UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Trabalho Prático 1:
Pokémon, eu escolho você!

Alec Natanael de Sousa e Silva Reis
2023032240

Belo Horizonte
20/06/2023

Introdução

O Desafio proposto pelo trabalho foi a elaboração de um código, exclusivamente em linguagem C, de um simulador simples de batalhas pokémon de dois jogadores. Este programa deve ser capaz de ler um arquivo de entrada em formato de texto (.txt) contendo as seguintes informações:

- A primeira linha contendo respectivamente, a quantidade M de pokémons do jogador 1, e a quantidade N de pokémons do jogador 2;
- Após a primeira linha (que contém as quantidades de pokémons), as M linhas seguintes correspondem aos pokémons do jogador 1, e as N linhas seguintes a estas correspondem aos pokémons do jogador 2;
- Cada linha de um pokémon deve conter os seguintes atributos em sequência: nome, ataque, defesa, vida e tipo elementar.

Obtendo as informações do arquivo, o programa deve ser capaz de simular uma batalha de turnos alternados onde o pokémon do jogador 1 ataca o pokémon do jogador 2, e vice versa, passando para o próximo pokémon de cada jogador a cada derrota, até que um dos jogadores fique sem pokémons, o que caracteriza o fim do jogo e a vitória do jogador que ainda tem pelo menos um pokémon.

Ao simular a batalha, o programa deve imprimir na tela mensagens informando qual pokémon derrota qual em cada luta, e ao final, deverá imprimir mensagens informando o jogador vencedor, quais pokémons restaram, e quais pokémons foram derrotados (de ambos os jogadores).

Descrição do Algoritmo e Procedimentos utilizados

O programa elaborado para este trabalho foi escrito em dois arquivos: o arquivo base “main.c”, contendo o algoritmo principal de funcionamento do jogo; e um arquivo de biblioteca “pokemon.h”, contendo funções e estruturas essenciais para as mecânicas do jogo. Para o bom funcionamento do programa, além da biblioteca personalizada “pokemon.h”, também são utilizadas as seguintes bibliotecas:

- stdio.h
- stdlib.h
- stdbool.h
- math.h
- string.h

O programa também utiliza algumas constantes globais e um tipo de estrutura (ambos registrados no arquivo “pokemon.h”):

- “qtd_player1”: constante de tipo int que armazena um número M que representa a quantidade de pokémons do jogador 1;
- “qtd_player2”: constante de tipo int que armazena um número N que representa a quantidade de pokémons do jogador 2;
- “pokemon”: Tipo de estrutura projetado para armazenar todos os atributos de um pokémon, ou seja, ela contém as variáveis:
    - “nome[30]”: Um vetor de caracteres que armazena o nome do pokémon;
    - “ataque”: Uma variável tipo float que armazena o número correspondente a capacidade de ataque do pokémon;
    - “defesa”: Uma variável tipo float que armazena o número correspondente a capacidade de defesa do pokémon;
    - “vida”: Uma variável tipo float que armazena o número correspondente a vida máxima do pokémon;
    - “tipo[10]”: Um vetor de caracteres que armazena o tipo do pokémon;

Todas as funções do jogo são devidamente explicadas nos tópicos a seguir:

Função main()

Ao ser executado, a função principal do programa segue o seguinte algoritmo:

1. Variáveis internas de main():
    a. “fim_jogo”: variável booleana que serve de marcador para o fim de jogo (valor inicial = false);
    b. “fim_luta”: variável booleana que serve de marcador para o fim de uma luta entre dois pokémons (valor inicial = false);
    c. “vez”: variável booleana que serve para indicar a vez de quem jogar (valor inicial = true);
    d. “opcao1”: variável de tipo “int” que serve de contador para indicar qual pokémon o jogador 1 vai usar na próxima rodada (valor inicial = 1);
    e. “opcao2”: variável de tipo “int” que serve de contador para indicar qual pokémon o jogador 2 vai usar na próxima rodada (valor inicial = 1);
    f. “vencedor”: variável de tipo “int” que guarda o número do jogador vencedor ou 0 se ainda não há um vencedor (valor inicial = 0);

g. “p1”: estrutura de tipo “pokemon” que armazena os atributos do pokémon do jogador 1 utilizado em cada rodada;
h. “p2”: estrutura de tipo “pokemon” que armazena os atributos do pokémon do jogador 2 utilizado em cada rodada;
i. “entrada”: ponteiro do tipo “FILE” que aponta para o arquivo “entrada.txt”.

