

# Documentação Pokemon Go

## Guilherme Henrique Resende de Andrade

### Dados necessários à compilação

```
gcc -c Jogador.c  
gcc -c Mapa.c  
gcc Main.c Jogador.o Mapa.o
```

## 1.Introdução

Pokemon Go é um jogo da atualidade que tem por objetivo conseguir o maior número de Pokemon<sup>1</sup> com Poder de Combate(CP) alto, o que faz um treinador mais forte que os outros. No jogo original, o jogador necessita caminhar no mundo real para encontrar os Pokemon, contudo, faremos com que este jogo execute de forma autônoma.

O objetivo deste trabalho é fazer com que a partir de um arquivo de entrada, o programa leia todas as informações e faça com que o jogador percorra a matriz de dados fazendo escolhas de forma a otimizar sua pontuação com o menor número de movimentos possível. Deverão ser implementadas funções para leitura do arquivo de entrada, inicialização(lista, jogador, mapa), movimentação, análise do melhor caminho, condição de movimento, salvar o caminho percorrido, análise para casos de empate, eleição de um ou mais vencedores, escrita no arquivo de saída.

<sup>1</sup> O substantivo não varia no plural.

## 2.Implementação

### Estrutura de Dados

Para implementação do trabalho foram criados os Tipos Abstratos de Dados Mapa, Jogador, Lista e Célula com as seguintes estruturas:

#### Mapa:

```
int **mapa;
```

A variável mapa é um ponteiro para ponteiro para inteiro, que é preenchido a partir de uma alocação dinâmica durante o tempo de execução devido ao fato de o tamanho da pode variar.

#### Jogador:

```
int id, passos, pokebolas, pontos, linha, coluna, pokemon[5];
```

A variável id é um inteiro que se refere ao identificador pessoal de cada jogador, já os campos passos, pokebolas, pontos, linha e coluna representam respectivamente o número de movimentos de cada jogador, a quantidade de pokebolas<sup>2</sup> que o jogador possui, a pontuação alcançada, e as coordenadas em que o jogador se localiza. O campo pokemon é um vetor estático que representa os monstros que podem aparecer no caminho do jogador, estes monstros são representados na notação numérica e só existem 5 possibilidades de aparição.

### Lista:

`t_celula *primeiro, *ultimo;`

A variável `primeiro` e `ultimo` são do tipo `t_celula` e guardam a primeira e a última célula da lista encadeada. O tipo `t_celula` é descrito por:

`int linha, coluna;`

`struct celula *prox;`

Onde as variáveis inteiras `linha` e `coluna` armazenam as coordenadas em que o jogador já esteve, enquanto a variável do tipo `t_celula *prox` armazena o endereço da próxima célula da lista encadeada.

<sup>2</sup> Pokebolas são os objetos utilizados para capturar os Pokemon.

### Funções e Procedimentos

Os TAD's criados são constituídos das seguintes funções:

**`t_jogador *aloca_jogador(int numero_de_jogadores):`** Recebe o campo inteiro 'numero\_de\_jogadores', aloca dinamicamente um vetor do tipo `t_jogador` e retorna o endereço do mesmo para a função chamadora.

**`void inicia_player(t_jogador *vetor_de_jogadores, FILE *in, int numero_de_jogadores):`** Recebe um ponteiro para um espaço de memória que armazena um vetor do tipo `t_jogador` de tamanho 'numero\_de\_jogadores' e recebe também um fluxo de dados que aponta para o arquivo de entrada. A partir disso, utiliza o fluxo de entrada para ler as coordenadas de cada jogador e salvar em suas respectivas estruturas, ainda assim, inicializa todas as variáveis relacionadas ao jogador, como pokebolas, pokemon, id.

**`void exibe_jogadores(t_jogador *vetor_de_jogadores, int numero_de_jogadores):`** Recebe um ponteiro para um espaço de memória que armazena um vetor do tipo `t_jogador` de tamanho 'numero\_de\_jogadores', posteriormente, cria um loop e exibe todas as características relacionadas com o jogador atual.

