Escola de Artes, Ciências e Humanidades - EACH-USP

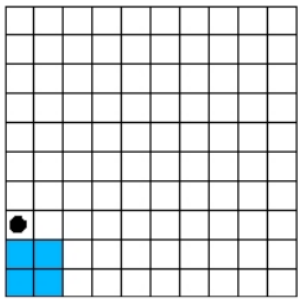
ACH2002 - Introdução à Ciência da Computação II

EACH - Segundo Semestre de 2010

Segundo Exercício Prático - Data de entrega: até 31 de outubro de 2010

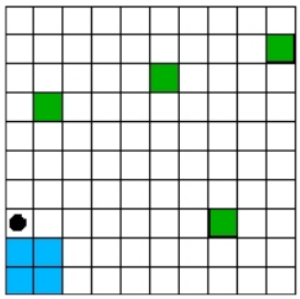
ROBÔ EM UMA SALA

Você agora é o responsável por programar o comportamento de um robô em uma sala. Imagine seu robô (a bolinha preta) em uma sala de 10 × 10 passos, como esta:

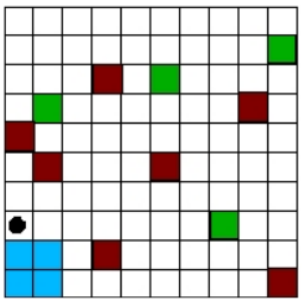


Seu robô é livre para se movimentar tanto verticalmente quanto horizontalmente, mas não nas diagonais. Ou seja, a cada momento ele pode escolher entre no máximo 4 posições para caminhar.

Nesta sala, a área azul é chamada de Área de armazenagem, e será usada para armazenar 4 blocos que estarão espalhados pela sala (e cuja posição o robô **desconhece**). Os blocos (em verde) podem estar em qualquer lugar, sendo sua posição recebida como parâmetro. Dessa forma, uma possível configuração seria:



Seu robô é livre para se movimentar pela área de armazenagem, inclusive pelos quadrados dos blocos, ou seja, os blocos não impedem o trânsito dele. Na sala (fora da área de armazenagem) pode haver alguns obstáculos (em vermelho), cuja quantidade e posição são parâmetros de entrada ao sistema. Por exemplo:

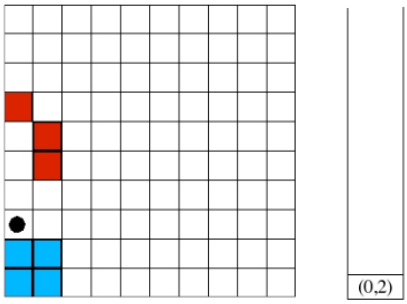


Desta vez o robô não pode passar por casas com obstáculos. Sendo assim, casas com obstáculos não podem ser consideradas quando da escolha dos movimentos do robô. Sua tarefa é, então, escrever um programa que receba as posições dos 4 blocos, além do número e posições dos obstáculos (a quantidade de

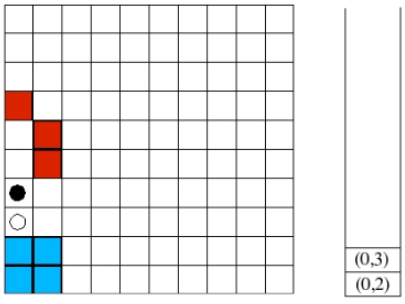
obstáculos é deixada em aberto). Seu programa deve fazer o robô andar pela sala em busca dos blocos, evitando os obstáculos. O tamanho da sala é sempre 10 × 10. Cada ação executada pelo robô deve ser impressa em tela, no formato definido mais adiante.

Ao chegar em uma casa onde exista um bloco, o robô deve coletá-lo, removendo-o dessa casa (para isso ele entra na casa, ou seja, também aqui os blocos não atrapalham seu movimento). Ele então deve voltar imediatamente à área de armazenagem, seguindo de volta seus passos (o robô somente carrega um bloco por vez). Ao voltar à posição de onde começou a busca (ao lado da área de armazenagem), carregando um bloco, o robô deve descarregar o bloco dentro da área. Para tal, não há a necessidade de movimentar o robô dentro da área. Você pode supor que há um serviço automático de armazenagem, a quem o robô entrega o bloco. Esse serviço de armazenagem, por sua vez, guarda os blocos nas seguintes posições (tomadas em ordem): (0,0), (1,0), (0,1) e (1,1). Dessa forma, o trajeto do robô compreende duas fases: uma busca por blocos, que ocorre na sala (fora da área de armazenagem) e seu armazenamento final (dentro da área).