2. Abre o arquivo “entrada.txt” sob o ponteiro “entrada” no modo leitura;
3. Por meio da função “ler_quantidades()”, lê os dois números na primeira linha do arquivo de entrada, atribui o primeiro à “qtd_player1” representando a quantidade de pokémons do jogador 1, e o segundo à “qtd_player2” que representa a quantidade de pokémons do jogador 2;
4. Por meio de uma função específica para cada jogador (“pokemon_player1()” e “pokemon_player2 ()”) , lê em “entrada” as linhas relativas aos primeiros pokémons que cada um vai usar (obedecendo que as M primeiras linhas são do jogador 1 e as N últimas são do jogador 2), e guarda nas duas estruturas de tipo pokemon “p1” e “p2” (“p1” para o jogador 1 e “p2” para o jogador 2), representando o pokémon de cada jogador, guardando todos os atributos de cada pokémon;
5. Inicia-se um while loop “do jogo” que continua enquanto uma variável booleana “fim_jogo” não adquire o valor “true”;
6. Dentro do loop “do jogo” tem um while loop “de cada luta” que continua enquanto a variável booleana “fim_luta” não adquire o valor “true”. Sempre antes de entrar nesse loop, a variável “fim_luta” recebe o valor “false” por garantia.
7. Dentro do loop “de cada luta”, o programa primeiro verifica a variável booleana “vez” que indica a vez de cada jogador atacar (true para o jogador 1 / false para o jogador 2). Caso “true”, a estrutura “p2” recebe a estrutura retornada pela função “ataque()”, que nada mais é do que uma atualização da vida do pokémon após sofrer um ataque, em seguida é trocado o valor da variável “vez” para “false”. Caso “false”, “p1” recebe a atualização da função “ataque()” e a variável “vez” é trocada para “true”.
8. Em seguida, ainda dentro do loop “de cada luta”, verifica-se as condições de fim de luta, ou seja, se a vida de “p1” ou a vida de “p2” está menor ou igual a zero. Para cada caso segue-se praticamente o mesmo protocolo com pequenas diferenças:
    a. Imprime na tela uma mensagem: “[Nome do pokémon vencedor] venceu [Nome do pokémon derrotado]”;
    b. Incrementa +1 na variável “opcao1” (caso “p1” derrotado) ou “opcao2” (caso “p2” derrotado), que nada mais são do que contadores que indicam qual pokémon de cada jogador será utilizado na próxima batalha (é essencial como argumento das funções “pokemon_player1()” e “pokemon_player2 ()”);
    c. A variável “fim_luta” recebe o valor “true”;
    d. Verifica-se as condições de fim de jogo, ou seja, se “qtd_player1 - opcao1 < 0” (no caso em que “p1” é derrotado) ou “qtd_player2 - opcao2 < 0” (no caso em que “p2” é derrotado), tais condições indicam se acabaram os pokémons disponíveis do jogador 1 ou do jogador 2, para cada caso:
        i. Se satisfeita a condição: Variável “vencedor” recebe 1 (caso jogador 2 derrotado) ou 2 (caso jogador 1 derrotado). Variável “fim_jogo” recebe “true”;
        ii. Se não satisfeita a condição: Estrutura “p2” recebe o próximo pokémon do jogador 2 por meio da função “pokemon_player2()” (caso jogador 2 derrotado) ou Estrutura “p1” recebe o próximo pokémon do

jogador 1 por meio da função “pokemon_player1()” (caso jogador 1 derrotado);

9. Quando o loop “de cada luta” termina, ainda dentro do loop “do jogo”, o programa verifica se o valor da variável “vencedor” é igual a 1 (jogador 1 venceu) ou 2 (jogador 2 venceu). Caso esteja inalterada em 0, consequentemente, a variável “fim_jogo” continua “false”, passa sem interagir com o verificador e o loop “do jogo” se repete. Caso esteja como 1 ou 2:
    a. Imprime na tela a mensagem “Jogador 1 venceu!” ou “Jogador 2 venceu!”;
    b. Imprime a mensagem: “Pokemons sobreviventes: ” com quebra de linha ao final;
    c. Chama a função “imprime_restantes()” que vai imprimir na tela todos os pokémons que não foram derrotados (restantes do jogador 1);
    d. Imprime a mensagem: “Pokemons derrotados: ” com quebra de linha ao final;
    e. Chama a função “imprime_derrotados()” que vai imprimir todos os pokémons que foram derrotados durante a batalha (jogador 1 e jogador 2);
10. Fecha o arquivo “entrada” e encerra o programa;

Função ler_quantidades()

Essa função tipo void é responsável apenas por ler as quantidades de pokémons que cada jogador possui, localizadas na primeira linha do arquivo de entrada. Recebe como parâmetro o ponteiro do arquivo de entrada, e possui apenas um comando fscanf que põe as quantidades nas duas constantes globais tipo int: “qtd_player1” e “qtd_player2”.

Funções pokemon_player1() e pokemon_player2()

Essas funções são responsáveis por “pegar” as informações de um pokémon nas linhas seguintes do arquivo de entrada e retornar uma “struct pokémon” contendo os dados do pokémon escolhido. Ambas são muito semelhantes, só se diferenciam no fato que a “pokémon_player2” pula mais linhas antes de começar a ler (por que os pokémons do jogador 2 estão depois das M linhas dos pokémons do jogador 1).

Ambas as funções possuem três parâmetros:

- O ponteiro do arquivo de entrada (FILE* entrada);
- Uma struct pokémon chamada “player” que é usada como variável para guardar os dados do pokémon e depois ser retornada pela função. No caso deste código, as chamadas da função sempre passam como argumento as structs “p1” ou “p2”.
- Uma variável int “opcao”, que indica qual pokémon ler na ordem em que eles aparecem no arquivo.

E como variáveis internas:

- “linha”: tipo int, serve como contador indicando em qual linha o cursor está. Valor inicial = 0;
- “ch”: tipo char, é parte de um macete utilizado para pular linhas por onde passa o cursor;

Assim, o algoritmo de ambas as funções é o seguinte:

1. Utilização do comando rewind(entrada) para mover o cursor de leitura do arquivo de entrada para o seu início;
2. Fazer o cursor de leitura do arquivo pular linhas até que “linha” seja igual à “opcao”. Isso pode ser feito por meio de dois loops while aninhados, o interno pula caracteres

(lê com a função fgetc(entrada) armazenando em “ch”) até parar ao encontrar uma quebra de linha “\n”, e o externo repete o interno enquanto “linha” != “opcao”, incrementando +1 em “linha” cada vez que o loop interno termina. No caso de “pokemon_player2”, a condição é “linha” != “opcao” + “qtd_player1”, o que pula as linhas do jogador 1.