**`int andar(t_jogador *jogador, t_lista *caminho_jogador, int numero_de_jogadores, int **mapa, int tamanho_do_mapa):`** Recebe um ponteiro para tipo `t_jogador`, um ponteiro para a lista que grava o caminho percorrido pelo jogador e as variáveis inteiras `**mapa` e `tamanho_do_mapa` são respectivamente para armazenar por referência a matriz do jogo e as dimensões do mapa. Desta forma, a função analisa se o jogador possui pokebolas para realizar a jogada, se não tiver, ele procura algum pokestop<sup>3</sup> na vizinhança para repor suas pokebolas, caso ele não encontre nenhum pokestop em suas proximidades, ele anda pelo caminho de menor dano até encontrar algum. Contudo, se o jogador possuir alguma pokebola, ele procura pelo pokemon de maior CP em sua vizinhança e o captura.

**void move(t\_jogador \*jogador, int posicao):** Recebe um ponteiro para tipo t\_jogador e um inteiro 'posicao' que representa o local para o qual o jogador se moverá. Com isso, a função move o jogador para a posição determinada da vizinhança.

**int \*explorar(int \*\*mapa, int linha, int coluna, int tamanho\_do\_mapa):** Recebe ponteiro para ponteiro para inteiro que armazena o endereço da matriz de jogo, e as variáveis do tipo inteiro linha, coluna, tamanho\_do\_mapa que armazenam respectivamente as coordenadas do jogador e as dimensões do mapa. Assim sendo, é criado um vetor dinâmico de inteiros para armazenar os dados da vizinhança. Posteriormente, são realizados testes com a vizinhança para julgar quais locais são passíveis de serem visitados e posteriormente é retornado o vetor de vizinhos para a função anterior da pilha de execução.

**int \*vencedor(t\_jogador \*vetor\_de\_jogadores, int numero\_de\_jogadores):** Recebe um ponteiro para espaço na memória que guarda um vetor do tipo t\_jogador de dimensão 'numero\_de\_jogadores'. Com isso, a função compara todas as posições do vetor de forma a eleger um vencedor dentre eles com base no número de pontos. Contudo, se não for possível, utiliza como critério de desempate o número de pokemon com CP alto que cada um possui, e se ainda assim o empate persistir, é utilizado o número de passos como critério de desempate. Assim sendo, a função retorna o/os vencedores para a função chamadora.

**int possui\_pokebolas(t\_jogador jogador):** Recebe em passagem por valor uma variável do tipo t\_jogador e confere se o jogador possui pokebolas para realizar a jogada, se sim, retorna 1, caso contrário, retorna 0.

**void pokestop(t\_jogador \*jogador):** Recebe o endereço de memória de uma variável do tipo t\_jogador e incrementa em 1 o número de pokebolas do jogador referenciado.

**void FLV(t\_lista \*caminho\_jogador, t\_jogador jogador):** Recebe um ponteiro para t\_lista e uma variável do tipo t\_jogador. Com isso, a função faz com que a primeira e a última célula da lista apontem para o mesmo lugar, ou seja, a lista não possui membros ainda.

**int vazia(t\_lista caminho\_jogador):** Recebe um campo do tipo t\_lista e retorna 1 para casos em que a lista estiver vazia, e 0 caso contrário.

**void caminho\_percorrido(t\_lista \*caminho\_jogador, t\_jogador jogador):** Recebe ponteiro para tipo t\_lista e uma variável do tipo t\_jogador que são utilizadas para salvar o caminho atual do jogador na lista encadeada.

**void imprime\_saida(t\_lista \*caminho\_jogador, t\_jogador jogador, FILE \*out):** Recebe uma variável ponteiro para tipo t\_lista, uma variável do tipo t\_jogador e uma variável ponteiro para fluxo de dados do tipo FILE. O método imprime a saída formatada do jogador no arquivo.

