Pokemon Go

Documentação

Algoritmos e Estrutura de Dados II
Universidade Federal de Minas Gerais

Tatiana Santos Camelo de Araujo – 2015086298

1. Introdução

O objetivo deste trabalho é implementar um jogo similar ao Pokémon Go. Para facilitar a compreensão do código, foram usadas nomes intuitivos para variáveis e funções, derivadas do inglês, desta forma, procurou-se que o entendimento do leitor não seja comprometido ao longo do código.

Houve utilização de Tipos Abstratos de Dados, criou-se um tipo jogador (chamado no código como "type_player"). São implementadas funções para que, ao ler uma entrada, o programa simule as melhores escolhas – isto é, as que gerem mais pontos ou menos danos – de cada jogador e apresente um vencedor, de acordo com um sistema de pontuação.
O programa foi feito em forma para ser compilado com "gcc *.c -Wall -o main".

2. Implementação

Para a implementação, foram usados 5 arquivos no total, para melhor modularização do código, sendo estes: main.c player.c; player.h; maps.c e maps.h.

Player

As implementações para player, player.c e player.h, possuem informações como o TAD player, que contém um vetor para Pokédex, e suas funções. O Pokédex foi feito como um vetor. A cada Pokémon de certo tipo capturado, um contador na posição referente ao CP do Pokémon aumenta em 1, mostrando quantos Pokémon de cada tipo o player tem. A primeira posição do vetor ficou determinado pela contabilização de score do player, de forma que ao andar em área de perigo ou ao capturar um Pokémon novo, será esta a posição que sofrerá alteração da contagem de seus pontos totais. Todas as outras localizações mostraram a quantidade de Pokémon capturados, em que o CP é igual a posição no vetor.

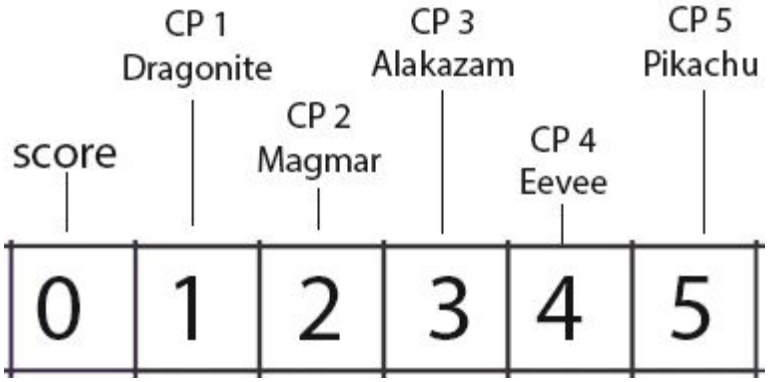

Imagem: Representação do vetor Pokédex.

Sobre as funções nestes arquivos, tem-se *player_data()* que lê a string de informações do jogador dado na entrada, nome e posição, e as salva no TAD player. Deve-se resaltar que devido o uso da função *strtok* para a divisão da string, a informação obtida em (i = 1) será lixo, pois refere-se a informações entre dois tokens escolhidos de quebra da string.

A função chamada de explorar(vizinhos) é representada neste trabalho por *explore().* Na visualização do player das casas possíveis de locomoção, considerou-se sua posição inicial como o conjunto ordenado (i, j), e cada uma das 8 possibilidades de movimento como variações destas coordenadas.

i+1, j-1	i+1, j	i+1, j+1
i, j-1	i, j	i, j+1
i-1, j-1	i-1, j	i-1, j+1

Imagem: Coordenadas de cada possibilidade de locomoção.

No caso do mapa geral, foi pensado em 8 diferenciações padrões do movimento do player, em que ele seja limitado pelos lados e quinas da matriz. Os quadrados em preto representam locais em que o player possa estar e em roxo, os possíveis movimentos.

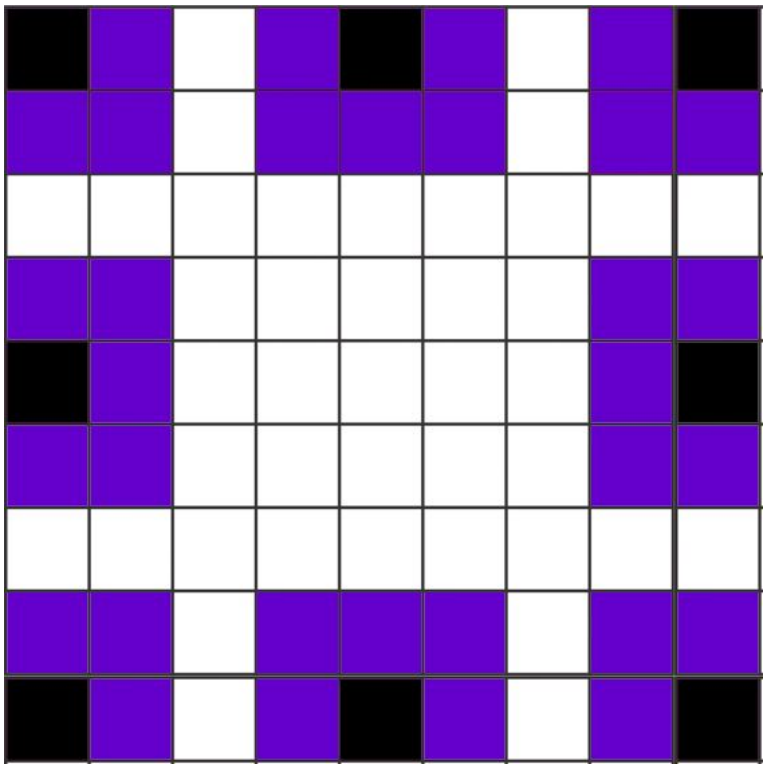

Imagem: Mapa com limitações de movimentação do player.