3. Ler a linha que corresponde ao pokémon escolhido por meio de um comando fscanf, atribuindo cada um dos itens às variáveis que compõem a struct “player”.
4. Retornar “player”.

Função ataque()

Esta função é utilizada para calcular as mecânicas de ataque de um pokémon a outro, retornando uma struct do pokemon atacado (a fim de atualizar a vida perdida após sofrer um ataque). Seus parâmetros são:

- Uma struct pokemon “atacante”, que traz as informações do pokemon que inflige o ataque;
- Uma struct pokemon “defensor”, que traz as informações do pokemon que sofre o ataque;

O algoritmo da função é o seguinte:

1. É estabelecida uma condição que depende do valor retornado pela função compara_tipos(), um inteiro que pode ser 0, 1 ou -1.
    - Se o valor retornado for 0, não há vantagem ou desvantagem decorrente dos tipos dos pokémons envolvidos, então o programa verifica se “atacante.ataque” é maior que “atacante.defesa” +1:
        - Se for maior: Subtrai de “defensor.vida” a diferença entre “atacante.ataque” e “defensor.defesa”;
        - Se for menor ou igual: Subtrai 1.0 de “defensor.vida”;
        - Obs: A necessidade de adicionar +1 na condição é para evitar que, se por um motivo a diferença entre “atacante.ataque” e “defensor.defesa” for menor que 1, o dano causado seja essa diferença, o que não faz sentido pelas regras do jogo.
    - Se o valor retornado for 1, então o tipo do atacante tem vantagem sobre o tipo do defensor. Então o programa soma a “atacante.ataque” 20% de seu valor e, a partir daí, repete os mesmos passos do caso 0;
    - Se o valor retornado for -1, então o tipo do atacante tem desvantagem sobre o tipo do defensor. Então o programa subtrai de “atacante.ataque” 20% de seu valor e, a partir daí, repete os mesmos passos do caso 0.
2. Por fim, o programa retorna a struct “defensor”.

Função compara_tipos()

Essencial para o funcionamento de ataque(), a função compara_tipos(), que têm os mesmos parâmetros de sua dependente, retorna um inteiro que pode ser 0 (sem vantagem ou desvantagem), 1 (vantagem do atacante) ou -1 (desvantagem do atacante).

Essa comparação dos tipos funciona por meio de uma série de condições encadeadas que possuem a mesma estrutura, só mudando os nomes comparados:

1. Uma variável interna int “op”, é declarada para guardar o retorno final do programa, ela já começa com o valor 0;

2. Uso do comando strcmp(atacante.tipo (ou defensor.tipo), “tipo a ser detectado”)==0, que basicamente detecta a string em “atacante.tipo” ou “defensor.tipo” como os tipos a serem comparados em determinada condição;
3. São 5 tipos no total (“eletrico”, “agua”, “fogo”, “gelo”, “pedra”). Para cada “atacante.tipo”, verifica-se se o “defensor.tipo” é forte ou fraco contra ele. Se for forte, “op” recebe 1, se for fraco, “op” recebe -1. Caso nenhum dos dois, nada acontece, e a variável “op” continua a valer 0.
4. Após rodar uma vez, a função retorna “op”.

Função imprime_restantes()

Após terminadas todas as batalhas, essa função sem retorno é chamada para imprimir na tela os pokémons que sobreviveram à batalha. Seus parâmetros são:

- O ponteiro do arquivo de entrada (FILE* entrada);
- Uma variável int “vencedor”, que recebe um número indicando qual jogador venceu o jogo;
- int “opcao”, guarda um número que indica qual pokémon do jogador vencedor derrotou o último pokémon do jogador perdedor.

Também há as seguintes variáveis internas:

- “linha”: tipo int, serve como contador indicando em qual linha o cursor está. Valor inicial = 0;
- “ch”: tipo char, é parte de um macete utilizado para pular linhas por onde passa o cursor;
- “char nome[30]”: Vetor de caracteres que armazena o nome de cada pokémon lido para depois imprimi-lo na tela.

O algoritmo da função funciona da seguinte maneira:

1. Usa o comando rewind() para voltar o cursor de leitura para o início do arquivo;
2. Estabelece uma condição para caso “vencedor == 1” (jogador 1 venceu) ou “vencedor == 2” (jogador 2 venceu);
3. Ambos os casos são muito semelhantes, só mudando a primeira parte:
    a. Caso jogador 1 venceu: É criado um loop while que permanece enquanto a diferença entre “qtd_player1” (constante global) e “opcao” for maior ou igual a zero (essa diferença +1 é exatamente a quantidade de pokémons sobreviventes). Dentro desse loop, há um mecanismo para fazer o cursor do leitor do arquivo “pular” linhas até a posição necessária, idêntico ao mecanismo usado na função “pokemon_player1()”.
    b. Caso jogador 2 venceu: É criado um loop while que permanece enquanto a diferença entre “qtd_player2” (constante global) e “opcao” for maior ou igual a zero (essa diferença +1 é exatamente a quantidade de pokémons sobreviventes). Dentro desse loop, há um mecanismo para fazer o cursor do leitor do arquivo “pular” linhas até a posição necessária, idêntico ao mecanismo usado na função “pokemon_player2()”.
4. Dentro do loop while e depois de ajustar a posição do cursor no arquivo, os dois casos seguem o mesmo procedimento;
5. Executa-se o comando fscanf para ler somente o nome do pokémon, guardando no vetor “nome”;
6. Executa-se o comando printf para imprimir “nome” na tela, pulando uma linha ao final.

7. Incrementa +1 na variável “opcao” preparando para passar para o próximo pokémon na próxima repetição do loop;
8. O algoritmo termina com o fim do loop em um dos casos.