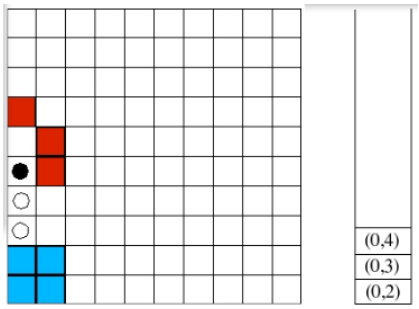
A cada movimento, o robô deve escolher para onde ir. Para essa escolha, o conceito que iremos aplicar se chama busca em profundidade. Sua idéia básica é razoavelmente simples: o robô anda preferivelmente em uma mesma direção, marcando seus passos. Se não for mais possível andar nesta direção, ele escolhe a próxima de sua preferência. Caso não seja mais possível andar em nenhuma nova direção, ele volta um passo, escolhendo novamente em que direção ir (dentre as ainda não visitadas). Caso mais uma vez não haja movimento possível, ele deve dar mais um passo atrás, e assim por diante. Como exemplo, considere a seguinte situação:



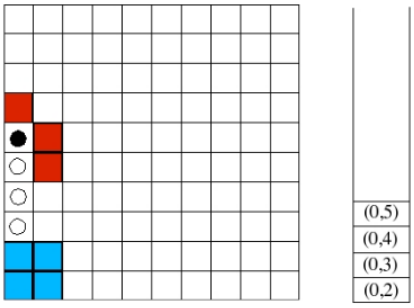
À esquerda temos o mapa da sala, enquanto que à direita temos uma estrutura que marca o caminho percorrido pelo robô ((x,y) cartesianos). Agora suponha que nosso robô prefere a seguinte ordem de direções para ir (ordem esta que também você deverá seguir): cima, direita, baixo, esquerda, em ordem. Ou seja, ele sempre tenta ir para cima, caso não possa, tenta à direita, e assim por diante. No caso acima, ele segue sua preferência e vai à casa imediatamente acima, armazenando sua nova posição na estrutura, acima do dado que lá estava:



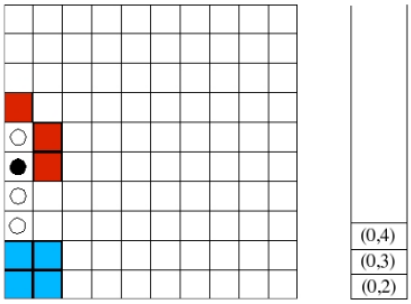
Note que, além de armazenar a posição, ele também marca os quadros por onde passa (bola branca). Novamente, ele se move para cima, já que não há impedimentos, armazenando essa nova posição na estrutura:



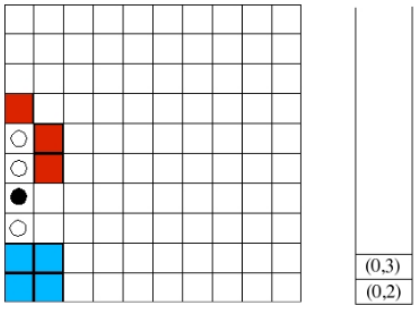
E mais uma vez acima...