Nos quadrantes que o player não terá acesso, não foi permitida sua movimentação, além de preencher a matriz da vizinhança (*neighbourhood*) com o valor 10. Este valor foi escolhido arbitrariamente, apenas para preencher a matriz neighbourhood, considerando um valor em que o player nunca irá interagir, já que o maior número de identificação da célula é o 6, referente ao Pokéstop.

A função *where_to()* faz uma busca a fim de estabelecer qual a ordem de prioridade de célula para locomoção do Player, escolhendo sempre a de mais alto valor até 5 – Pokémon de mais alto CP – ou 6 se ele não tiver nenhuma Pokéball. Foi criada uam variável com valor muito baixo para fazer todas comparações e guardar qual a célula da matriz de possibilidades de locomoção é a melhor escolha para o jogador, salvado, então, o maior valor e o par ordenado de localização.

Andar(vizinhanca) foi implementada como *walkthrough().* Esta função faz o player andar até a célula julgada melhor por *where_to()* e executa as consequências desta escolha, como ganho de Pokéball, ganho ou perda de score e contagem de Pokémon de cada tipo. Esta função também salva o caminho feito pelo jogador, salvando em dois vetores os valores para x e y, como os pares ordenados de cada célula visitada.

A função *caminho_percorrido()* foi chamada de print_path(). Esta função une dois vetores do TAD player, que salva as informações do caminho feito e os une em um novo vetor, de forma que siga a ordem do par ordenado (x, y) de cada célula visitada.

Já *score_winner()* é a função que compara os scores de todos os jogadores alocados no vetor. Lê-se a primeira posição do vetor Pokedex, que foi, como já mencionado, separada para a contagem de pontos durante o jogo. Isto encerra as funcionalidades nos arquivos player(.c e .h).

Maps

Os arquivos relacionados ao mapa são maps.h e maps.c. Foi decidido usar a ordenação do mapa como matriz, sendo que a função *map_fill()* lê o arquivo de entrada e coloca o valor de cada célula an ordenação da matriz.

Main

Este é o arquivo principal do programa. Por ele, abre-se e lê o arquivo de entrada e faz as chamadas para alocação da matriz e das informações iniciais dos jogadores. Chama as funções já citadas a fim de criar o fluxo do jogo e determinar os caminhos a serem percorridos, assim como o vencedor.

3. Estudo de Complexidade

map_fill() - a função percorre uma matriz de tamanho N*N por meio de dois loops. Desta forma, sua complexidade é O(n)*O(n) = O(n^2).

player_data() - esta função simplesmente divide e salva partes de uma string nas informações dos N jogadores. Sua complexidade é O(n)*O(1)*O(1)*O(1) = O(n).

explore() - é uma função que utiliza de certa força bruta para limitar as hipoteses de movimentação do jogador. Como há limitação para o teste dependendo da localizaçao do player, pode-se dizer que este tipo de força bruta tem complexidade do tipo O(nlogn).

where_to() - é uma função composta de dois loops e um if simples, de complexidade, respectivamente, O(n), O(n) e O(1). A multiplicação das três complexidades gera a complexidade total da função, O(n^2).

walkthrough() - esta função faz comparações simples e salva informações em um vetor. Todas estas funcionalidades têm complexidade O(1), sendo, então, esta a compelxidade da função total.

print_path() - faz duas comparações de 0 a N, e utuliza a função simples para imprimir o caminho. As contas para a complexidade total são O(n)+O(n)+O(1), totalizando em O(n).

score_winner() - Esta função possui um while que faz comparações entre todos jogadores, sendo então sua complexidade como O(n), em que se tem N jogadores e dentro há um outro loop de 0 a N jogadores, totalizando a complexidade de O(n^2) da função.

main() - a função main é gestora de todas as outras funções, fazendo suas chamadas e utilizações. Desta forma, a complexidade é a soma de todas as funções utilixadas ao longo do código, onde prevalece então, a compelxidade mais alta. Desta forma, O(n^2).

4. Conclusão

O trabalho não foi finalizado. Houve certa dificuldade em implementar algumas das operações e fazer a relação delas no main. Embora a lógica para soluções dos problemas não tenha sido difícil de fato, ocorreu desorganização para a criação do trabalho como um todo, levando a entrega incompleta.

Considerando isto, não houveram testes, mas sim a implementação de todas funções necessárias para o decorrer do jogo.

5. Bibliografia

https://www.tutorialspoint.com;
http://www.cplusplus.com.

6. Anexos

O programa enviado constitui de:
player.c
player.h
maps.c
maps.h
main.c