Função imprime_derrotados()

Essa função é chamada após a “imprime_restantes()” para ler e imprimir na tela todos os pokémons derrotados, tanto do do jogador perdedor, quanto do jogador vencedor. Seus parâmetros são os mesmos de “imprime_restantes()” com as seguintes variáveis internas:

- “linha”: tipo int, serve como contador indicando em qual linha o cursor está. Valor inicial = 0;
- “ch”: tipo char, é parte de um macete utilizado para pular linhas por onde passa o cursor;
- “char nome[30]”: Vetor de caracteres que armazena o nome de cada pokémon lido para depois imprimi-lo na tela.
- “c”: tipo int, é um contador que será utilizado para indicar qual pokémon será impresso na tela (onde o contador “linha” deve parar a cada impressão). Valor inicial = 1.

O algoritmo da função é o seguinte:

1. Usa o comando rewind() para voltar o cursor de leitura para o início do arquivo;
2. Estabelece uma condição que se divide em dois casos: “vencedor == 2” e “vencedor == 1”.
    - Caso jogador 2 vença: Deve-se imprimir na tela todos os pokémons listados antes do pokémon que venceu a última batalha. Para fazer isso:
        i. Cria-se um while loop, que permanece enquanto o contador “c” for menor que a soma “qtd_player1+opcao” (que corresponde a linha do pokémon do jogador 2 vencedor da última batalha);
        ii. Dentro do loop, para pular linhas até posicionar o cursor na posição correta, utiliza-se o mesmo truque usado anteriormente em “imprime_restantes()”, “pokemon_player1” e “pokemon_player2” mas alterando a condição do while mais externo para “linha != c”, porque “c” é a referência de em qual linha do arquivo o cursor de leitura deve se posicionar;
        iii. Depois de pular as linhas necessárias, lê o nome do pokémon com fscanf, guardando-o em “nome”;
        iv. Imprime “nome” na tela;
        v. Incrementa +1 em “c”.
        vi. Repete o loop lendo a próxima linha, caso “c” não tenha chegado na linha do pokémon que venceu a última luta, o que termina o loop e o algoritmo.
    - Caso jogador 1 vença: Deve-se imprimir primeiro todos os pokémons do jogador 1 antes da linha correspondente a “opcao”, e depois todos os pokemons do jogador 2. Para isso, utiliza-se dois while loops semelhantes ao do caso anterior, mas com condições distintas:

i. O primeiro loop é idêntico ao do caso anterior com exceção da condição de repetição que passa a ser simplesmente enquanto “c” é menor que “opcao”. Ou seja, esse loop imprime na tela todos os pokémons do jogador 1 antes da linha “opcao”, que corresponte à linha onde está o pokémon vencedor da última batalha.
ii. Antes de iniciar o segundo loop, a variável “c” recebe seu valor inicial 1.
iii. Inicia-se o segundo loop, que se repete enquanto “c” for menor ou igual a “qtd_player2” e com conteúdo semelhante aos anteriores, mas com o mecanismo de pular linhas alterado em seu while externo, que passa a ter a condição de repetição como enquanto “linha” for diferente da soma “c + qtd_player1”, por que agora as linhas a serem imprimidas são todas as linhas contadas por “c” depois das M linhas (qtd_player1) de pokémons do jogador 1. Ou seja, esse loop imprime na tela todos os pokémons do jogador 2, o derrotado nesse caso.
iv. Depois de terminado o último loop, nada mais acontece e a função chega ao fim.

Exemplo de Execução

A fim de ficar claro o funcionamento do programa, segue um exemplo de execução com um determinado input:

Input:

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

Processo:

1. Leitura dos valores da primeira linha 3 e 2, atribuindo-os respectivamente às variáveis globais “qtd_player1” e “qtd_player2”.
2. Leitura da linha contendo o primeiro pokémon do jogador 1: “Squirtle”, guardando seus dados na estrutura “p1”.
3. Leitura da linha contendo o primeiro pokémon do jogador 2: “Golem”, guardando seus dados na estrutura “p2”.
4. Início da batalha, simulando cada ataque de Squirtle (p1) em Golem (p2), e depois de Golem (p2) em Squirtle (p1), alternadamente, até que a vida de um desses fique menor ou igual a zero após um ataque do adversário. Seguindo as regras do jogo, Squirtle causa 5 de dano em Golem, e Golem também causa 5 de dano em Squirtle. Ao final da luta, Squirtle derrota Golem, restando ainda 10 pontos de vida para o pokémon do jogador 1.

