UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

TRABALHO PRÁTICO

BATALHA POKEMON

Anny Caroline Almeida Marcelino

Julho 2024

Sumário

1. Introdução..... 3
2. Descrição do Algoritmo e Procedimentos Utilizados..... 3
Tipos de Variáveis e Estruturas de Dados..... 3
Funções..... 3
2.1 Diagrama.....4
2.2 Processo.....4
2.3 Execução..... 5
3. Testes e Erros..... 6
3.1 Processo de Criação do Algoritmo.....6
3.1.1 Exemplos de Erros Comuns e Correções.....7
3.2 Testes Realizados..... 8
4. Conclusão.....9

1. Introdução

O objetivo deste Trabalho Prático é desenvolver um programa em C que simule batalhas de Pokémons entre dois jogadores. Em que cada jogador possui um conjunto Pokémons que batalham entre si de forma alternada.

A quantidade e os dados dos Pokémons são lidos de um arquivo nomeado input.txt, sendo a quantidade no máximo 100 e no mínimo 1. O duelo entre os jogadores ocorre em turnos, em que cada jogador ataca uma vez, no qual o pokémon atacante pode receber um bônus de ataque em relação ao seu adversário. A batalha acontece até que um dos jogadores não tenha nenhum pokémon com vida.

2. Descrição do Algoritmo e Procedimentos Utilizados

Tipos de Variáveis e Estruturas de Dados

Estrutura Pokémon (*`struct pokemon`*): Define os atributos de cada Pokémon, incluindo nome, ataque, defesa, vida e tipo.

Constantes: Definem valores constantes como o número máximo de caracteres (`MAX`) para strings e o número de jogadores (`PLAYERS`).

Variáveis de Controle:

`qntd_pokemon[PLAYERS]`: Array que armazena a quantidade de Pokémon de cada jogador.

`total`: Variável que armazena o total de Pokémons no jogo somando as quantidades de ambos os jogadores.

`player_turno`: Variável que indica o turno atual da batalha (jogador 1 ou jogador 2).

`p1` e ***`p2`***: Índices que controlam qual Pokémon de cada jogador está em batalha durante a simulação.

Funções

***`int sobreviventes`*:**

Função que percorre o array de Pokémons do jogador, conta quantos Pokémons ainda estão vivos e retorna a quantidade de Pokémons sobreviventes.

`void statusPokemons:`

Procedimento que imprime os nomes dos Pokémons sobreviventes e derrotados de ambos os jogadores.

`float bonusAtaque:`

Função que retorna o bônus de ataque com base nos tipos dos Pokémons atacante e oponente, dependendo da relação de força (+20% de dano) e fraqueza (-20% de dano) entre os tipos.

