Trabalho Prático - Programação Estruturada e Orientada a Objetos

Projeto e Implementação

Existe um exercício da inteligência Artificial bastante conhecido chamado *Mundo Wumpus* que consiste em um jogo onde o jogador (um agente autônomo) deve entrar em uma caverna onde mora o terrível monstro Wumpus, que guarda um grande tesouro.

O objetivo do jogador é conseguir o ouro, matar o Wumpus com uma flecha e voltar à entrada da caverna, desviando dos buracos e tentando não ser devorado pelo Wumpus.

Em nosso trabalho, você e sua equipe terão de desenvolver um versão diferente do mundo Wumpus baseado no anime e manga de sucesso **One Piece**. O ambiente do exercício será a um oceano chamado **Grand Line** e um **pirata** será o nosso agente autônomo (sem muita inteligência). O pirata deseja carregar o máximo possível do tesouro **One Piece** presente na ilha chamada *Raftel*, que está no fim da **Grand Line**, até o início da **Grand Line**. Contudo, a **Grand Line** possui condições climáticas e correntes marítimas totalmente imprevisíveis o que dificulta a navegação neste oceano para qualquer um que queira se aventurar por lá. Além disso, a força militar da Marinha está cada vez mais ativa na **Grand Line** devido à grande quantidade de piratas neste oceano. Felizmente o **pirata** conseguiu desenvolver seu Haki da Cor de Observação devido ao seu arduo treinamento, que o permite sentir quando está próximo ao **One Piece** ou à **Marinha**.

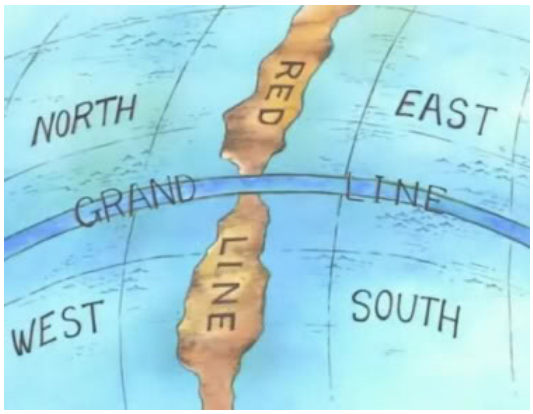

Figura 1 – *Grand Line*

Com base na descrição anterior, faça o que se pede:

(a) Crie uma exceção chamada AcaoInvalidaException que informa na mensagem qual ação foi inválida. Nos itens “l” e “n” são informados quais são estas ações inválidas. Use superconstrutores para indicar mensagens específicas para ações inválidas específicas.

(b) Crie um enumerador chamado TipoElemento, que contém os valores “Pirata”, “Marinha”, “EspacoVazio”, “Obstaculo” e “One Piece”.

(c) Crie uma classe abstrata chamada Elemento que contém dois atributos, uma String nome e um TipoElemento tipo. Crie um construtor que inicializa os atributos dessa classe. Crie também métodos de get e set para os atributos.

(d) Crie uma classe abstrata Pessoa que herda de Elemento e que contém o atributo inteiro hp. Crie um construtor que inicializa os atributos nome, tipo e hp. Lembre-se de usar o método super para inicializar os atributos nome e tipo no construtor. Crie também os métodos get e set para o atributo hp.

(e) Crie uma classe Pirata que herda de Pessoa. Esta classe possui os atributos peso e pesoadicional (inicialmente zero). Crie um construtor que inicializa os atributos nome, tipo (Pirata), hp e peso. Lembre-se de usar o método super para inicializar os atributos nome, tipo e hp no construtor. Crie também os métodos get e set para os atributos peso e pesoadicional.

(f) Crie uma classe Marinha que herda de Pessoa. Essa classe possui o atributo estado que pode assumir os valores “alerta” ou “dormindo”. Crie um construtor que inicializa os atributos nome, tipo (Marinha), hp e estado. Lembre-se de usar o método super para inicializar os atributos nome, tipo e hp no construtor. Crie também os métodos get e set para o atributo estado.

(g) Crie uma classe One Piece que herda de Elemento. Essa classe possui o atributo peso.

Crie um construtor que inicializa os atributos nome, tipo (One Piece) e peso. Lembre-se de usar o método super para inicializar os atributos nome e tipo no construtor. Crie também os métodos get e set para o atributo peso.