5. O programa então imprime a mensagem “Squirtle venceu Golem” e lê o segundo pokémon do jogador 2: “Charmander”, atualizando a estrutura “p2” com seus atributos.
6. Inicia-se a batalha entre Squirtle e Charmander, com Charmander infligindo o primeiro ataque, seguido por Squirtle e vice-versa, alternadamente. Seguindo as regras do jogo, Squirtle causa apenas 1 de dano em Charmander, e Charmander também causa apenas 1 de dano em Squirtle (perde poder de ataque por causa dos tipos), mesmo assim, Charmander derrota Squirtle, restando ainda 3 pontos de vida para o pokémon do jogador 2.
7. O programa então imprime a mensagem “Charmander venceu Squirtle” lê o segundo pokémon do jogador 1: “Vulpix”, atualizando a estrutura “p1” com seus atributos.
8. Inicia-se a batalha entre Vulpix e Charmander, com Vulpix infligindo o primeiro ataque, seguido por Charmander e vice-versa, alternadamente. Seguindo as regras do jogo, Vulpix causa apenas 1 de dano em Charmander, e Charmander causa 5 de dano em Vulpix, apesar da grande diferença de dano, Vulpix tem vida o suficiente para derrotar Charmander, restando ainda 5 pontos de vida para o pokémon do jogador 1.
9. O programa então imprime a mensagem “Vulpix venceu Charmander” e verifica que acabaram as opções de pokémon disponíveis para o jogador 2.
10. Então o programa declara o jogador 1 como vencedor imprimindo a mensagem “Jogador 1 venceu!”. Em seguida imprime “Pokemons sobreviventes” e inicia o algoritmo para imprimir na tela todos os pokémons que não foram derrotados.
11. O programa lê no arquivo o nome do pokémon do jogador 1 vencedor da última batalha, Vulpix, e imprime na tela. Em seguida, lê a linha seguinte e imprime o nome do segundo pokémon restante “Onix” na tela, terminando de imprimir todos os pokémons sobreviventes.
12. O programa imprime na tela a mensagem “Pokemons derrotados:”, e inicia o algoritmo para imprimir na tela todos os pokémons derrotados durante o jogo.
13. Primeiro lê e imprime na tela o nome dos pokémons derrotados do jogador 1, no caso, somente “Squirtle” . Em seguida lê e imprime na tela, em sequência, o nome de todos os pokémons do jogador 2: “Golem” e “Charmander”, terminando de imprimir todos os pokémons derrotados.
14. O programa fecha o arquivo de entrada e termina sua execução.

Output:

Squirtle venceu Golem
Charmander venceu Squirtle
Vulpix venceu Charmander
Jogador 1 venceu!
Pokemons sobreviventes:
Vulpix
Onix
Pokemons derrotados:
Squirtle
Golem
Charmander

Testes e Erros

Processo de Criação do Algoritmo

O processo de criação do algoritmo foi muito fluido, sem grandes problemas na lógica requerendo uma reformulação completa do trabalho já feito. Os maiores desafios do projeto foram relacionados ao uso correto da linguagem C para adaptação do algoritmo pensado.

Primeiramente, decidi que, por motivos de organização, todas as mecânicas do jogo que exigissem uma função à parte deveriam ser organizadas em um arquivo de biblioteca a parte “pokemon.h” , deixando mais claro o funcionamento do algoritmo da função principal do programa. Então trabalhei primeiro focado na função “main()” para montar um esboço de como funcionaria o programa e quais funções extras seriam necessárias. À medida que a função foi ganhando forma, fui criando em “pokemon.h” as funções auxiliares para o funcionamento do jogo, testando cada uma aos poucos.

Um desafio particular desse projeto, foi o de não utilizar uma forma de ler e guardar todos os pokémons do arquivo de entrada de uma vez, para depois ir utilizando-os no decorrer da execução do programa, mas ler e armazenar somente os pokémons que estão sendo utilizados em determinado momento, recorrendo a leitura do arquivo toda vez que se precisasse “pegar” um pokémon novo. Tomei essa decisão porque o meu compilador não permite utilizar vetores de structs com tamanho definido por variáveis (ex: “qtd_player1” lida no arquivo definiria o tamanho de um vetor de structs para armazenar todos os pokémons do jogador 1), então as únicas estruturas utilizadas são “p1” para o jogador 1 e “p2” para jogador 2, armazenando cada uma um pokémon por vez, e toda nova consulta por pokémon deveria ser feita lendo o arquivo.

Justamente por causa desse desafio, foi necessário pesquisar algumas técnicas para melhor manipular a leitura de um arquivo, o que também foi uma oportunidade de aprendizado. Dois exemplos muito claros disso foram a função “rewind()” e a técnica utilizada para avançar o cursor de leitura do arquivo até a posição desejada.

A parte principal do jogo de simular as batalhas e revelar o vencedor foi praticamente terminada em um dia. As funções que imprimem os pokémons sobreviventes e derrotados foram feitas depois, e demandaram um desafio à parte por serem um tanto complexas, já que envolviam muita manipulação na leitura do arquivo. Outras funções auxiliares como “ataque()” e “compara_tipos()” foram bem intuitivas de se fazer por que as regras do jogo são simples de se replicar em um algoritmo de computador.

Exemplos de Erros Comuns e Correções

Divergência no resultado das batalhas: O resultado das batalhas não está condizente com os exemplos dos casos de teste. Esse é um erro relacionado à interpretação das regras do jogo, onde determinados confrontos entre tipos de pokémons provocam um decréscimo ou acréscimo de 20% no ataque de um pokémon a outro. No caso, este acréscimo/decréscimo estava sendo feito diretamente no dano que um pokémon inflige em outro, e não no atributo “ataque” antes de realizar o cálculo do dano que seria causado.

- **Solução:** Mudar a lógica da função “ataque()” para incrementar/decrementar 20% em “atacante.ataque” antes de realizar a verificação se “atacante.ataque” é maior ou não que “defensor.defesa”.

Falha em modificar structs: Este erro ocorreu na elaboração das funções “pokemon_player1()” e “pokemon_player2()”, responsáveis por modificar as structs que armazenam o pokémon de cada jogador, as funções eram inicialmente do tipo void (sem retorno), o problema era que ao passar uma estrutura como argumento da função, as modificações são realizadas em uma cópia, e não diretamente na struct original, então as funções não conseguiam realizar o seu propósito.

- **Solução:** Mudar o tipo da função para que ela retorne o tipo abstrato “struct pokemon”, assim é possível atribuir as modificações às structs que precisam ser modificadas.