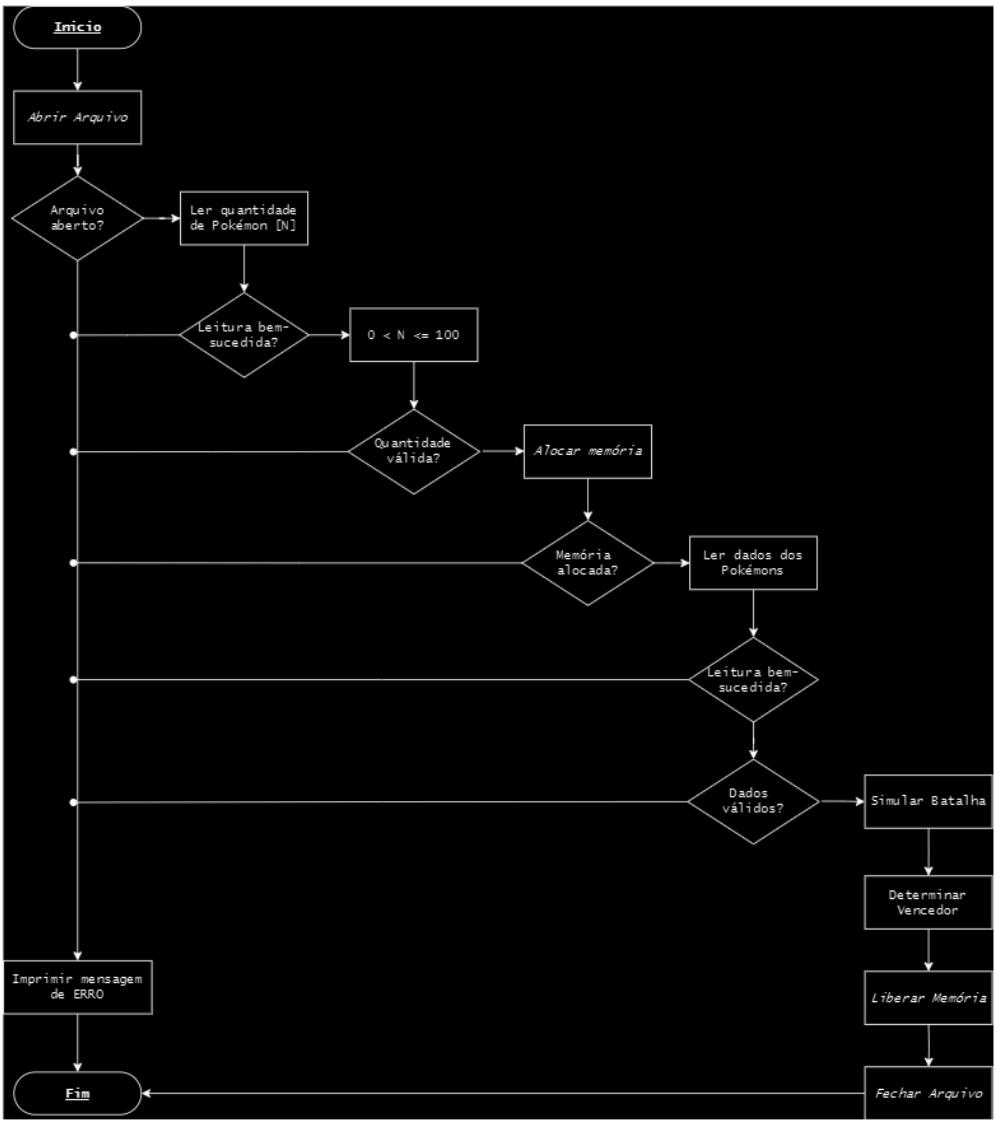
`void Batalha:`

Procedimento que simula uma batalha entre dois Pokémons. Calculando o dano dado ao oponente com base no ataque do Pokémon atacante, o bônus de ataque e a defesa do oponente.

`void imprime:`

Procedimento que imprime todas as informações de todos os pokémons de ambos os jogadores.

2.1 Diagrama



2.2 Processo

I. ***Leitura de Dados do Arquivo:*** Abre o arquivo input.txt e lê a quantidade de Pokémon para cada jogador.
Obs: o arquivo para leitura dos dados é pré definido como input.txt no mesmo diretório que o código.

II. ***Alocação Dinâmica de Memória:*** Após validar as quantidades de Pokémon, aloca se dinamicamente dois vetores de Pokémon (*player1* e *player2*).

III. **Leitura de dados II:** Lê os dados de cada Pokémon (nome, ataque, defesa, vida, tipo) e armazena nos vetores `player1` e `player2`. Isto é feito através de um loop, que enquanto não estiver no final do arquivo é feita a leitura e armazenagem dos dados, através de um ponteiro que aponta para a posição no vetor a ser preenchida e por uma variável que serve com índice e é incrementada a cada nova linha lida.

IV. ***Simulação de Batalha:*** A batalha é simulada em um loop que continua enquanto houver Pokémon sobreviventes de ambos os jogadores. Em cada iteração o turno varia e o jogador ataca com base no *player_turno*. O Pokémon atacante utiliza a função *Batalha* para calcular e aplicar o dano ao Pokémon adversário.
Obs.: a batalha sempre começa pelo jogador 1(player_turno = 0)

V. ***Verificação de Sobrevivência:*** Após cada ataque, verifica se o Pokémon adversário foi morto(vida <= 0), caso sim o índice do vetor de pokémons do jogador derrotado é incrementado.