(h) Crie uma classe Grand Line. A classe Grand Line possui o atributo cenario que é uma lista de listas de elementos (Vector<Vector<Elemento>>). Crie um construtor que inicializa a lista externa (new Vector<Vector<Elemento>>( )). Crie também os métodos get e set para o atributo cenário.

(i) Crie um enumerador Movimento, que contém os valores “up” (1), “down” (2), “left” (3), “right” (4).

(j) Crie um método Inicializar em Grand Line que inicializa o cenário. Ele recebe como argumento um valor inteiro n que define o tamanho máximo do cenário (ou seja, no máximo o cenário terá n x n Elementos). Na posição (0,0) deve haver um Pirata e na posição ($n-1$, $n-1$) deve haver um Elemento do tipo One Piece (O Pirata deve ter um peso menor que o do One Piece e o peso deve ser em quilogramas). O cenário deve ter também uma Marinha que está dormindo e está em uma posição gerada aleatoriamente e diferente da posição do Pirata e do One Piece. Por fim, deve haver também $n/2$ Elementos Obstaculo em posições geradas aleatoriamente em que não estejam nem o Pirata, nem o One Piece e nem a Marinha. Garanta também que sempre haverá um caminho entre o Pirata e o One Piece ao inicializar o cenário. Todos as posições anteriores ao de um Pirata, One Piece, Marinha ou Obstaculo em cada

linha devem ser preenchidos com Elementos do tipo EspacoVazio. Todas as posições posteriores ao último Elemento Pirata, One Piece, Marinha ou Obstaculo em uma linha não possui elementos. Ou seja, cada linha tem no mínimo 0 e no máximo $n-1$ Elementos.

(k) Crie um método visualizarCenario em Grand Line. Esse método deve imprimir na tela os elementos do cenário e remover os valores de linhas que tenham apenas Elementos do tipo EspacoVazio. Para tal, crie uma variável booleana que recebe valor verdadeiro se a linha tiver algum elemento que não seja do tipo EspacoVazio e falso caso contrário. E limpa os elementos da linha caso a variável tenha valor falso (você pode usar o método clear() do Vector para limpar a linha). Só devem ser impressos na tela os Elementos do tipo Pirata, Marinha e One Piece.

(l) Crie um método mover Pessoa, que recebe como parâmetros a Grand Line onde a Pessoa está e um Movimento, e altera a posição do Pessoa da seguinte forma:

- “up” move a Pessoa dentro do cenário no eixo y em uma posição acima.
- “down” move a Pessoa dentro do cenário no eixo y em uma posição abaixo.
- “right” move a Pessoa dentro do cenário no eixo x em uma posição para a direita.
- “left” move a Pessoa dentro do cenário no eixo x em uma posição para a esquerda.

A Pessoa pode se mover apenas dentro da Grand Line para uma posição que possua um Elemento EspacoVazio, ou para uma posição que ainda não possui nenhum Elemento. Nesse segundo caso, deve ser adicionado os Elementos necessários (A Pessoa deve ser adicionada à posição para o qual ele se moveu e, se não houver, devem ser criados Elementos do tipo EspacoVazio nas posições anteriores a ele na linha). A Pessoa pode ainda se mover para uma posição que possua um Elemento One Piece se ele for um Pirata, ou no caso da Pessoa seja a Marinha, ele também pode se mover para a mesma posição em que está a Pessoa do tipo Pirata. Vale ressaltar, que ao se mover a posição anterior da Pessoa recebe um Elemento do tipo EspacoVazio.

Nos seguintes casos o movimento não poderá ocorrer, ou seja, ao tentar se movimentar a Pessoa não sairá do lugar:

- Caso a Pessoa tente se mover para uma posição fora da Grand Line (tentar se mover para uma posição com coordenadas x ou y vazias ou com coordenadas x ou y maior que o tamanho das dimensões do cenário -1, ou seja, maior que $n-1$). Trate a exceção AcaoInvalidaException, retornando que o movimento é inválido porque a Pessoa (use o nome da Pessoa) está tentando sair da Grand Line;
- Caso a Pessoa tente se mover para uma posição onde há um obstáculo. Trate a exceção AcaoInvalidaException, retornando que o movimento é inválido porque a Pessoa está tentando atravessar um obstáculo;

- Finalmente, caso a Pessoa seja um Pirata, ele saberá se há alguém da Marinha na posição ao lado e deverá se mover para uma posição diferente da que o marinheiro está.