**int \*\*aloca(int tamanho\_mapa):** Recebe o inteiro que representa a dimensão da matriz de jogo e aloca dinamicamente na memória a matriz correspondente.

**void inicia\_mapa(int \*\*mapa, FILE \*in, int tamanho\_mapa):** Recebe um ponteiro para ponteiro para inteiro que representa a matriz de jogo, um ponteiro para FILE que representa o fluxo de entrada de dados e um inteiro 'tamanho\_mapa' que guarda as dimensões do mapa. Com isso, ele percorre todo mapa preenchendo as posições com os dados correspondentes do arquivo de entrada.

**void exhibe\_mapa(int \*\*mapa, int tamanho\_mapa):** Recebe um ponteiro para ponteiro para inteiro que armazenará o mapa e um inteiro 'tamanho\_mapa' que representa a dimensão da matriz. A partir disso, o método imprime todo mapa na tela com um loop.

### **Programa Principal**

O programa principal declara cinco variáveis do tipo inteiro, sendo elas 'tamanho\_mapa', 'numero\_de\_jogadores', 'i', 'condicao', '\*vencedores', uma variável para cada um dos tipos t\_mapa, t\_jogador, t\_lista, sendo elas 'mapa', '\*vetor\_de\_jogadores', '\*caminho\_jogador', e duas variáveis do tipo FILE '\*in' e '\*out'.

Desta forma, a variável '\*caminho\_jogador' é inicializada com uma alocação dinâmica no momento da declaração. No momento da declaração também são inicializadas as variáveis '\*in' e '\*out' com a chamada da função fopen que retorna seus respectivos fluxos de dados;

Na continuação do programa, a variável 'tamanho\_mapa' é preenchida a partir do arquivo de entrada com a função fscanf, e posteriormente, são chamadas as funções que alocam o mapa dinamicamente e o inicia, respectivamente, aloca() e inicia\_mapa(). Por conseguinte, a variável 'numero\_de\_jogadores' também é preenchida com a função fscanf, e desta forma, possibilita a chamada das funções responsáveis por alocar dinamicamente o vetor de jogadores e iniciar os jogadores, sendo elas, aloca\_jogador() e inicia\_player().

A próxima função do arquivo trata do fechamento do fluxo de entradas com a função fclose. Feito isso, o programa inicia um loop que faz com que cada jogador ande até o limite de passos, ou até ser impossibilitado de caminhar.

Dentro do loop, são chamadas as funções FLV que é responsável por criar lista vazia, e depois é criado outro loop que executa os passos de cada jogador. Posteriormente, os dados do jogador corrente são salvos no arquivo de saída. Com isso, novamente é aberto o fluxo de entrada, lidos todos os dados da matriz e fechado o fluxo de arquivos novamente.

Por final, é inicializada a variável '\*vencedores' com a chamada da função vencedor() e imprimido no arquivo de saída todos os vencedores. A última função do arquivo é o fechamento do fluxo de saída com fclose.

<sup>3</sup> Pokestops são pontos em que os jogadores recebem recargas de pokebola.

## **3.Complexidade**

**Procedimento aloca\_jogador:** Esse procedimento só executa um comando de atribuição, portanto o custo do procedimento é  **$O(1)$** ;

**Procedimento inicia\_player:** Esse procedimento executa algumas instruções de atribuição e possui dois loops do tamanho do número de jogadores, desta forma, o custo do procedimento é  **$O(1)$** .

**Procedimento exhibe\_jogadores:** Esse procedimento executa apenas um loop do tamanho do número de jogadores, portanto, o custo do procedimento é  **$O(1)$** .

**Função andar:** Essa função no pior caso executa cinco comparações e três loops do tamanho do número de jogadores. Assim sendo, o custo da função é  **$O(1)$** .

**Procedimento move:** Esse procedimento realiza apenas um Switch em uma variável, desta forma, o custo do procedimento é  **$O(1)$** .

**Função explorar:** Essa função, no pior caso, executa um loop de tamanho três e duas comparações, por conseguinte, o custo da função é  **$O(1)$** .

