedor da batalha Pokémon entre os dois treinadores, o programa oferece detalhes de como a batalha se desenrolou, listando os Pokémons que participaram, mostrando, primeiro, os sobreviventes, e, em seguida, os derrotados.

2 Estruturas

2.1 Pokemon

```
typedef struct
{
    char Nome[100];
    float Ataque;
    float Defesa;
    float Vida;
    char Tipo[100];
    char SuperEfetivo[100];
} Pokemon;
```

Essa estrutura contém os atributos pedidos na proposta, mais um atributo extra “SuperEfetivo”, que facilita o processo de checagem do multiplicador de dano nas batalhas.

2.2 Treinador

```
typedef struct
{
    Pokemon *Pokemons;
    int PokemonsTotal;
    int PokemonsVivos;
    bool vencedor;
} Treinador;
```

Essa estrutura não foi explicitamente pedida na proposta do trabalho, mas foi implementada para facilitar a lógica de batalha. Ela contém um ponteiro para um array de Pokemons que será alocado dinamicamente, o total de Pokemons que o treinador possui, o total de Pokemons vivos e um booleano que indica se o treinador é o vencedor da batalha.

3 Funções utilitárias

Essas funções foram implementadas para facilitar a manipulação das estruturas Pokemon e Treinador.

3.1 lerTreinadores

```
Treinador *lerTreinadores(const char nomeArq[])
{
  Treinador *treinadores = (Treinador *)malloc(2 * sizeof(Treinador));
  Treinador *treinador1 = &treinadores[0];
  Treinador *treinador2 = &treinadores[1];
  Pokemon *pokemonPtr;

  FILE *arquivoP = fopen(nomeArq, "r");

  if (arquivoP == NULL)
  {
    printf("Falha ao abrir o arquivo '%s', tente novamente.\n", nomeArq);
    exit(1);
  }

  fscanf(arquivoP, "%d %d", &treinador1->PokemonsTotal, &treinador2->PokemonsTotal);

  setupTreinador(treinador1, treinador1->PokemonsTotal);
  setupTreinador(treinador2, treinador2->PokemonsTotal);

  for (int i = 0; i < treinador1->PokemonsTotal; i++)
  {
    pokemonPtr = &treinador1->Pokemons[i];
    fscanf(arquivoP, "%s %f %f %f %s", pokemonPtr->Nome,
    &pokemonPtr->Ataque,
    &pokemonPtr->Defesa,
    &pokemonPtr->Vida,
    pokemonPtr->Tipo);

    getSuperEfetividade(pokemonPtr);
  }

  for (int i = 0; i < treinador2->PokemonsTotal; i++)
  {
    pokemonPtr = &treinador2->Pokemons[i];
    fscanf(arquivoP, "%s %f %f %f %s",
    pokemonPtr->Nome,
    &pokemonPtr->Ataque,
    &pokemonPtr->Defesa,
```

```
      &pokemonPtr->Vida,
      pokemonPtr->Tipo);

      getSuperEfetividade(pokemonPtr);
    }

    fclose(arquivoP);

    return treinadores;
}
```

Essa função lê um arquivo de texto e retorna um array de Treinadores, que é alocado dinamicamente para poder ser acessado fora da função.

3.2 setupTreinador

```
void setupTreinador(Treinador *treinador, int qtdPokemons)
{
    if (qtdPokemons <= 0 || qtdPokemons > 100)
    {
        printf("Quantidade de pokemons '%d' invalida, tente novamente.\n", qtdPokemons);
        exit(1);
    }

    treinadorPtr->PokemonsTotal = qtdPokemons;
    treinadorPtr->PokemonsVivos = qtdPokemons;
    treinadorPtr->Pokemons = (Pokemon *)malloc(qtdPokemons * sizeof(Pokemon));
    treinadorPtr->vencedor = false;
}
```

Nessa função, é feita a inicialização de um treinador, alocando dinamicamente um array de Pokemons, para que o programa não precise alocar memória desnecessariamente para um treinador que possua menos que a quantidade máxima de Pokemons permitida.