Testes Realizados

Além do teste mostrado no tópico “Exemplo de Execução” (Caso de Teste 1), também foram realizados vários outros testes para garantir que o programa está funcionando corretamente:

Caso de Teste 2:

Esse input e output estão registrados no documento “Casos de Teste”, que contém exemplos de testes e seus resultados esperados caso o programa funcione corretamente. Como se pode ver, o programa deu exatamente a resposta prevista, confirmando seu bom funcionamento.

Input:
5 5
Squirtle 15 25 40 agua
Sandshrew 17 13 28 pedra
Charmeleon 23 15 24 fogo
Raichu 19 16 28 eletrico
Cloyster 59 12 51 gelo
Charmander 25 12 36 fogo
Pikachu 25 15 24 eletrico
Dewgong 17 26 33 gelo
Wartortle 26 41 19 agua
Sandslash 62 31 40 pedra

Output:
Squirtle venceu Charmander
Pikachu venceu Squirtle
Sandshrew venceu Pikachu
Dewgong venceu Sandshrew
Charmeleon venceu Dewgong
Wartortle venceu Charmeleon
Wartortle venceu Raichu
Cloyster venceu Wartortle

Sandslash venceu Cloyster
Jogador 2 venceu!
Pokemons sobreviventes:
Sandslash
Pokemons derrotados:
Squirtle
Sandshrew
Charmeleon
Raichu
Cloyster
Charmander
Pikachu
Dewgong
Wartortle

Caso de Teste 3:

Esse é outro dos testes propostos no documento “Casos de Teste”. Novamente o programa deu exatamente o output esperado:

Input:
11 5
Pachirisu 15 17 9 eletrico
Ampharos 15 14 11 agua
Infernape 18 9 23 fogo
Spheal 14 18 18 gelo
Rampardos 9 18 22 pedra
Lotad 14 18 20 agua
Magmortar 21 16 24 fogo
Sealeo 15 19 28 gelo
Suicune 16 15 25 agua
Dugtrio 12 26 24 pedra
Charmander 18 12 26 fogo
Lombre 17 5 39 agua
Heatran 19 12 11 fogo
Walrein 15 14 20 gelo
Bonsly 11 19 17 pedra
Pikachu 35 30 24 eletrico

Output:
Pachirisu venceu Lombre
Heatran venceu Pachirisu
Ampharos venceu Heatran
Walrein venceu Ampharos
Infernape venceu Walrein
Bonsly venceu Infernape
Spheal venceu Bonsly
Pikachu venceu Spheal
Pikachu venceu Rampardos

Pikachu venceu Lotad
Pikachu venceu Magmortar
Pikachu venceu Sealeo
Pikachu venceu Suicune
Pikachu venceu Dugtrio
Pikachu venceu Charmander
Jogador 2 venceu!
Pokemons sobreviventes:
Pikachu
Pokemons derrotados:
Pachirisu
Ampharos
Infernape
Spheal
Rampardos
Lotad
Magmortar
Sealeo
Suicune
Dugtrio
Charmander
Lombre
Heatran
Walrein
Bonsly

Caso de Teste 4:

Esse é de longe o mais complexos dos “Casos de Teste”, ainda sim o output foi idêntico ao esperado:

Input:
26 26
Squirtle 10 15 15 agua
Bastiodon 18 30 38 pedra
Shinx 13 15 28 eletrico
Chimchar 12 14 9 fogo
Cloyster 20 25 35 gelo
Blastoise 15 20 25 agua
Onix 18 25 35 pedra
Pikachu 15 10 30 eletrico
Remoraid 12 14 18 agua
Spheal 15 18 22 gelo
Suicune 25 22 40 agua
Rampardos 25 20 40 pedra
Lombre 15 14 20 agua
Charmander 20 15 25 fogo
Ninetales 25 20 35 fogo
Wartortle 15 22 28 agua

Chimchar 20 16 25 fogo
Magneton 20 18 32 eletrico
Dugtrio 15 22 30 pedra
Magmortar 25 20 32 fogo
Sealeo 20 25 30 gelo
Bonsly 12 18 25 pedra
Raichu 18 12 35 eletrico
Lapras 25 30 40 gelo
Luxio 16 18 30 eletrico
Piloswine 18 22 28 gelo
Snorunt 16 18 25 gelo
Octillery 20 18 28 agua
Glalie 22 20 30 gelo
Heatran 28 24 38 fogo
Diglett 12 20 28 pedra
Psyduck 14 16 22 agua
Pachirisu 15 12 30 eletrico
Lotad 10 12 15 agua
Cranidos 20 18 32 pedra
Vulpix 18 15 22 fogo
Golduck 18 20 30 agua
Wartortle 12 18 20 agua
Growlithe 11 13 14 fogo
Dewgong 18 22 30 gelo
Shieldon 15 25 30 pedra
Magnemite 12 20 28 eletrico
Monferno 22 18 28 fogo
Ninetales 19 17 27 fogo
Voltorb 14 16 25 eletrico
Charmeleon 22 18 30 fogo
Swinub 12 14 20 gelo
Luxray 22 20 35 eletrico
Infernape 30 22 35 fogo
Magnezone 25 22 40 eletrico
Growlithe 15 12 18 fogo
Blastoise 20 25 40 agua

Output:
Snorunt venceu Squirtle
Bastiodon venceu Snorunt
Bastiodon venceu Octillery
Glalie venceu Bastiodon
Glalie venceu Shinx
Glalie venceu Chimchar
Cloyster venceu Glalie
Heatran venceu Cloyster
Heatran venceu Blastoise
Heatran venceu Onix