Se a Pessoa for um Pirata e ele estiver a uma ou duas posições de distância do One Piece e não houver nenhum Obstaculo ou Marinha no caminho, ele saberá exatamente para que direção seguir. Em outras palavras, se um Pirata estiver nas posições ($n-1$, $n-3$) ou ($n-3$, $n-1$) e não houver Obstaculo ou Marinha no caminho (posições ($n-1$, $n-2$) ou ($n-2$, $n-1$)) ele saberá qual direção seguir para chegar ao One Piece. Se um Pirata se choca contra um Obstaculo, a Marinha acorda e pode se mover na posição oposta ao Pirata (ou seja, se o Pirata usa o método mover recebendo Down e a Marinha está acordada, ele se move recebendo UP. De forma similar, se o Pirata usa o método mover recebendo Left e a Marinha está acordada, ele se move recebendo Right e assim por diante) por duas rodadas, depois ele volta a dormir. Após cada movimento (inclusive quando o movimento for inválido), chame o método visualizarCenario da Grand Line.

(m) Crie na classe Pirata um método que verifica se o Pirata encontrou o One Piece (está na mesma posição) e retorna um booleano.

(n) Crie na classe Pirata um método carregar que adiciona 10 kg ao pesoadicional do Pirata se ele estiver sobre o One Piece e retira 10 kg do peso do One Piece. O pesoadicional não pode superar o peso do Pirata. Caso o Pirata tente carregar mais que o seu peso, a ação deve ser impedida e uma exceção do tipo AcaoInvalidaException deve ser disparada com a mensagem “Não é possível carregar mais One Piece”. Caso o peso do One Piece chegue a zero, o Elemento One Piece deve ser removido da Grand Line e o novo Rei dos Piratas deve ser anunciado.

(o) Crie na classe Pirata também o método descarregar, que esvazia o pesoadicional do Pirata apenas se ele estiver na posição (0,0). Caso contrário nada acontece.

Crie uma classe Main que inicializa uma Grand Line com dimensões máximas 7x7. O algoritmo deve continuar executando até o Pirata presente na Grand Line conseguir mover todo o One Piece para o início da Grand Line (Ele tem de descarregar todo o One Piece) ou até ele ser derrotado pela Marinha (caso a Marinha se mova para a mesma posição do Pirata). Os movimentos do Pirata devem ser aleatórios a não ser que o Pirata sinta a presença do One Piece ou da Marinha próximos. Quanto ao One Piece, o Pirata tenta carregar o máximo possível e somente para com essa ação se uma exceção informando que não é possível carregar mais peso é disparada ou se não houver mais One Piece. Caso a Marinha esteja acordada e ainda tenha uma ação de movimento antes de dormir e o Pirata esteja carregando ou descarregando o One Piece, a Marinha se moverá aleatoriamente. Ao final do código deve ser impressa a configuração final da Grand Line (basta usar visualizarCenario mais uma vez ao fim do código).

Documentação

Elaborar um relatório sobre o projeto da primeira parte deste trabalho. Esse relatório deve conter o nome dos autores e a documentação de cada classe, com exceção das classes main.

A documentação das classes deve seguir o modelo abaixo para cada classe:

1. Informar o nome da classe
2. Informar a data de criação da classe
3. Informar os atributos da classe e o que indicam (ex.: O atributo HP representa o nível de vida do personagem)
4. Informar para cada método:
    a) Nome do método
    b) Atributos de entrada
    c) Saída esperada
    d) O que o método faz

    * Observação:

Não é preciso apresentar os métodos get e set no relatório. Lembre-se que um método que adiciona um elemento em um atributo matriz ou Vector não é um método set. O método set atribui um valor do tipo do atributo, nesses casos, uma matriz ou um Vector.

Observações Finais:

- O nome de todas as classes deve obrigatoriamente inicializar com letras maiúsculas.
- O nome de todos os métodos e os atributos deve começar obrigatoriamente com letras minúsculas.
- Todos os atributos de todas as classes devem ser private.
- Todos os métodos de todas as classes devem ter a indicação de algum modificador de acesso (public, protected, private).
- Os atributos da superclasse não devem ser copiados em suas subclasses (sobescrita de atributos).
- Não há métodos a serem sobrescritos, embora no item i haja a sobrecarga de um método.
- Qualquer violação de algum dos itens anteriores acarretará no decréscimo de pontos.