VI. ***Determinação do Vencedor:*** A batalha continua até que todos os Pokémon de um jogador sejam derrotados. O jogador que ainda tiver Pokémon com vida restante é declarado vencedor.

VII. ***Liberação de Memória:*** O programa libera a memória alocada dinamicamente para os vetores de Pokémon. E fecha o arquivo de entrada.

2.3 Execução

O jogo começa com a leitura do arquivo dos números de Pokémon dos jogadores 1 e 2, em seguida, um vetor do tamanho referente a quantidade de Pokémons é alocado para ambos os jogadores. O programa lê e armazena os dados de cada Pokémon, que incluem nome, ataque, defesa, vida e tipo. A batalha então é simulada em turnos alternados entre os jogadores até que um dos jogadores não tenha mais Pokémon com vida. Durante a batalha o dano é calculado, o jogador restante é declarado vencedor e é mostrado os Pokémons sobreviventes e derrotados.

input.txt:

```
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
```

Output esperado:

```
JOGADOR 1  -  Quantidade Pokemons: 3
NOME                        | DEFESA | ATAQUE | VIDA  | TIPO
-------------------------------------------------------------
-
Squirtle                    | 15.0   | 10.0   | 15.0  | agua
Vulpix                      | 15.0   | 15.0   | 15.0  | fogo
Onix                        | 20.0   | 5.0    | 20.0  | pedra

JOGADOR 2  -  Quantidade Pokemons: 2
NOME                        | DEFESA | ATAQUE | VIDA  | TIPO
-------------------------------------------------------------
-
Golem                       | 5.0    | 20.0   | 10.0  | pedra
Charmander                  | 15.0   | 20.0   | 12.0  | fogo

Squirtle venceu Golem
Charmander venceu Squirtle
Vulpix venceu Charmander

Jogador 1 venceu!

Pokemons sobreviventes:
Vulpix
Onix

Pokemons derrotados:
Squirtle
Golem
Charmander
```

3. Testes e Erros

3.1 Processo de Criação do Algoritmo

Separei o processo de criação do código em etapas, começando pela abertura e leitura do arquivo. Após verificar que o arquivo foi aberto com sucesso, prossegui para a leitura e armazenamento dos dados. Optei por usar vetores diferentes para cada jogador, para facilitar a inserção dos dados utilizei um while para ler todas as linhas do arquivo, um único índice e um ponteiro vetores para ambos os vetores. Na etapa de simulação da batalha entre os Pokémons, através de um while que rodava até que um dos jogadores não tivesse mais nenhum Pokémon vivo, alternei os turnos dos jogadores e simulei os ataques através das funções `Batalha` e `bonusAtaque`. Inicialmente durante o desenvolvimento desta etapa, por

desatenção houve um erro em que o dano não era calculado corretamente devido ao atributo vida ser inteiro, resultando em cálculos incorretos para o bônus de ataque e dano que são do tipo float. Por fim, implementei as funções que apenas exibem as informações ao usuário: a função `imprime`, que exibe todas as informações de ambos os jogadores e a função `statusPokemon` que exibe quais Pokémon estão vivos e quais foram derrotados.

Durante o processo de criação do código desenvolvi algumas versões e as aprimorei até chegar na versão final, as principais alterações foram:

Leitura de dados dos Pokémons:

Inicialmente usava as funções `fprint` e `strtok` para ler e separar os dados, junto com duas variáveis de controle de quantidade de conteúdos lidos por linha e de Pokémons cadastrados. Além disso, necessitava de uma função adicional para converter e diferenciar os tipos de dados (nome, defesa, ataque, vida ou tipo), o que deixava o código confuso e lento.

