

Trabalho Prático 1 - Caça aos Pokemóns

Deiziane Natani da Silva

Turma TA1

1. Introdução

O trabalho tem como objetivo praticar os conceitos de Tipo Abstrato de Dados (TAD) e análise de complexidade vistos em aula.

O TP consiste em um jogo de caça aos Pokémons, onde o jogador deve buscar capturar o maior número de pokémons e sofrer o menor dano possível. O jogador que conseguir o maior score se torna um mestre pokemón.

Cada jogador possui inicialmente 3 pokebolas e para cada Pokémon capturado uma pokebola é utilizada. Existem os pokestops, onde é possível que haja a recarga de pokebolas quando o jogador não possuir nenhuma. Os Pokémon possuem Combat Power (CP), sendo este valor variando de 1 a 5.

Em caso de empate, o jogador que possuir o maior número de pokemóns de CP alto vence. Caso ainda haja empate, é verificado o caminho percorrido pelos jogadores. O que tiver o menor caminho será o vencedor. Se não houver desempate, todos os jogadores serão considerados vencedores.

O mesmo jogador não pode visitar a mesma célula mais de uma vez. Este também pode visitar uma célula sem capturar um Pokémon – caso em que o jogador não possui pokebolas. Cada jogador é independente e possui um mapa próprio, sendo assim, o movimento de um não interfere no movimento do outro.

Os jogadores podem se mover no máximo $3N-1$ vezes, onde N é o tamanho do mapa quadrado, ou até que estejam cercados por células já visitadas. Células vazias são marcadas no mapa como 0 e pokestops são marcados como 6. Cada Pokémon é marcado com números de 1 a 5. Cada jogador só pode se mover para uma das 8 células vizinhas por vez.

Será feito um programa em C, que recebe uma entrada através de um arquivo chamado “entrada.txt” e escreve o resultado em um arquivo “saida.txt”. Este possui um TAD obrigatório chamado “Jogador”, que armazena uma tabela Pokedex, com os pokemóns capturados e seus CP’s. O TAD deve implementar no mínimo três métodos - explorar(vizinhos), andar(vizinhança) e caminho_percorrido().

2. Implementação

Estrutura de Dados

Para a implementação do trabalho foi criado um Tipo Abstrato de Dados (TAD) Jogador, com a seguinte estrutura:

- Vetor char nome[3]:
 - Armazena o nome do jogador.
- Int score:
 - Armazena o score do jogador, este é aumentado cada vez que o jogador captura um Pokémon.
- Ponteiro int pokemons:
 - Armazena o Combat Power (CP) de todos os Pokémons capturados. Estes números são usados para pesquisar o nome dos Pokémons na Pokédex.
- Int poke:
 - Armazena o número de pokemons capturados.
- Vetor int posição[2]:
 - Armazena as coordenadas x e y que o jogador se encontra no momento.
- Int pokebolas:
 - Armazena o número de pokebolas que o jogador possui. Cada vez que o jogador captura um pokémon, esse número é decrementado de uma unidade. Quando o jogador visita um Pokestop e possui 0 pokebolas, esse número é incrementado de uma unidade.
- Ponteiro int caminho:
 - Armazena o caminho que o jogador percorreu desde a posição inicial.
- Ponteiro para ponteiro int mapa:
 - Armazena o mapa individual de cada jogador.
- Int tamanho_caminho:
 - Armazena o tamanho do caminho percorrido pelo jogador. Este valor é utilizado no critério de desempate.

Funções e Procedimentos

O TAD Jogador possui as seguintes funções:

- **int* explorar (int t, Jogador jog[], int numJ)**
 - Essa função recebe o tamanho da matriz *t*, o vetor de jogadores, e o número do jogador da rodada ($J1=1, J2=2...JN=N$).
 - Com esses valores, a função consegue retornar um vetor contendo o todas as posições vizinhas ao jogador (os primeiros 8 números do vetor) e também a posição mais vantajosa para este se movimentar (os dois últimos números do vetor).