3.3 getSuperEfetividade

```
void getSuperEfetividade(Pokemon *pokemon)
{
    char *tipo = pokemon->Tipo;

    char *tipos[5] = {"elétrico", "água", "fogo", "gelo", "pedra"};
    char *contraTipos[5] = {"água", "fogo", "gelo", "pedra", "elétrico"};
    for (int i = 0; i < 5; i++)
    {
        if (strcmp(tipo, tipos[i]) == 0)
        {
            strcpy(pokemon->SuperEfetivo, contraTipos[i]);
            return;
        }
    }
}
```

Nesse trecho de código, são definidos dois arrays de strings, tipos e contraTipos, em que um elemento do array "tipos" é super efetivo contra o elemento do array "contraTipos" na mesma posição. Foi feita essa escolha para que a checagem fosse feita por um loop iterativo, ao invés de uma série de if's encadeados.

3.4 limparTreinadores

```
void limparTreinadores(Treinador treinadores[2])
{
    for (int i = 0; i < 2; i++)
    {
        free(treinadores[i].Pokemons);
    }
    free(treinadores);
}
```

A função "limparTreinadores" é responsável por liberar a memória alocada dinamicamente, para evitar vazamento de memória.

4 Batalha e Pós-Batalha

Nesse capítulo, serão descritas as funções que implementam a lógica de batalha do jogo. As funções se encontram no arquivo “Pokemon.c”

4.1 Batalha

```
void Batalha(Treinador treinadores[2])
{
  Treinador *treinador1 = &treinadores[0];
  Treinador *treinador2 = &treinadores[1];

  Pokemon *pokemonPtr1, *pokemonPtr2;

  int i = 0, j = 0;

  while (i < treinador1->PokemonsTotal && j < treinador2->PokemonsTotal)
  {
    pokemonPtr1 = &treinador1->Pokemons[i];
    pokemonPtr2 = &treinador2->Pokemons[j];

    Atacar(pokemonPtr1, pokemonPtr2);

    if (pokemonPtr2->Vida <= 0)
    {
      printf("%s venceu %s\n", pokemonPtr1->Nome, pokemonPtr2->Nome);
      j++;
      treinador2->PokemonsVivos--;
      continue;
    }

    Atacar(pokemonPtr2, pokemonPtr1);
    if (pokemonPtr1->Vida <= 0)
    {
      printf("%s venceu %s\n", pokemonPtr2->Nome, pokemonPtr2->Nome);
      i++;
      treinador1->PokemonsVivos--;
    }
  }
  setVencedor(treinadores);
  printOverview(treinadores);
}
```

Essa função é responsável por executar a lógica e loop de batalha entre os treinadores. Ela recebe um array de treinadores e acessa os Pokemons de cada treinador, atacando um ao outro até que um dos treinadores

fique sem Pokemons vivos.

4.2 Atacar e ataqueMultiplier

```
void Atacar(Pokemon *ataque, Pokemon *defesa)
{
  float multiplier = ataqueMultiplier(ataque, defesa);
  float dano = ataque->Ataque * multiplier - defesa->Defesa;
  if (dano <= 0)
  {
    dano = 1;
  }
  defesa->Vida -= dano;
}
```

Nessa função sem retorno, é calculado o dano do o Pokemon de ataque , junto de um multiplicador, que, ao final, é descontado da vida do Pokemon de defesa.

```
float ataqueMultiplier(Pokemon *ataque, Pokemon *defesa)
{
  float multiplier = 1;

  if (strcmp(ataque->SuperEfetivo, defesa->Tipo) == 0)
  {
    multiplier = 1.2;
  }
  else if (strcmp(defesa->SuperEfetivo, ataque->Tipo) == 0)
  {
    multiplier = 0.8;
  }
  return multiplier;
}
```

A função acima calcula o multiplicador de ataque do Pokemon, seguindo os critérios do roteiro. Primeiro, é checado caso o atributo "SuperEfetivo", do Pokemon de ataque, é igual ao atributo "Tipo"do Pokemon de defesa. Caso a checagem passe, o multiplicador é definido como 1.2. Caso falhe, é feito a checagem contrária à anterior, que define o multiplicador como 0.8. Caso nenhum dos casos passe, é utilizado o valor padrão de 1.