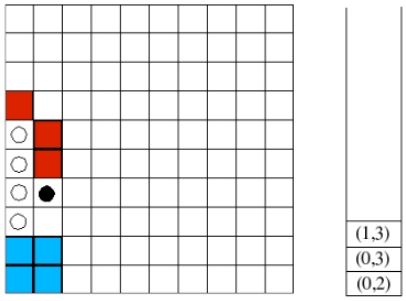
Aqui ele tenta ir para cima, para a direita, para baixo e para a esquerda, mas não há casa livre para onde ir (note que a de baixo não está livre por ele já ter passado por ela). Então ele volta uma casa e continua a busca de sua posição anterior. Para saber que posição era esta, ele deve remover da estrutura sua posição antiga – (0,5) – e olhar a última coordenada que lá ficou armazenada – (0,4):



Novamente ele tenta todas as direções e, como não obtém sucesso, executa o backtracking mais uma vez, removendo (0,4) – sua posição atual – da estrutura, e indo à posição anterior, ou seja, (0,3):



Aqui ele tenta ir para cima sem sucesso, porém a casa à direita está vazia, então ele se move para lá, armazenando a nova posição:



E assim vai, até achar um bloco ou percorrer a sala toda (caso em que volta à área de armazenagem, avisando que não encontrou nada). Ao encontrar um bloco (ou percorrer a sala toda e não achar nenhum), o robô volta à área de armazenagem (azul) seguindo sua própria trilha, e limpando as marcações que fez pela sala. Assim, ao recomeçar a fase de busca, tanto a estrutura com o caminho atualmente percorrido, quanto as marcações na sala são limpos.

Essa estrutura auxiliar, em que novos elementos são adicionados ao seu topo, enquanto que, para retirarmos algo, devemos fazê-lo apenas desse topo, é conhecida como pilha. Você, contudo, não precisa implementá-la explicitamente. Basta, para tal, construir uma versão recursiva para a fase da busca. A grande vantagem, nesse caso, é que, durante a recursão, o computador acaba usando uma pilha. Assim, além do código ficar mais legível, você ainda poupa o tempo de criar a estrutura e gerenciá-la. Isso, contudo, é apenas uma sugestão de implementação.

Nesse projeto, a área de armazenagem compreende a área delimitada por (0,0) e (1,1), sendo que o robô inicia sua busca a partir da posição (0,2). Ao finalizar a busca, o robô deve alertar o usuário (via mensagem apropriada). Uma mensagem de erro também deve ser enviada caso os 4 blocos não possam ser encontrados.

No quadro abaixo, você encontra as classes e interfaces a serem implementadas. As classes hachuradas são aquelas que você precisará implementar/completar:

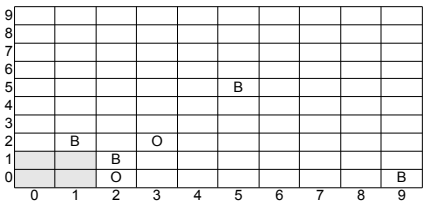
Nome	Tipo	Resumo
IRobo	interface	Interface que contém a assinatura dos métodos que definem um robô e seus movimentos.
ISala	interface	Interface que contém a assinatura dos métodos que definem a sala onde a busca será executada.
Mensageiro	classe	Classe com os métodos a serem utilizados para envio de mensagens ao usuário. Não passe nenhuma mensagem, além das previstas nessa classe.
Sala	classe	Classe que define a sala onde será feita a busca. Implementa a interface ISala (cabe a você fazer essa implementação).
Robo	classe	Classe Robo que implementa o robô e a busca que ele faz. Implementa a interface IRobo (cabe a você fazer essa implementação).
TestaRobo	classe	Classe que testa a busca desenvolvida. Serve como exemplo de como o robô deve ser executado. Use para seus testes apenas.

A descrição detalhada de cada classe (javadoc), bem como a implementação das classes fornecidas, você encontra no arquivo ep2.zip, disponibilizado na página da disciplina. Esse arquivo contém 2 diretórios:

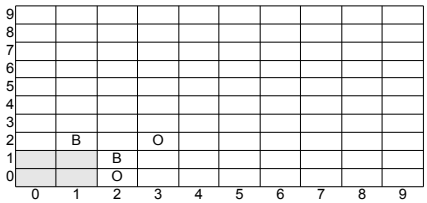
- ep2: com as classes e interfaces java a serem utilizadas. Além disso, o diretório também contém dois exemplo de saída (Exemplo_ok.txt, Exemplo_falha1.txt e Exemplo_falha2.txt), que ilustram como seria a busca feita nas seguintes condições (para ver a grande diferença entre eles, olhe a última

mensagem enviada ao usuário, em cada um desses arquivos):