- **void andar(int* exp, Jogador j[], int numJ)**
 - Essa função recebe como parâmetro o vetor retornado pela função explorar, o vetor de jogadores, e o número do jogador da rodada (J1=1, J2=2...JN=N).
 - Com esses valores, a função consegue executar o movimento do jogador para a melhor posição verificada pela função explorar. Caso o jogador capture um Pokémon ao andar, o número de pokebolas do jogador diminui e o score aumenta. Caso seja uma área de dano, o score diminui. Caso seja um pokestop, o número de pokebolas aumenta. Para todos os casos, a posição do jogador muda, e as coordenadas x e y novas são armazenadas.
- **Jogador* caminho_percorrido(Jogador j[], int numJ)**
 - Essa função recebe o vetor de jogadores, e o número do jogador da rodada *NumJ* (J1=1, J2=2...JN=N).
 - Com esses valores, a função apenas retorna o caminho percorrido pelo jogador da rodada.
- **void pokemons_capturados(char pokelista[][10], Jogador j[], int numJ)**
 - Essa função recebe, a Pokédex, o vetor de jogadores, e o número do jogador da rodada *NumJ* (J1=1, J2=2...JN=N).
 - Com esses valores, a função imprime na tela o nome dos Pokémons capturados pelo jogador.

Programa Principal

O programa principal lê os valores do arquivo “entrada.txt”: tamanho do mapa, o mapa, o número de jogadores, o nome e a posição inicial de cada jogador. Primeiro é criado um vetor do tipo Jogador, o tamanho do vetor é de acordo com o número de jogadores lido. Em seguida, são feitas as atribuições iniciais a cada jogador, definindo o número de pokebolas, score, caminho, etc. É feita a verificação se a posição inicial do jogador possui um Pokémon, é uma área de dano ou célula vazia.

A posição inicial também é marcada como já visitada, recebendo o valor 7, que foi definido como indicador de células visitadas e de bordas. Após isso é executado um loop que chama as funções para cada jogador, até que o jogo é encerrado. No fim do jogo, é criado um arquivo “saída.txt” e no arquivo é escrito o nome de cada jogador, o caminho percorrido por este e o vencedor.

Para definir um vencedor, primeiramente é verificado qual deles possui o maior score. Em caso de scores iguais, é feita uma segunda verificação. Essa leva em conta quanto Pokémons de maior CP cada jogador possui, o que tiver mais

Pokémons de CP alto, vence. Se ainda houver empate, é verificado o tamanho do caminho, armazenado na variável *tamanho_caminho* do TAD Jogador, percorrido pelos jogadores empatados, o que tiver feito o menor caminho vence. Caso não haja desempate segundos os critérios verificados, todos são considerados vencedores.

Não é feita nenhuma leitura do teclado, e toda a saída é feita no arquivo txt. Apenas a função *pokemons_capturados* pode imprimir (opcionalmente) o nome dos Pokémons, capturados pelo jogador em questão, no console.

Organização do Código, Decisões de Implementação e Detalhes Técnicos

O código está dividido em três arquivos, sendo estes: *main.c*, *jogador.c* e *jogador.h*. Os arquivos *jogador.c* e *jogador.h* implementam o TAD Jogador, e o *main.c*, implementa o programa principal.

Foi utilizada a constante MARKED com o valor 7, para definir regiões de bordas e já visitadas, isso facilita a identificação de onde o jogador pode se mover. A decisão da melhor posição a ser visitada leva em conta somente a vizinhança de 8, sendo que a melhor célula é escolhida onde o número da célula é maior. Sendo assim, mesmo que ainda possua pokebolas e tenha pokemons na vizinhança, se também houver um pokestop, o jogador o escolherá para visitar, pois esta célula é marcada com o maior número (6). O jogador continuará jogando até que não possa mais executar movimentos seja por já ter visitado todas as células ou por estar rodeado de células marcadas ou bordas.

A pokedéx criada no TAD, o ponteiro *int *pokemons*, armazena somente o CP do Pokémon, dessa forma, para que se saiba o nome do Pokémon correspondente ao CP, é necessário executar a função *pokemons_capturados*.

Como o movimento de um jogador no mapa não influencia no movimento de outro, cada jogador possui um mapa próprio. São feitas todas as jogadas do primeiro jogador, e quando este encerra seu jogo, o próximo jogador inicia seu jogo, e o loop segue até que todos os jogadores tenham encerrado seu jogo.

O compilador utilizado foi o Dev-C++ v. 5.10, no sistema operacional Windows 7.

