UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

TRABALHO PRÁTICO - PDS1

Carolina Penido Barcellos

Junho de 2024

1. Introdução

O trabalho consiste em um jogo de batalhas de Pokémons entre dois jogadores (1 e 2). Cada jogador tem um número determinado de Pokémons entre 1 e 100, e cada Pokémon possui 5 características, sendo elas: nome (string), tipo (string), poder de ataque (float), poder de defesa (int) e vidas (float). É importante ressaltar que alguns Pokémons ficam mais fortes ou mais fracos quando batalham contra outros, dependendo do seu tipo e do tipo do seu adversário. Por exemplo, um Pokémon do tipo "água" fica mais forte quando batalha contra o Pokémon do tipo "fogo", nesse caso o seu poder de ataque aumenta em 20%, e ele fica mais fraco quando batalha contra um Pokémon do tipo "elétrico", fazendo o seu poder de ataque se reduzir em 20%.

No início do jogo, o primeiro Pokémon do Jogador 1 irá atacar o primeiro Pokémon do Jogador 2. Caso seu poder de ataque contra o adversário seja maior que o poder de defesa do adversário, o adversário perderá um número de vidas equivalente à diferença entre os dois poderes. Caso contrário, perderá apenas 1 vida.

Se no final da batalha o número de vidas do Pokémon defensor for menor ou igual a 0, ele é derrotado e o próximo Pokémon do Jogador 2 entra na batalha, atacando o Pokémon vitorioso do Jogador 1. Se não for o caso, o mesmo Pokémon do Jogador 2 passa a atacar o Pokémon do Jogador 1, e eles continuam batalhando até que algum seja derrotado. Eles devem ir alternando no ataque. O jogo acaba quando todos os Pokémons de um jogador são derrotados.

O objetivo do jogo é mostrar o resultado de todas as batalhas, ou seja, imprimir o nome do Pokémon vencedor de cada batalha e o nome do Pokémon derrotado, por exemplo (Beltrano venceu Fulano), e mostrar também, ao final, qual foi o Jogador vencedor, que derrotou todos os pokémons do outro, e uma lista com os nomes dos Pokémons sobreviventes e uma com os nomes de todos os Pokémons derrotados no jogo.

A entrada esperada no programa é um arquivo txt, contendo a quantidade de Pokémons dos dois jogadores na primeira linha, e, nas seguintes, todas as informações necessárias para as batalhas acerca desses Pokémons. Uma linha deve ser reservada para conter as informações de cada Pokémon. Um exemplo de entrada é este:

```
1 1
Squirtle 10 15 15 agua
Vulpix 15 15 15 fogo
```

Para a saída, é importante imprimir na tela do computador essas informações presentes no arquivo e, depois, mostrar o resultado de cada uma das batalhas (quem venceu quem), o jogador vencedor (Jogador 1 ou 2), os Pokémons sobreviventes e os Pokémons derrotados. Por exemplo:

1 1
Squirtle 10 15 15 agua
Vulpix 15 15 15 fogo

Squirtle venceu Vulpix
Jogador 1 venceu
Pokémons sobreviventes:
Squirtle
Pokémons derrotados:
Vulpix

2. Descrição do Algoritmo e Procedimentos utilizados

Em primeiro lugar, criei um *arquivo .txt* para utilizar como caso de teste para o trabalho prático, chamado "arquivotrabalhopratico.txt". Com o *#define*, eu defini que MAX é igual a 100, que é o número máximo de Pokémons por cada jogador. Depois escrevi a *função main* e comecei declarando um ponteiro para *FILE,* chamado "arq", abri o arquivo e escrevi que desejo que ocorra a sua leitura, utilizando *fopen* e o modo de abertura *read, "r".*

Escrevi um *if* para o caso do arquivo não existir ou estar vazio. Ele imprime (com o *printf*) "Erro ao abrir o arquivo" se o ponteiro arq for igual a *NULL* (apontador que não aponta para nada), o que é útil para evitar erros no programa e mostrar ao usuário exatamente o problema se ele ocorrer.

Eu usei *fgets* para ler a primeira linha do arquivo, que corresponde ao número de Pokémons de cada jogador, e *strtok e atoi* para separar a string presente na linha em dois valores inteiros. Armazenei esses valores nas variáveis tipo int N e M, sendo "N" o número de Pokémons do jogador 1 e "M" o número de Pokémons do jogador 2. Caso eles sejam positivos de 1 a MAX(100), esses valores serão printados na tela e o restante do programa seguirá normalmente, entretanto, se não forem, o programa irá printar "A quantidade de Pokémons por jogador deve ser um número inteiro positivo de 1 a MAX(100)" e o jogo não será realizado.