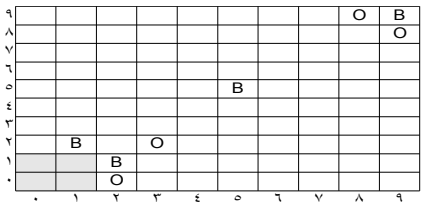
- Para o exemplo em Exemplo_ok.txt, a seguinte configuração de blocos e obstáculos:



- Para o exemplo em Exemplo_falha1.txt, a seguinte configuração de blocos e obstáculos (note que não foram espalhados 4 blocos, por isso a falha):



- Para o exemplo em Exemplo_falha2.txt, a seguinte configuração de blocos e obstáculos (note que um dos blocos está inacessível, por isso a falha):



- Toda vez que o robô tenta ir à casa em que se encontra o obstáculo ele gera um aviso de "Obstáculo detectado";
- Dois blocos, presentes em (2,1) e (1,2). Toda vez que o robô entra em uma casa em que se encontra um bloco ele gera um aviso de "Bloco recolhido", simulando que o recolheu. Note que, com blocos, o robô efetivamente entra na casa, como indica o aviso "Busca em" nas mesmas coordenadas, imediatamente antes do "Bloco recolhido". Isso não acontece com obstáculos, uma vez que o robô não consegue ir àquela casa.

- ep2_doc: com a documentação gerada pelo javadoc (inicie em index.html).

Com relação às mensagens enviadas pelo robô, já vimos "Obstáculo detectado" e "Bloco recolhido". Além destas, há também:

- "Busca em": gerada toda vez que o robô entra em uma determinada casa para procurar um bloco;
- "Retorno a": gerada toda vez que o robô, **após ter encontrado o bloco**, faz seu caminho de volta à área de armazenagem. Note que essa mensagem não é gerada toda vez que ele faz o backtracking mas, em vez disso, apenas quando se dirige de volta à área de armazenagem seguindo seus passos;
- "Bloco armazenado": gerada toda vez que o robô chega de volta ao local de onde começou a busca, carregando um bloco, e desejando armazená-lo na área de armazenamento;
- "Nenhum bloco encontrado": gerada quando não há mais blocos a serem buscados, ou porque não havia nenhum bloco na sala, ou então porque o bloco está inacessível, dependendo de como estão dispostos os obstáculos.
- "Busca terminada. Blocos recuperados": gerada ao final da coleta dos quatro blocos.

Regras de Elaboração e Submissão

Este trabalho é individual, cada aluno deverá implementar e submeter via Col sua solução.

A submissão será feita via um arquivo zip (o nome do arquivo deverá ser o número USP do aluno,

por exemplo 1234567.zip). Este arquivo deverá conter EXATAMENTE os seguintes arquivos: Sala.java e Robo.java. Observação: no arquivo zip não adicionar (sub-)diretórios ou outros arquivos, apenas esses dois arquivos. Note que estes arquivos são .java (e não .class). Além disso, o arquivo .zip **NÃO DEVE SER DESCOMPACTADO NO COL**. Trabalhos que infringirem esta regra não serão corrigidos.

Além deste enunciado, você encontrará na página da disciplina um arquivo zip contendo todos os arquivos envolvidos neste trabalho (note que os arquivos a serem implementados também estão disponíveis no site, você precisará apenas completá-los).

Qualquer tentativa de fraude ou cola implicará em nota zero para todos os envolvidos. Guarde uma cópia do trabalho entregue. A data de entrega é 31 de Outubro (domingo) às 23:50h pelo sistema COL, não serão aceitos trabalhos entregues após esta data (não deixe para submeter na última hora).

Não implemente métodos desnecessários. Não utilize conceitos ou classes que não foram estudados em aula. Só complemente a implementação das classes solicitadas.

Caso o EP não compile a nota do trabalho será zero. É importante que você teste seu trabalho executando a classe TestaRobo (note que ela não testa todas as funcionalidades de todas as classes). Preencha o cabeçalho existente no início do arquivo Robo.java (há espaço para se colocar turma, nome do professor, nome do aluno e número USP do aluno). Todas as classes pertencem ao pacote ep2.