A fim de aprimorar o código passei a utilizar `fscanf` e um ponteiro do mesmo tipo que o vetor. O `fscanf` facilita a leitura e a verificação da quantidade de Pokémons, e o ponteiro simplifica a inserção de dados nos vetores. Também utilizei uma única variável de controle que calcula a quantidade de pokémons inseridos e serve como índice para os vetores dos jogadores. Assim, tornando o código um pouco mais limpo e eficiente.

Alternância de turnos na Batalha:

Inicialmente os turnos se alteravam dentro de uma cascata de ifs, onde dependendo da vida de quem atacava e o turno atual, o novo turno era definido. Ao perceber que este método era desnecessariamente complicado decidi simplificar a alternância de turnos utilizando uma fórmula simples: `player_turno = 1 - player_turno`. Dessa forma, a variável `player_turno` varia entre 0 e 1, e torna o código mais claro e simples.

3.1.1 Exemplos de Erros Comuns e Correções

- **Erro:** Leitura incorreta dos dados do arquivo
  - **Correção:** Revisão da função de leitura para garantir que todos os dados fossem lidos e convertidos corretamente.

    Optei pelo uso `fscanf` pela razão de que a função retorna a quantidade de conteúdos lidos do arquivo, permitindo assim verificar possíveis erros.

- **Erro:** Problemas ao manipular os vetores de Pokémon.

- **Correção:** Revisão do código para garantir que os índices fossem atualizados corretamente após a troca de turno dos jogadores e garantir que os acessos aos vetores estivessem dentro dos limites válidos.

  No loop que simula o duelo, a cada pokémon morto o índice de pokémons do jogador derrotado é incrementado, a fim de garantir que os índices destes vetores não ultrapassem os limites mínimos e máximos utilizei a função sobreviventes como condição do loop, visto que o máximo e mínimo dela equivale ao dos índices.

➢ **Erro:** Problemas no cálculo de dano.

- **Correção:** Refatoração da função *bonusAtaque* a partir da simplificação da comparação das strings de *tipo*, o que consequentemente permitiu bom funcionamento da função *Batalha.*

3.2 Testes Realizados

Foram realizados testes abrangentes, como:

 Testes de Validação de Entrada: Verificação da formatação dos dados no arquivo, por simulações de arquivos vazios ou com formatos inválidos.

- Verifica se a quantidade de pokémons informada nas primeiras linhas é equivalente a quantidade de pokémons cadastrados.

input.txt	**Output Esperado**
1 1 Golduck 11 18 13 agua	Número incorreto de pokémons informado

- Verifica se a quantidade de Pokémons informada é no mínimo 1 e no máximo 100

input.txt	**Output Esperado**
0 101 Raichu 14 18 21 eletrico ...	Número invalido de pokemons para cadastro

- Verifica se há atributos negativos.

input.txt	Output Esperado
1 1 Geodude 11 -23 -22 pedra Rapidash 24 11 24 fogo	Os atributos do seu pokémon não podem ser negativos

- Verifica se a formatação dos dados no arquivo segue o padrão desejado, como no caso das informações dos pokémons, os 5 dados que devem estar separados por um espaço simples.

input.txt	Output Esperado
2 3 Golduck 11 18 13 agua Geodude 11 23 22 pedra Raichu 14 18 21 eletrico Rapidash 24 11 24 fogo Ciencia de Dados 2024	Erro ao ler do arquivo, os dados fornecidos não seguem o padrão esperado

- **Testes de Boas Práticas:**.
  - Verifica se o arquivo foi aberto, caso não exibe a mensagem "Erro ao abrir o arquivo"
  - Verifica se a memória foi alocada, caso não exibe a mensagem "Erro ao alocar memória"

4. Conclusão

A implementação deste programa possibilitou uma aplicação divertida dos conceitos de programação em C adquiridos durante o semestre, esta experiência me permitiu aprofundar meu conhecimentos sobre algoritmos, memória e recursividade.

Para futuras melhorias, considera-se a expansão do programa com recursos adicionais, como interface gráfica interativa para melhor visualização e interação durante as batalhas.