Posteriormente, criei uma variável estruturada (*struct*), colocada no início do algoritmo, que dei o nome de "pokemon", e usei o comando *typedef* para criar um novo tipo com base nessa variável estruturada. Essa variável contém as strings "nome", tamanho 50, e "tipo", tamanho 10, um inteiro "defesa" e os floats "ataque" e "vida". Dessa forma, declarei dois vetores ("pokemos_jogador_1" e "pokemons_jogador_2") do tipo "pokemon". Essa alocação foi feita dinamicamente, de acordo com os valores N e M presentes no arquivo lido. Assim, utilizei *malloc e sizeof* para determinar o espaço de memória que deve ser reservado para cada apontador para pokemon e, no final do programa, liberei essa memória, usando *free*(nome da variável).

Eu utilizei o comando *for* para percorrer todos os elementos do primeiro vetor (de 0 até o N) e, dentro desse for, usei *fscanf* para ler os valores do arquivo, que correspondem às informações dos Pokémons do jogador 1 e, depois, *printf* para

imprimi-los na tela do mesmo modo como está no arquivo. Fiz a mesma coisa para os elementos do segundo vetor, lendo-os de 0 a M.

Depois disso fechei o arquivo (usando *fclose(arq)*) e criei uma função chamada "batalhas_pokemons", que é muito importante no jogo e foi colocada no início do programa. Ela verifica os tipos dos Pokémons que estão batalhando e identifica se o ataque do Pokémon atacante será aumentado em 20% (ataque SUPER_EFETIVO, que defini previamente como igual a 1.2, com o *#define*), reduzido em 20% (ataque MENOS_EFETIVO, que definido como igual a 0.8) ou permanecerá igual nessa batalha. Essa função recebe dois parâmetros do tipo pokemon (pk1 e pk2), sendo pk1 atacante e pk2 defensor, e, utilizando *strcmp* para realizar comparações entre as strings, retorna o novo poder de ataque (um float) do pk1 dependendo da situação.

Criei as variáveis de tipo float ataque_final_jogador1 e ataque_final_jogador2 para armazenar os valores do poder de ataque dos Pokémons após aplicar a função "batalhas_pokemons".

Por fim, foi necessário usar o comando *do while*, que verifica se a variável tipo int *atacante* (inicializado com 1) é igual a 1 ou 2. Se ela for igual a 1, isso significa que o Pokémon 1 está atacando o 2, e vice-versa. Também criei duas variáveis tipo int chamadas: "jogador_atual1" e "jogador_atual2", que correspondem ao índice do vetor do jogador 1 e do jogador 2. Elas começam sendo iguais a 0.

É importante ressaltar que o comando *do-while* executa o código enquanto a variável "jogador_atual1" for menor que "N" e a variável "jogador_atual2" for menor que "M".

Se atacante for igual a 1:

Caso o valor do campo ataque do pokemons_jogador_1 de índice "jogador_atual1", após possível alteração ao ser submetido à função "batalhas_pokemons", seja maior que o valor do campo defesa do pokemons_jogador_2 de índice "jogador_atual2", a diferença de valor entre os dois será subtraída do valor do campo vida do pokemons_jogador2 de índice "jogador_atual2". Se o valor não for maior, é subtraído apenas 1 do campo vida do "pokemons_jogador2" de índice "jogador_atual2". Depois, usei outro *if* para verificar se o valor do campo vida do "pokemons_jogador2" de índice "jogador_atual2" é menor, igual ou maior que 0. Se for menor ou igual a 0, o "pokemons_jogador2" de índice "jogador_atual2" perdeu, seu nome será adicionado (usando *strcat*) à string "derrotados", criada no início do programa e de tamanho 1000, será somado 1 ao jogador_atual2 e o "pokemons_jogador1" de índice "jogador_atual1" segue vitorioso, passando a batalhar com o próximo Pokémon do Jogador 2. Caso contrário, os Pokémons continuarão batalhando até que um deles seja derrotado. Em todos os casos, após o Pokémon do Jogador 1 atacar, o Pokémon do Jogador 2 deve ser o atacante. Estava com dúvida em relação a quem iria atacar em cada situação, mas perguntei ao monitor e ele me disse que é assim que deve ocorrer mesmo,

intercalando os atacantes. Tudo isso foi feito utilizando comandos *if e else if*, dentro de outro if, cuja condição é o atacante ser igual a 1.