4.3 setVencedor

```
void setVencedor(Treinador treinadores[2])
{
  for (int i = 0; i < 2; i++)
  {
    if (treinadores[i].PokemonsVivos == 0)
    {
      treinadores[i].vencedor = false;
```

```
    }
    else
      treinadores[i].vencedor = true;
  }
}
```

Essa função tem, unicamente, o papel de facilitar a checagem de vencedor na execução de "printOverview"

4.4 printOverview

```
void printOverview(Treinador treinadores[2])
{
  Treinador *treinadorPtr;
  Pokemon *pokemonPtr;
  for (int i = 0; i < 2; i++)
  {
    treinadorPtr = &treinadores[i];
    if (treinadorPtr->vencedor)
    {
      printf("Jogador %d venceu\n", i + 1);
      printf("Pokemons sobreviventes:\n");
      for (int j = 0; j < treinadorPtr->PokemonsTotal; j++)
      {
        pokemonPtr = &treinadorPtr->Pokemons[j];
        if (pokemonPtr->Vida > 0)
        {
          printf("%s\n", pokemonPtr->Nome);
        }
      }
    }
  }

  printf("Pokemons derrotados:\n");
  for (int i = 0; i < 2; i++)
  {
    treinadorPtr = &treinadores[i];
    for (int j = 0; j < treinadorPtr->PokemonsTotal; j++)
    {
      pokemonPtr = &treinadorPtr->Pokemons[j];
      if (pokemonPtr->Vida <= 0)
      {
        printf("%s\n", pokemonPtr->Nome);
      }
    }
  }
}
```

Na função acima, o primeiro for-loop exerce a função de encontrar o treinador vencedor e iterar sobre os

Pokemons vivos e imprimir na tela. Já o segundo for-loop exerce a função de iterar sobre os Pokemons de ambos treinadores, imprimindo apenas os derrotados.

5 Funções Extras

5.1 help

```
void help(const char progname[]) {
    printf("Modo de uso: %s [OPTION]\n", progname);
    printf("Options:\n");
    printf("  --help                %-30s\n", "Mostra essa mensagem e sai do programa.");
    printf("  --print               %-30s\n", "Imprime os treinadores e sai do programa.");
    printf("  --custom [arquivo]    %-30s\n", "Le de um arquivo .txt com um nome da sua escolha.");
    printf("  --exemplo             %-30s\n", "Roda a batalha de exemplo. Funciona como --custom exemplo.txt");
    printf("  --instruct            %-30s\n", "Imprime as instrucoes de uso do programa e sai do programa.");

    exit(0);
}
```

Essa função é executada quando o programa é executado no terminal com "[NOME DO PROGRAMA] –help".

5.2 printTreinadores

```
int printTreinadores(Treinador treinadores[2])
{
    for (int i = 0; i < 2; i++)
    {
        printf("Treinador %d\n", i + 1);
        Treinador *treinador = &treinadores[i];
        for (int j = 0; j < treinador->PokemonsTotal; j++)
        {
            Pokemon *pokemon = &treinador->Pokemons[j];
            puts("--------------------");
            printf("Nome: %s\n", pokemon->Nome);
            printf("Ataque: %.2f\n", pokemon->Ataque);
            printf("Defesa: %.2f\n", pokemon->Defesa);
            printf("Vida: %.2f\n", pokemon->Vida);
            printf("Tipo: %s\n", pokemon->Tipo);
            printf("Super Efetivo contra: %s\n", pokemon->SuperEfetivo);
            puts("--------------------");
        }
    }
}
```

Essa função é responsável por imprimir os treinadores e seus respectivos Pokemons. Pode ser usada para fins de debug ou para mostrar os Pokemons dos treinadores antes de uma batalha. Será acessada somente pela interface de linha de comando.

5.3 printInstruct

```
void printInstruct(const char nomeArquivo[])
{
    setlocale(LC_ALL, ""); // Configuracao para suportar UTF-8

    FILE *arquivo = fopen(nomeArquivo, "r");
    if (arquivo == NULL) {
        perror("Erro ao abrir o arquivo");
        exit(1);
    }

    char buffer[1024];
    while (fgets(buffer, sizeof(buffer), arquivo) != NULL) { // Ler linha por linha
        printf("%s", buffer);
    }

    fclose(arquivo);
}
```