**Função vencedor:** Essa função realiza cinco comparações e cinco loops do tamanho do número de jogadores, sendo assim, o custo da função é  **$O(1)$** .

**Função possui\_pokebolas:** Essa função realiza apenas uma comparação, portanto, o custo da função é  **$O(1)$** .

**Procedimento pokestop:** Esse procedimento realiza apenas uma atribuição, portanto, o custo do procedimento é  **$O(1)$** .

**Procedimento FLV:** Esse procedimento faz uma alocação dinâmica e realiza cinco atribuições, desta forma, o custo do procedimento é  **$O(1)$** .

**Função vazia:** Essa função realiza apenas uma comparação, por conseguinte, o custo da função é  **$O(1)$** .

**Procedimento caminho\_percorrido:** Esse procedimento apenas faz uma alocação dinâmica e realiza cinco atribuições, portanto, o custo do procedimento é  **$O(1)$** .

**Procedimento imprime\_saida:** Esse procedimento cria um loop que, no pior caso, se repete  $3n-1$  vezes com o custo 1 por execução. Desta forma, o procedimento é da ordem de  **$O(n)$** .

**Função aloca:** Essa função realiza a alocação dinâmica da matriz, para isso ela cria um loop do tamanho da dimensão do mapa com uma operação de custo 1, portanto, o custo da função é  **$O(n)$** .

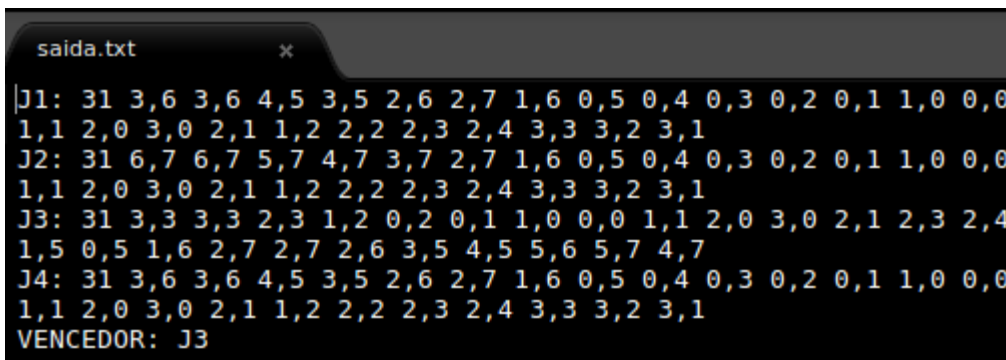
**Procedimento inicia\_mapa:** Esse procedimento preenche a matriz com os dados pertencentes à cada posição, para isso, são criados dois loops aninhados que se repetem N vezes (tamanho do mapa). Assim sendo, o loop mais interno executa N vezes, o que é da ordem de  $O(n)$ . Já o loop externo executa uma instrução  $O(n)$ , N vezes, o que gera uma ordem de  $O(n^2)$ .

**Procedimento exhibe\_mapa:** Esse procedimento preenche a matriz com os dados pertencentes à cada posição, para isso, são criados dois loops aninhados que se repetem N vezes (tamanho do mapa). Assim sendo, o loop mais interno executa N vezes, o que é da ordem de  $O(n)$ . Já o loop externo executa uma instrução  $O(n)$ , N vezes, o que gera uma ordem de  $O(n^2)$ .

**Programa Principal:** O programa principal apenas executa as funções acima, e chama a função **inicia\_mapa()** quatro vezes, assim sendo, o custo de execução do mesmo é da ordem de  $O(n^2)$ .