3. Análise de Complexidade

A análise de complexidade será feita em função da variável *t_matriz*, que indica o tamanho do mapa quadrado recebido pelo arquivo. Como este número é sempre maior ou igual ao número de jogadores, é mais significativo.

Função explorar(): Inicialmente a função executa 8 comparações para verificar se há bordas na região vizinha, em seguida são feitos dois loops que fazem a verificação da melhor posição da vizinhança. Como esse valor é sempre 8, independentemente do tamanho do mapa, eles são classificados como **O(1)**.

Função andar(): Nesta função, o jogador se move para a posição indicada como a melhor pela função explorar. São feitas 3 comparações e várias atribuições $O(1)$. Como o número de comparações e atribuições não depende do tamanho do mapa, eles são classificados como **$O(1)$** .

Função caminho_percorrido(): A função apenas retorna o caminho percorrido, esse comando é classificado como **$O(1)$** .

Função pokemons_capturados(): A função imprime o nome dos Pokémons capturados, utilizando um loop que executa até o número total de Pokémons capturados, fazendo a impressão de cada nome. Como nenhum desses valores depende de t_matriz , elas tem custo **$O(1)$** .

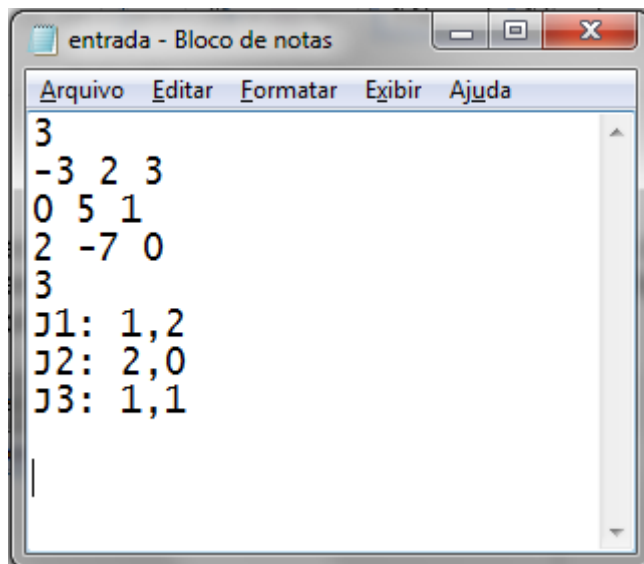
Programa principal: o main lê os dados de entrada através de loops. Após ler o tamanho da matriz mapa, através de loops aninhados é feita a leitura de cada célula da matriz. Essa operação possui custo $O(n)$ para o primeiro loop e $O(n)$ para o segundo, sendo assim, o custo total será $O(n)O(n) = O(n^2)$. Em seguida a alguns comandos de leitura, são feitas cópias do mapa para cada jogador, sendo feito 3 loops aninhados. O loop externo não se relaciona ao tamanho do mapa, mas sim à quantidade de jogadores, valor não levado em conta nessa análise. Os loops internos, ambos tem o mesmo custo $O(n^2)$.

São feitas as leituras dos nomes e das posições dos jogadores, também em um loop que se relaciona ao número de jogadores.

Após vários comando executados de custo $O(1)$, o programa principal chama as funções até $3*n$ vezes, ou até que um jogador não possa mais se movimentar. O loop tem custo $O(n)$. Dessa forma, a sua complexidade é regida pela função de maior custo computacional: $O(1) + O(1) + O(1) + O(n^2) = O(n^2)$