Se atacante for igual a 2:

Caso o valor do campo ataque do pokemons_jogador_2 de índice "jogador_atual2", após possível alteração ao ser submetido à função "batalhas_pokemons", seja maior que o valor do campo defesa do pokemons_jogador_1 de índice "jogador_atual1", a diferença de valor entre os dois será subtraída do valor do campo vida do pokemons_jogador1 de índice "jogador_atual1". Se o valor não for maior, é subtraído apenas 1 do campo vida do "pokemons_jogador1" de índice "jogador_atual1". Depois, usei outro *if* para verificar se o valor do campo vida do "pokemons_jogador1" de índice "jogador_atual1" é menor, igual ou maior que 0. Se for menor ou igual a 0, o "pokemons_jogador1" de índice "jogador_atual1" perdeu, seu nome será adicionado (usando *strcat*) à string "derrotados", criada no início do programa e de tamanho 1000, será somado 1 ao jogador_atual1, e o "pokemons_jogador2" de índice "jogador_atual2" segue vitorioso e passa a batalhar com o próximo Pokémon do jogador 1. Caso contrário, os Pokémons continuarão batalhando até que um deles seja derrotado. Em todos os casos, após o Pokémon do Jogador 2 atacar, o Pokémon do Jogador 1 deve ser o atacante. Tudo isso foi feito utilizando comandos *if e else if*, dentro de outro else if, cuja condição é o atacante ser igual a 2.

Após o do while, haverá um jogador vencedor. Caso "jogador_atual1" seja igual a N, isso significa que o Jogador 2 venceu, pois os seus Pokémons batalharam contra todos os Pokémons do Jogador 1 e derrotaram todos. Assim, é impresso na tela "Jogador 2 venceu". Caso contrário, o Jogador 1 venceu, e isso é impresso.

Antes de finalizar o código, ainda é necessário verificar o nome dos Pokémons sobreviventes. Para tal, foi criada uma string chamada "sobreviventes" no início do código, de tamanho 1000. Para o caso do Jogador 1 ter sido o vencedor, serão adicionadas as strings do campo "nome" dos Pokémons do Jogador 1 cujos índices são iguais ou maior do que "jogador_atual1". Já para o caso de o Jogador 2 ser vencedor, a mesma coisa será feita, porém adicionando as strings do campo "nome" dos Pokémons do Jogador 2 cujos índices são iguais ou maior do que "jogador_atual2".

Por fim, as strings "sobreviventes" e "derrotados" são impressas na tela (utilizei o printf) da maneira pedida, e a função main retorna o valor 0.

3. Exemplo de execução

Input:

2 2
Pachirisu 15 17 9 eletrico
Ampharos 15 14 11 agua

Lombre 17 5 39 agua
Heatran 19 12 11 fogo

Processo:

1. Leitura do número de Pokémon para cada jogador.
2. Leitura dos dados de todos os Pokémons.
3. Verificar se o ataque do primeiro Pokémon atacante é super efetivo, menos efetivo ou não se altera nessa situação de batalha.
4. Realizar a dinâmica das batalhas. (Pode-se perceber que, nesse caso, o Jogador 1 é o vencedor e Ampharos é o único Pokémon sobrevivente.)
5. Imprimir os Pokémons sobreviventes e os Pokémons derrotados.

Output:

Pachirisu venceu Lombre
Heatran venceu Pachirisu
Ampharos venceu Heatran
Jogador 1 venceu
Pokémons sobreviventes:
Ampharos
Pokémons derrotados:
Lombre
Pachirisu
Heatran

4. Testes e erros

4.1 Processo de Criação do Algoritmo

Eu achei o processo de criação do algoritmo bem trabalhoso. Foram necessárias várias versões até eu finalmente desenvolver um programa com a lógica certa e que funcionasse o melhor possível dentro das circunstâncias. O mais difícil para mim foi fazer e organizar o loop que está relacionado com as batalhas dos Pokémons. Abaixo explicarei alguns dos meus erros e como eles foram corrigidos.

4.2 Exemplos de Erros Comuns e Correções

Erro: O resultado das batalhas estava dando errado em todos os testes. Os Pokémons que deveriam ser derrotados eram mostrados como vencedores das batalhas.

Correção: Após realizar vários testes vi que o problema estava na função "batalhas_pokemons", que tem como objetivo verificar se o poder de ataque do

atacante aumenta, diminui ou permanece igual na batalha. Ela estava alterando os valores do poder de ataque dos Pokémons permanentemente, de modo que no início de uma nova batalha o ataque do Pokémon estava alterado devido à situação de batalha anterior. Isso era um erro que ocorria devido ao fato de que na função foi utilizada a passagem de parâmetros por referência. No início de cada batalha, o Pokémon deve ter seu poder de ataque original e, depois, ser submetido à função "batalhas_pokemons" para verificar se naquele caso específico há alguma modificação. Para resolver o problema, troquei a passagem de parâmetros por referência na função para a passagem por valores, fazendo a função retornar um float (novo poder de ataque), ou invés de não retornar nada como antes (void). Posteriormente, eu criei duas variáveis chamadas "ataque_final_jogador1" e "ataque_final_jogador2", que armazenam os valores retornados pela função "batalhas_pokemons" em cada caso e são usadas na comparação entre os poderes de ataque e defesa dos Pokémons. Isso melhorou a situação, fazendo o programa realizar a maioria dos testes corretamente, porém ainda havia algum outro erro no código impedindo seu funcionamento 100% correto.