Heatran venceu Pikachu
Heatran venceu Remoraid
Heatran venceu Spheal
Suicune venceu Heatran
Suicune venceu Diglett
Suicune venceu Psyduck
Suicune venceu Pachirisu
Suicune venceu Lotad
Suicune venceu Cranidos
Suicune venceu Vulpix
Suicune venceu Golduck
Suicune venceu Wartortle
Suicune venceu Growlithe
Dewgong venceu Suicune
Rampardos venceu Dewgong
Shieldon venceu Rampardos
Lombre venceu Shieldon
Lombre venceu Magnemite
Monferno venceu Lombre
Monferno venceu Charmander
Ninetales venceu Monferno
Ninetales venceu Ninetales
Ninetales venceu Voltorb
Ninetales venceu Charmeleon
Ninetales venceu Swinub
Luxray venceu Ninetales
Luxray venceu Wartortle
Chimchar venceu Luxray
Infernape venceu Chimchar
Infernape venceu Magneton
Infernape venceu Dugtrio
Infernape venceu Magmortar
Infernape venceu Sealeo
Infernape venceu Bonsly
Infernape venceu Raichu
Infernape venceu Lapras
Luxio venceu Infernape
Magnezone venceu Luxio
Magnezone venceu Piloswine
Jogador 2 venceu!
Pokemons sobreviventes:
Magnezone
Growlithe
Blastoise
Pokemons derrotados:
Squirtle
Bastiodon
Shinx

Chimchar
Cloyster
Blastoise
Onix
Pikachu
Remoraid
Spheal
Suicune
Rampardos
Lombre
Charmander
Ninetales
Wartortle
Chimchar
Magneton
Dugtrio
Magmortar
Sealeo
Bonsly
Raichu
Lapras
Luxio
Piloswine
Snorunt
Octillery
Glalie
Heatran
Diglett
Psyduck
Pachirisu
Lotad
Cranidos
Vulpix
Golduck
Wartortle
Growlithe
Dewgong
Shieldon
Magnemite
Monferno
Ninetales
Voltorb
Charmeleon
Swinub
Luxray
Infernape

Caso de Teste 5:

O último dos casos de teste também ocorreu conforme o esperado.

Input:
5 7
Golduck 11 18 13 agua
Geodude 11 23 22 pedra
Raichu 14 18 21 eletrico
Rapidash 24 11 24 fogo
Rhydon 15 11 21 pedra
Charizard 20 22 19 fogo
Slowbro 21 16 25 agua
Magnemite 18 22 20 eletrico
Dewgong 22 14 22 agua
Tentacool 14 11 24 agua
Graveler 17 10 19 pedra
Marowak 19 22 21 pedra

Output:
Charizard venceu Golduck
Geodude venceu Charizard
Slowbro venceu Geodude
Raichu venceu Slowbro
Magnemite venceu Raichu
Magnemite venceu Rapidash
Rhydon venceu Magnemite
Dewgong venceu Rhydon
Jogador 2 venceu!
Pokemons sobreviventes:
Dewgong
Tentacool
Graveler
Marowak
Pokemons derrotados:
Golduck
Geodude
Raichu
Rapidash
Rhydon
Charizard
Slowbro
Magnemite

Testes individuais de tipos:

Essas batalhas de um pokémon para cada jogador, foram feitas para testar se o algoritmo está obedecendo as regras do jogo durante o combate. Para isso ficar bem claro, a cada ciclo do jogo são impressos os pontos de vida de cada pokémon:

Elétrico x Água:

Input:
1 1
Pikachu 25 15 24 eletrico
Squirtle 17 25 40 agua

Output:
Pikachu 24.00
Squirtle 40.00
Pikachu 24.00
Squirtle 35.00
Pikachu 23.00
Squirtle 35.00
Pikachu 23.00
Squirtle 30.00
Pikachu 22.00
Squirtle 30.00
Pikachu 22.00
Squirtle 25.00
Pikachu 21.00
Squirtle 25.00
Pikachu 21.00
Squirtle 20.00
Pikachu 20.00
Squirtle 20.00
Pikachu 20.00
Squirtle 15.00
Pikachu 19.00
Squirtle 15.00
Pikachu 19.00
Squirtle 10.00
Pikachu 18.00
Squirtle 10.00
Pikachu 18.00
Squirtle 5.00
Pikachu 17.00
Squirtle 5.00
Pikachu 17.00
Squirtle 0.00
Pikachu venceu Squirtle
Jogador 1 venceu!
Pokemons sobreviventes:
Pikachu
Pokemons derrotados:
Squirtle

Conclusão: O Pikachu ganha 20% de ataque por vantagem de “eletrico” sobre “agua”, indo a 30, 30 - 25 corresponde aos 5 de dano que Pikachu inflige em Squirtle a cada

turno. Enquanto isso, Squirtle causa apenas 1 de dano em Pikachu a cada turno, pois seu ataque fica inferior à defesa do adversário graças ao decréscimo de 20%.

Água x Fogo:

Input:
1 1
Blastoise 20 25 40 agua
Charizard 30 20 40 fogo

Output:
Blastoise 40.00
Charizard 40.00
Blastoise 40.00
Charizard 36.00
Blastoise 39.00
Charizard 36.00
Blastoise 39.00
Charizard 32.00
Blastoise 38.00
Charizard 32.00
Blastoise 38.00
Charizard 28.00
Blastoise 37.00
Charizard 28.00
Blastoise 37.00
Charizard 24.00
Blastoise 36.00
Charizard 24.00
Blastoise 36.00
Charizard 20.00
Blastoise 35.00
Charizard 20.00
Blastoise 35.00
Charizard 16.00
Blastoise 34.00
Charizard 16.00
Blastoise 34.00
Charizard 12.00
Blastoise 33.00
Charizard 12.00
Blastoise 33.00
Charizard 8.00
Blastoise 32.00
Charizard 8.00
Blastoise 32.00
Charizard 4.00
Blastoise 31.00
Charizard 4.00

Blastoise 31.00
Charizard 0.00
Blastoise venceu Charizard
Jogador 1 venceu!
Pokemons sobreviventes:
Blastoise
Pokemons derrotados:
Charizard

Conclusão: Graças ao bônus de 20% para Blastoise, este passa a ter 24 de ataque, 4 pontos a mais que a defesa de Charizard, que perde 20% de ataque e passa a ter 24, que é menor que a defesa de Blastoise. Resultado: Blastoise causa 4 de dano em Charizard a cada rodada, enquanto Charizard, causa apenas 1 de dano em Blastoise.

