Universidade do Algarve

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informático

Programação Orientada Por Objetos

Relatório Final

Discente: Hugo Paixão, nº 64514

Docentes: Hélder Daniel

Índice

1.Nota:

A estratégia probabilistic não foi desenvolvida.

2. Responsabilidades das Classes

SpotTheShips (client):

- Scanner sc: ler input - Variável global.
- List <String> strategies: estratégias possíveis - Variável global.
- List <char> orientations: orientações possíveis - Variável global.
- Main(): recebe input gera a lista inicial, seleciona a estratégia indicada e envia os dados para output.

InputCheckOutput:

- getCheckStrategy(): lê a String que identifica a estratégia a usar do input e verifica se passa as condições necessárias para ser valida, caso contrário emite mensagem de erro e da exit no programa.
- getCheckIntegers(): lê todos os inteiros e verifica se passam nas condições necessárias para serem validos, caso contrário emite mensagem de erro e da exit no programa.
- getCheckOrientation(): lê todos os char que identificam a orientação dos ships e verifica se passa nas condições para ser valido, caso contrário emite mensagem de erro e da exit no programa.

- output(int arg1, int arg2, List<Position> arg3): faz o output, faz o print do resultado do problema.

Position:

- Position(int arg1, int arg2, char arg3): construtor da classe, contém posição (x, y) e o valor da posição;
- getX(): obtém o valor de x;
- getY(): obtém o valor de y;
- getType(): obtém o valor da posição (x, y);
- setType(char arg): altera o valor da posição (x, y).
- equals(Object o):
- hashCode():

MatrixAndList:

- fillStrategies(): coloca todas as estratégias possíveis na lista strategies.
- fillOritentations(): coloca todas as orientações possíveis na lista orientations.
- fillMatrix(char[][] arg1, int arg2, int arg3): enche uma matriz char com '.'.
- placeShips(char[][] arg1, int[] arg2, int [] arg3, int [] arg4, char[] arg5): verifica a orientação dos ships e envia dependendo dessa orientação para uma das funções para colocar os ships na matriz.
- placeShipsNS(char[][] arg1, char arg2, int arg3, int arg4, int arg5): coloca os ships na matriz se estes forem de orientação 'N' ou 'S'.
- palaceshipsOE(char[][] arg1, char arg2, int arg3, int arg4, int arg5): coloca os ships na matriz se estes forem de orientação 'O' ou 'E'.
- putMatrixInList(): coloca matriz completa já com os ships numa lista passando todas as posições para Object Position.

ISpotTheShips (interface):

- solve(): seleciona a estratégia a seguir;
- noScans(): número de scans necessários para encontrar todos os ships.

Linear:

- int noScans: número de scans necessários - Variável global.
- solve(): implementa a estratégia relativa a esta abordagem.
- noScans(): retorna o número de scans necessários para encontrar todos os ships.

Smart:

- int i: variável para incrementar as posições da lista - Variável global.
- int j: variável para incrementar o número de scans desnecessários- Variável global.
- int k: variável que identifica a posição abaixo de cada posição - Variável global.
- int var: número de scans abaixo de cada posição - Variável global.
- int count: número total de ships - Variável global.
- int noScans: número de scans necessários - Variável global.
- int column: número de colunas - Variável global.
- List <Position> scanList: - Variável global.
- solve(): implementa a estratégia relativa a esta abordagem.
- markdiagonals(): marca as diagonais de uma posição que contenha um ship.
- checkEast(): verifica a posição da direita caso seja encontrado um ship.
- checkSouth(): verifica a posição abaixo de uma posição que contenha um ship.
- noScans(): retorna o número de scans necessários para encontrar todos os ships.

3.Padrões de design

solve() – seleciona a estratégia a seguir.

4.Resolução

Na class SpotTheships contem 2 listas uma para as estratégias possíveis e outra para as orientações possíveis. E o método main onde está todo o input lido caso este passe aos testes necessários para ser considerado válido. Antes de começar a ler o input as listas com as estratégias e as orientações validas são criadas e construídas na class MatrixandList.

A leitura do input e o print do output é feito através da class InputCheckOutput, que contem 3 métodos para leitura de input e um método para output. Os métodos de input verificam se a estratégia lida é valida se nenhum dos inteiros dado pelo utilizador é negativo e se as orientações dos ships são validas. Após a leitura da estratégia e o tamanho do board e o número de ships, é criada uma matriz que servirá como board e são criados 4 arrays, 3 de inteiros e 1 char, os int contem a row, a column e o tamanho do ship e o char tem a orientação deste. Após a criação e inicialização dos 4 arrays são lidas as posições e as orientações dos ships.

Quando acaba a leitura de dados os voltamos à class MatrixAndList para colocar os ships na matriz, após isso a matriz é passada para uma lista de objetos Position, da class Position que contem a row(x), column(y), e o valor da posição da matriz(type). Após a lista estar criada, é decidida que estratégia usar comparando as opções com o input lido.

Depois segundo o input a estratégia decide a class que vai ser usada, class Linear, Smart ou Probabilistic. Após a resolução a lista final é enviada para a class InputChekOutput para dar output do resultado.

5.Diagrama UML