## 4. Testes

Alguns testes foram realizados de forma a comprovar o funcionamento do programa. Os testes foram feitos em uma máquina com Intel Core I5, 2 Gb de memória RAM e configurada com sistema operacional Linux.



```
saida.txt
J1: 31 3,6 3,6 4,5 3,5 2,6 2,7 1,6 0,5 0,4 0,3 0,2 0,1 1,0 0,0
1,1 2,0 3,0 2,1 1,2 2,2 2,3 2,4 3,3 3,2 3,1
J2: 31 6,7 6,7 5,7 4,7 3,7 2,7 1,6 0,5 0,4 0,3 0,2 0,1 1,0 0,0
1,1 2,0 3,0 2,1 1,2 2,2 2,3 2,4 3,3 3,2 3,1
J3: 31 3,3 3,3 2,3 1,2 0,2 0,1 1,0 0,0 1,1 2,0 3,0 2,1 2,3 2,4
1,5 0,5 1,6 2,7 2,7 2,6 3,5 4,5 5,6 5,7 4,7
J4: 31 3,6 3,6 4,5 3,5 2,6 2,7 1,6 0,5 0,4 0,3 0,2 0,1 1,0 0,0
1,1 2,0 3,0 2,1 1,2 2,2 2,3 2,4 3,3 3,2 3,1
VENCEDOR: J3
```

1-Teste com mapa de dimensão 8x8.

```
saida.txt x
J1: 99 27,29 27,29 27,28 26,27 25,27 26,28 25,29 25,28 24,27
23,27 22,26 22,25 21,26 20,27 20,28 19,29 18,28 18,27 19,27
18,26 18,25 17,25 18,24 19,23 20,22 21,23 20,23 19,22 18,21
17,21 16,21 17,20 16,20 15,20 14,20 13,20 12,19 13,18 12,17
12,16 11,15 10,16 9,16 8,15 7,14 6,14 6,13 5,12 4,11 3,10 2,9
2,8 1,7 0,6 0,5 0,4 1,4 2,3 2,2 1,2 0,1 0,0 1,0 2,1 3,0 4,0 4,2
5,2 5,3 6,3 7,3 6,2 5,1 4,1 3,2 3,3 2,4 3,5 3,6 2,5 1,5 1,6 0,7
0,8 0,9 0,10 0,11 0,12 0,14 1,13 2,12
J2: 118 29,19 29,19 28,18 27,17 26,16 26,17 25,18 24,19 25,19
24,20 24,21 25,21 25,22 25,23 26,24 26,23 27,22 28,23 28,24
27,23 27,24 26,25 27,25 28,25 29,26 28,26 27,27 26,26 25,25
24,24 23,23 22,24 21,24 21,23 20,22 19,23 18,24 18,23 17,23
16,23 15,22 14,23 13,23 12,24 13,25 12,26 12,25 11,26 10,25
9,24 9,23 8,22 9,21 10,20 11,20 10,21 9,20 8,19 8,18 9,19 8,20
7,21 7,20 6,19 7,19 6,18 5,17 4,17 3,17 2,16 1,15 0,15 0,14
1,13 2,12 3,11 2,10 3,10 2,9 2,8 1,7 0,6 0,5 0,4 1,4 2,3 2,2
1,2 0,1 0,0 1,0
J3: 48 1,6 1,6 0,5 1,5 2,6 3,5 3,6 2,5 1,4 0,4 0,3 0,2 0,1 0,0
1,0 2,1 2,2 1,2 2,3 3,2 4,2 5,3 5,2 6,2 7,3 8,2 7,1 7,0 8,0 8,1
7,2 6,3 6,4 5,5 4,5 4,6 4,7 4,8 3,7 2,8 1,7 0,6 0,7 0,8 0,9
0,10 0,11 0,12 0,14 1,13 2,12 3,11 2,10 3,10 2,9 3,8 4,9 4,10
VENCEDOR: J2
```

2-Teste com mapa de dimensão 30x30

## 5.Conclusões

A implementação do trabalho transcorreu sem maiores problemas e os resultado ficaram dentro do esperado.

A principal dificuldade encontrada foi a determinação de quais casas da vizinhança poderiam ser visitadas.

## Referências

- <http://www.cprogressivo.net/>
- <http://stackoverflow.com/>