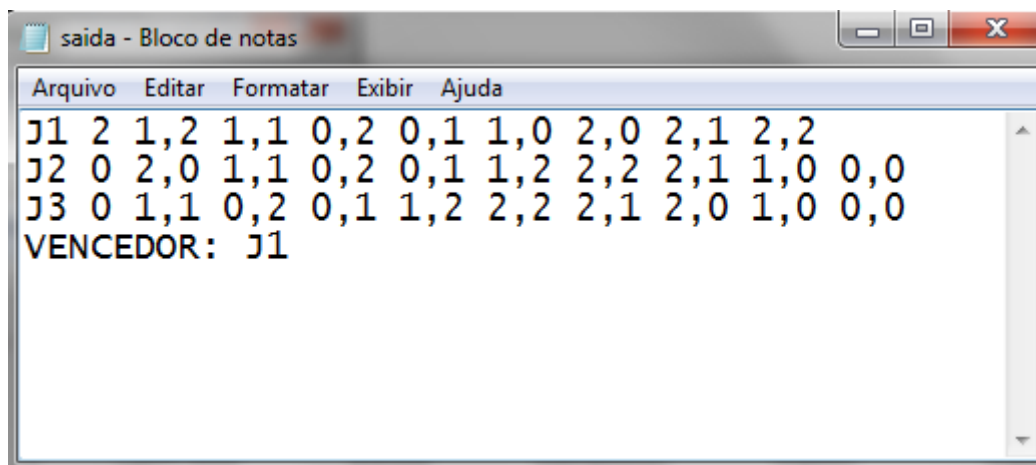
4. Testes

Foram feitos testes no programa utilizando os casos de teste disponibilizados no moodle e o exemplo mostrado na especificação do TP. A figura abaixo mostra um exemplo de arquivo de entrada:



```
3
-3 2 3
0 5 1
2 -7 0
3
J1: 1,2
J2: 2,0
J3: 1,1
```

Após a execução do programa, a saída gerada foi:



```
J1 2 1,2 1,1 0,2 0,1 1,0 2,0 2,1 2,2
J2 0 2,0 1,1 0,2 0,1 1,2 2,2 2,1 1,0 0,0
J3 0 1,1 0,2 0,1 1,2 2,2 2,1 2,0 1,0 0,0
VENCEDOR: J1
```

Para um dos exemplos dos casos de teste oferecidos:

```
entrada - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
8
4 0 0 0 4 6 4 0
6 0 4 -7 0 3 5 -2
6 6 0 0 6 0 0 6
5 0 0 4 0 6 0 0
0 0 0 0 0 5 0 6
0 1 3 2 -8 0 6 5
6 0 1 2 5 0 0 6
4 1 2 6 0 6 6 0
4
J1: 3,6
J2: 6,7
J3: 3,3
J4: 3,6
```

Como J1 e J4 começaram do mesmo ponto, eles pegaram os mesmos pokemóns e percorreram o mesmo caminho. Logo, eles empataram nos scores, e não desempataram através dos critérios de desempate. Dessa forma, ambos foram considerados vencedores, como mostra abaixo:

```
saida - Bloco de notas
Arquivo  Editar  Formatar  Exibir  Ajuda
J1 33 3,6 2,7 1,6 0,5 0,4 1,5 2,4 3,5 4,5 5,6 4,7
5,7 6,7 7,6 7,5 6,4 7,3 6,3 5,2 5,3 6,2 7,2 7,1 6,0
7,0
J2 19 6,7 5,6 4,7 5,7 4,6 3,5 2,4 3,3 2,2 2,1 1,0
2,0 3,0 3,1 3,2 2,3 1,2 0,1 0,0 1,1 0,2 0,3 0,4 0,5
1,6
J3 25 3,3 2,4 3,5 4,5 5,6 4,7 5,7 6,7 7,6 7,5 6,4
7,3 6,3 5,2 5,3 6,2 7,2 7,1 6,0 7,0 6,1 5,1 4,0 3,0
2,0
J4 33 3,6 2,7 1,6 0,5 0,4 1,5 2,4 3,5 4,5 5,6 4,7
5,7 6,7 7,6 7,5 6,4 7,3 6,3 5,2 5,3 6,2 7,2 7,1 6,0
7,0
VENCEDOR: J1 J4
```

5. Conclusão

O trabalho prático foi finalizado e através dele consegue-se fazer a simulação de uma partida de caça aos pokemóns.

As principais dificuldades encontradas por mim foram em relação à interação do TAD e das funções. Mas, apesar disso, não foi um trabalho difícil, mas sim bastante trabalhoso. Acredito que hajam melhores formas de implementá-lo, de forma a deixá-lo mais eficiente e mais organizado.

Referências

[1] C, Waldemar; C, Renato; R, Lucas. Introdução a estruturas de dados, 2ª edição, Editora Elsevier, 2004

Anexos

Listagem dos programas:

Main.c

Jogador.h

Jogador.c