Fogo x Gelo:

Input:
1 1
Ninetales 23 18 27 fogo
Cloyster 20 25 35 gelo

Output:
Ninetales 27.00
Cloyster 35.00
Ninetales 27.00
Cloyster 32.40
Ninetales 26.00
Cloyster 32.40
Ninetales 26.00
Cloyster 29.80
Ninetales 25.00
Cloyster 29.80
Ninetales 25.00
Cloyster 27.20
Ninetales 24.00
Cloyster 27.20
Ninetales 24.00
Cloyster 24.60
Ninetales 23.00
Cloyster 24.60
Ninetales 23.00
Cloyster 22.00
Ninetales 22.00
Cloyster 22.00
Ninetales 22.00
Cloyster 19.40
Ninetales 21.00
Cloyster 19.40
Ninetales 21.00

Cloyster 16.80
Ninetales 20.00
Cloyster 16.80
Ninetales 20.00
Cloyster 14.20
Ninetales 19.00
Cloyster 14.20
Ninetales 19.00
Cloyster 11.60
Ninetales 18.00
Cloyster 11.60
Ninetales 18.00
Cloyster 9.00
Ninetales 17.00
Cloyster 9.00
Ninetales 17.00
Cloyster 6.40
Ninetales 16.00
Cloyster 6.40
Ninetales 16.00
Cloyster 3.80
Ninetales 15.00
Cloyster 3.80
Ninetales 15.00
Cloyster 1.20
Ninetales 14.00
Cloyster 1.20
Ninetales 14.00
Cloyster -1.40
Ninetales venceu Cloyster
Jogador 1 venceu!
Pokemons sobreviventes:
Ninetales
Pokemons derrotados:
Cloyster

Conclusão: Ninetales ganha vantagem de tipo e passa a ter um ataque poderoso de 27.6, ou seja, 2.6 de dano em Cloyster a cada turno. Cloyster se limita a apenas 1 de dano em Ninetales, pois perde 20% de ataque.

Gelo x Pedra:

Input:
1 1
Lapras 25 30 40 gelo
Onix 18 25 35 pedra

Output:
Lapras 40.00

Onix 35.00
Lapras 40.00
Onix 30.00
Lapras 39.00
Onix 30.00
Lapras 39.00
Onix 25.00
Lapras 38.00
Onix 25.00
Lapras 38.00
Onix 20.00
Lapras 37.00
Onix 20.00
Lapras 37.00
Onix 15.00
Lapras 36.00
Onix 15.00
Lapras 36.00
Onix 10.00
Lapras 35.00
Onix 10.00
Lapras 35.00
Onix 5.00
Lapras 34.00
Onix 5.00
Lapras 34.00
Onix 0.00
Lapras venceu Onix
Jogador 1 venceu!
Pokemons sobreviventes:
Lapras
Pokemons derrotados:
Onix

Conclusão: Lapras ganha os 20% de ataque e consegue infligir 5 de dano em Onix a cada turno. Onix só consegue causar 1 ponto de dano em Lapras.

Pedra x Elétrico:

Input:
1 1
Sandshrew 28 28 30 pedra
Pikachu 35 30 24 eletrico

Output:
Sandshrew 30.00
Pikachu 24.00
Sandshrew 30.00
Pikachu 20.40

Sandshrew 29.00
Pikachu 20.40
Sandshrew 29.00
Pikachu 16.80
Sandshrew 28.00
Pikachu 16.80
Sandshrew 28.00
Pikachu 13.20
Sandshrew 27.00
Pikachu 13.20
Sandshrew 27.00
Pikachu 9.60
Sandshrew 26.00
Pikachu 9.60
Sandshrew 26.00
Pikachu 6.00
Sandshrew 25.00
Pikachu 6.00
Sandshrew 25.00
Pikachu 2.40
Sandshrew 24.00
Pikachu 2.40
Sandshrew 24.00
Pikachu -1.20
Sandshrew venceu Pikachu
Jogador 1 venceu!
Pokemons sobreviventes:
Sandshrew
Pokemons derrotados:
Pikachu

Conclusão: Sandshrew ganhou 20% de ataque e causou 3.6 de dano em Pikachu a cada turno. Pikachu perdeu 20% de ataque e causou apenas 1 de dano por rodada em Sandshrew.

Conclusão

Após passar por todas as etapas para conclusão do projeto, desde a elaboração do algoritmo, várias tentativas e erros, correções, testes, comentários e por fim a documentação completa do projeto, pode-se concluir que o projeto é um sucesso, com um algoritmo robusto e suficiente para simular qualquer batalha pokémon, seguindo as regras do jogo previamente disponibilizadas, contanto que se respeite o formato de entrada exemplificado nesta documentação e uma quantidade computável pela memória do computador onde se está executando o programa.

Pessoalmente, o processo de elaboração deste projeto serviu de muito aprendizado e desenvolvimento de conhecimento teórico e habilidades práticas de programação, o que está de acordo com o objetivo da disciplina.