Erro: O resultado das batalhas ainda estava dando errado em alguns casos de testes.

Correção: Para resolver o problema, alterei os tipos das variáveis ataque e vida presentes no struct pokemon de int para float. Tive que mudar isso, porque meu programa estava arredondando os valores de ataque e vida dos Pokémons nas batalhas, fazendo com que algumas situações mostrassem resultados errados e incoerentes. Após mudar o tipo para float, todas as batalhas passaram a ocorrer da maneira correta, mostrando os resultados certos.

Erro: O loop *do-while* não parava quando deveria.

Correção: Para isso analisei as condições colocadas dentro do comando *while*, e vi que estava mandando o loop acabar quando a variável "jogador_atual1" ou a variável "jogador_atual2" for maior que N ou M, que representam o número de pokémons por jogador. Isso não estava correto, pois se a primeira variável for igual a N ou a segunda variável for igual a M, o loop já deve parar de rodar. Logo, eu arrumei a condição do while para: (!(jogador_atual2>=M ||jogador_atual1>=N));

Erro: Não estava imprimindo o nome do último Pokémon vencedor das batalhas. O programa simplesmente imprimia (venceu Fulano) na última batalha, e não (Beltrano venceu Fulano), como seria o esperado.

Correção: Para corrigir o problema, tive que analisar todo o meu código novamente, e percebi que o que estava impedindo o programa de funcionar da maneira correta era o fato de que utilizei dois *ifs* na sessão de batalhas, dentro do loop do-while, quando o que deveria ter feito era colocar um if (dessa forma: if (atacante==1)) e um else if (dessa forma: else if (atacante==2)). Assim, não tem mais como ter mais de um ataque na mesma rodada, o que acontecia antes. O programa verá se o atacante é o 1 naquela rodada e, se for, ele executará o código

dentro do if, não executando o else if depois. Para o caso do atacante ser o 1, e não o 2, o programa irá executar apenas o else if. Ao corrigir isso, esse erro não voltou a acontecer.

Após corrigir estes erros e arrumar o código, o programa passou a funcionar perfeitamente em todos os casos de testes.

4.3 Testes realizados

Teste 1:

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
```

Output:

```
Squirtle venceu Golem
Charmander venceu Squirtle
Vulpix venceu Charmander
Jogador 1 venceu
Pokémons sobreviventes:
Vulpix
Onix
Pokémons derrotados:
Golem
Squirtle
Charmander
```

Teste 2:

Input:

```
1 1
Squirtle 10 15 15 agua
Vulpix 15 15 15 fogo
```

Output:

```
Squirtle venceu Vulpix
Jogador 1 venceu
Pokémons sobreviventes:
Squirtle
Pokémons derrotados:
Vulpix
```

Teste 3:

Input:

2 2
Magmortar 2 2 2 fogo
Vulpix 2 2 2 fogo
Monferno 2 2 2 fogo
Growlithe 2 2 2 fogo

Resultados/Saída (Output):

Magmortar venceu Monferno
Growlithe venceu Magmortar
Vulpix venceu Growlithe
Jogador 1 venceu
Pokémons sobreviventes:
Vulpix
Pokémons derrotados:
Monferno
Magmortar
Growlithe

Teste 4:

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
Pokémons sobreviventes:
Pikachu
Pokémons derrotados:
Lombre
Pachirisu
Heatran
Ampharos
Walrein
Infernape
Bonsly
Spheal
Rampardos
Lotad
Magmortar
Sealeo
Suicune
Dugtrio
Charmander

Teste 5:

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
Pokémons sobreviventes:
Dewgong
Tentacool
Graveler
Marowak
Pokémons derrotados:
Golduck
Charizard
Geodude
Slowbro
Raichu
Rapidash
Magnemite
Rhydon

5. Conclusão

Gostei bastante do trabalho. Achei que abordou todos os tópicos que discutimos em sala de aula durante o semestre de um modo divertido e criativo. Eu aprendi alguns tópicos na prática e comecei a dominar melhor os conceitos vistos em sala.

Uma sugestão para tornar o programa mais interessante seria introduzir diferentes tipos de terreno e temperaturas que podem afetar nas batalhas. Assim, terrenos aquáticos beneficiam Pokémons do tipo "água", aumentando também o seu poder de ataque ou de defesa, enquanto terrenos rochosos beneficiam Pokémons do tipo "pedra". E climas frios (baixas temperaturas) beneficiam Pokémons do tipo "gelo", já climas mais quentes beneficiam Pokémons do tipo "fogo".