DOCUMENTAÇÃO - TRABALHO PRÁTICO:
BATALHA POKÉMON

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Discente: Gabriel Filipe Martins de Barros.
Docente: Dr. Adriano Alonso Veloso.
Turma: TW
Disciplina: Programação e Desenvolvimento de Software 1.
Data: 25/01/2025

1. VISÃO GERAL

O objetivo deste projeto é resolver um problema de programação envolvendo a temática Pokémon. Neste programa, criado apenas com a linguagem C, é implementado um sistema de batalha entre os Pokémon de dois jogadores em uma lógica de turnos, levando em conta os atributos ataque, defesa, vida e tipo dos Pokémon, sendo que há relações de vantagem ou desvantagem a depender dos tipos do atacante e do defensor.

O *software* carrega os dados de um arquivo, aplica as regras de batalha e exibe o resultado.

2. ESTRUTURA

O programa foi dividido em vários arquivos, a fim de aumentar a organização e a legibilidade do código.

- Arquivo “Pokemon.h”:

  Define o tipo Pokemon através de uma *struct*, a qual possui os seguintes atributos:

- *nome*: Nome do Pokémon, tipo *char**;
- *tipo*: Tipo do Pokémon, que pode ser agua, fogo, eletrico, gelo ou pedra, também *char**;
- *ataque*: Quantidade de pontos de ataque do Pokémon, tipo *float*;
- *defesa*: Quantidade de pontos de defesa do Pokémon, tipo *float*;
- *vida*: Quantidade de pontos de vida do Pokémon, também do tipo *float*.

- Arquivo “Jogador.h”:

  Define o tipo Jogador também por uma *struct*, que possui os seguintes atributos:

- *quantPokemon*: Quantidade de Pokémon do jogador, tipo *int*;
- *pokemon*: Ponteiro que recebe o(s) Pokémon do jogador, tipo *Pokemon**.

- Arquivo “Batalha.h”:

  Aqui está a lógica central das batalhas. Contém as seguintes funções:

- *int efetividade( )*: Verifica se há uma relação entre os tipos dos Pokémon e retorna 1 caso o ataque seja super efetivo, 2 caso não efetivo e 0 se não há uma relação. Nesse sentido, a função recebe como parâmetros os ponteiros do tipo char *tipoAtacante* e *tipoDefensor*, para aplicar a lógica de efetividade, que segue as regras da seguinte tabela:

Atacante / **Defensor**	**agua**	**fogo**	**eletrico**	**gelo**	**pedra**
agua	sem efeito	fraco	forte	sem efeito	sem efeito
fogo	forte	sem efeito	sem efeito	fraco	sem efeito
eletrico	fraco	sem efeito	sem efeito	sem efeito	forte
gelo	sem efeito	forte	sem efeito	sem efeito	fraco
pedra	sem efeito	sem efeito	fraco	forte	sem efeito

- *float calcularDano( )*: Recebe os Pokémon *atacante* e *defensor*, além do *efeito* de ataque como parâmetros, então calcula a quantidade de dano causada pelo atacante, levando em consideração a *efetividade( )*. O cálculo de dano acontece ao multiplicar os pontos de ataque e o efeito, que pode variar 20% para mais ou para menos, a depender se o ataque é efetivo ou não. Então, desse valor são subtraídos os pontos de defesa do Pokémon que está recebendo o ataque, o que gera a variável *dano*. Por fim, essa variável é retornada, sendo 1.0 a quantidade de dano mínima.

- *void atacar( )*: Realiza um turno de ataque entre dois Pokémon, atualizando a vida do defensor de acordo com o valor retornado de *calcularDano( )*. Para isso, em seus parâmetros são esperados os dois Pokémon e seus números indexados, a fim de encontrar as criaturas em seus respectivos *arrays* — ou melhor, times.

- *int contarPokemonVivos( )*: Através de uma estrutura de repetição que itera sobre o *array* de Pokémon do jogador, esta função conta quantas criaturas ainda possuem mais de zero pontos de vida no time do jogador. Recebe apenas o jogador em que se quer realizar tal procedimento.

- *void realizarTurno( )*: Realiza turnos de ataque entre dois jogadores ao chamar a função *atacar( )*, sempre verificando se o defensor ainda tem pontos de vida positivos. Caso o defensor tenha sido derrotado, é a vez do próximo Pokémon do adversário atacar (se ele ainda

tiver algum), já que a batalha é por turnos. Além disso, seus parâmetros são os mesmos da função *atacar( )*.

- *void definirVencedor( )*: Recebe os dois jogadores como parâmetro e executa os turnos entre seus Pokémon até que um dos contestantes não tenha mais nenhuma criatura em seu time, através de uma estrutura de repetição, exibindo o resultado final: qual jogador foi o vencedor.

- *void mostrarSobreviventes( )* e *void mostrarDerotados( )*: Essas funções recebem um jogador e mostram os nomes de seus Pokémon com vida maior que zero e menor ou igual a zero, respectivamente, ou seja, exibem os Pokémon sobreviventes e os que foram derrotados no time.

- Arquivo “main.c”:

Este é o arquivo principal do projeto. Suas responsabilidades são:

- Carregar os dados a partir do arquivo “dados.txt” e armazená-los em suas respectivas variáveis;
- Inicializar os jogadores e seus Pokémon;
- Executar os processos de batalha entre os jogadores usando as funções de “Batalha.h”;
- Alocar espaços na memória e liberá-los quando não forem mais necessários.

- Arquivo “dados.txt”

Armazena, em texto, todos os dados necessários para o processo da batalha. Sua lógica de funcionamento será melhor explorada no próximo capítulo.

3. PROCESSO DE EXECUÇÃO

O programa é executado pela classe “main.c”, que segue os seguintes passos:

- *Leitura de dados*: Todos os atributos necessários para a batalha são lidos do arquivo “dados.txt”, no qual a primeira linha especifica a quantidade de Pokémon no time de cada jogador e as linhas seguintes contêm os atributos de cada Pokémon, seguindo a ordem: nome, ataque, defesa, vida e tipo;

- *Inicialização dos jogadores*: Os Pokémon são alocados dinamicamente e associados a seus respectivos jogadores através de uma estrutura *for*;

- *Simulação e resultado da batalha*: Turnos alternados são executados ao chamar a função *definirVencedor( )* de “Batalha.h”, que processa todo o contexto da batalha discutido na seção anterior, e exibe os resultados dos duelos até que todos os Pokémon de um dos

jogadores sejam derrotados. Por último, o jogador vencedor é mostrado, além dos Pokémon sobreviventes e derrotados dos dois times.

Por fim, com o objetivo de evitar vazamentos, a memória alocada é liberada quando os atributos não são mais necessários.

4. EXEMPLOS DE ENTRADA E SAÍDA DE DADOS

- Entrada:

```
3 2
Squirtle 10 15 15 agua
Vulpix 15 15 15 fogo
Onix 5 20 20 pedra
Golem 20 5 10 pedra
Charmander 20 15 12 fogo
```

O jogador 1 possui os Pokémon Squirtle, Vulpix e Onix, e o jogador 2 possui o Golem e o Charmander.

- Saída:

```
Squirtle venceu Golem.
Charmander venceu Squirtle.
Vulpix venceu Charmander.
Jogador 1 venceu!
Pokemon sobreviventes:
Vulpix
Onix
Pokemon derrotados:
Squirtle
Golem
Charmander
```