Licenciatura em Engenharia Informática e de Computadores

Fundamentos de Programação

Projecto - Primeira Parte

2 de Novembro de 2010

O objectivo da primeira parte do projecto é a definição e implementação de alguns dos tipos abstractos de informação que serão usados na realização do projecto final.

Os tipos implementados nesta primeira parte vão ser usados não só pelo programa que desenvolverá na segunda parte, como também por outros programas, nomeadamente o programa de avaliação automática. Assim, deve respeitar escrupulosamente as definições que são dadas de seguida, inclusivamente no que diz respeito aos nomes dos procedimentos, bem como ao número e ordem dos parâmetros formais.

Para além dos tipos abaixo descritos, deverá incluir no seu código a implementação do tipo lista simplificada apresentada no livro; não precisa, no entanto, de incluir a definição abstracta do tipo lista simplificada no seu relatório.

Para não sobrecarregar desnecessariamente o código, apenas os construtores de cada tipo devem verificar a validade dos argumentos que recebem.

1 O tipo posição

Uma posição é constituída por dois inteiros, entre 0 e 7, correspondentes a uma linha e uma coluna de um tabuleiro de jogo 8×8. O tipo posição deve disponibilizar as seguintes operações básicas:

1. Construtor.
   - $faz\text{-}pos : inteiro \times inteiro \mapsto posição$
     $faz\text{-}pos(l, c)$ devolve a posição correspondente à linha l, coluna c.
2. Selectores.
   - $linha\text{-}pos : posição \mapsto inteiro$
     $linha\text{-}pos(p)$ devolve a linha da posição p.
   - $coluna\text{-}pos : posição \mapsto inteiro$
     $coluna\text{-}pos(p)$ devolve a coluna da posição p.
3. Reconhecedor.
   - $pos? : universal \mapsto lógico$
     $pos?(arg)$ tem o valor $verdadeiro$, se arg é uma posição, e tem o valor $falso$, em caso contrário.
4. Teste.

- $pos =? : posição \times posição \mapsto lógico$
  $pos =?(p_1, p_2)$ tem o valor $verdadeiro$, se p_1 e p_2 são posições iguais, e tem o valor $falso$, em caso contrário.

Devem também ser disponibilizadas as seguintes operações de alto nível:[1]

- $distancia : posição \times posição \mapsto real$
  $distancia(p_1, p_2)$ devolve a distância entre as posições p_1 e p_2. Se $p_1 = (l_1, c_1)$ e $p_2 = (l_2, c_2)$, então $distancia(p_1, p_2) = \sqrt{(l_1 - l_2)^2 + (c_1 - c_2)^2}$.
- $mesma\text{-}direccao? : posição \times posição \mapsto lógico$
  $mesma\text{-}direccao?(p_1, p_2)$ tem o valor $verdadeiro$, se p_1 e p_2 estão na mesma linha, ou na mesma coluna, e tem o valor $falso$, em caso contrário.
- $adjacentes? : posição \times posição \mapsto lógico$
  $adjacentes?(p_1, p_2)$ tem o valor $verdadeiro$, se p_1 e p_2 são posições adjacentes, e tem o valor $falso$, em caso contrário. Duas posições são adjacentes se se encontram na mesma direcção e a distância entre elas é um.
- $adjacentes : posição \mapsto lista\ de\ posições$
  $adjacentes(p)$ devolve uma lista cujos elementos são as posições adjacentes à posição p. Por exemplo, $adjacentes(faz\text{-}pos(0, 0))$ devolve a lista $((1, 0)(0, 1))$, e $adjacentes(faz\text{-}pos(1, 2)$ devolve a lista $((2, 2)(0, 2)(1, 3)(1, 1))$.

2 O tipo conteúdo

Um elemento do tipo conteúdo é um dos seguintes símbolos: *p*, *b*, *v*. Os elementos deste tipo são usados para representar os conteúdos das posições de um tabuleiro do jogo das damas. O símbolo *p* representa uma peça preta, o símbolo *b* representa uma peça branca, e o símbolo *v* indica que a posição correspondente do tabuleiro está vazia.

3 O tipo jogada

Os elementos do tipo jogada são constituídos por duas posições: a primeira representa a posição da peça antes da jogada, o *início da jogada*, e a segunda representa a posição da mesma peça depois da jogada, o *fim da jogada*.

O tipo jogada deve disponibilizar as seguintes operações básicas:

1. *Construtor*:
   - $faz\text{-}jogada : posição \times posição \mapsto jogada$
     $faz\text{-}jogada(p_1, p_2)$ devolve uma jogada com início em p_1 e fim em p_2.

[1] Tenha em atenção que estas operações devem ser independentes da representação interna escolhida.

2. *Selectores*:

   - $inicio\text{-}jogada : jogada \mapsto posição$
     $inicio\text{-}jogada(j)$ devolve o início da jogada j.
   - $fim\text{-}jogada : jogada \mapsto posição$
     $fim\text{-}jogada(j)$ devolve o fim da jogada j.

3. *Reconhecedores*:

   - $jogada? : universal \mapsto lógico$
     $jogada?(arg)$ tem o valor $verdadeiro$, se arg é uma jogada, e tem o valor $falso$, em caso contrário.
   - $jogada\text{-}nula? : jogada \mapsto lógico$
     $jogada\text{-}nula?(j)$ tem o valor $verdadeiro$, se j é uma jogada nula, e tem o valor $falso$, em caso contrário. Uma jogada é considerada nula se a posição de início e de fim da jogada são iguais. Uma jogada nula representa o facto de o jogador que a faz não poder mover mais peças.[2]

4 O tipo posições de jogada

O tipo posições de jogada representa as posições entre o início e o fim de uma jogada. Os conteúdos destas posições são determinantes para determinar se uma jogada é válida.

Um elemento do tipo posições de jogada é constituído por duas listas de posições. Uma das listas contém as posições que devem conter peças do adversário antes da jogada ser efectuada, peças essas que serão capturadas durante a jogada. A outra lista contém as posições que devem estar livres para a jogada poder ser efectuada.

O tipo posições de jogada deve disponibilizar as seguintes operações básicas:

1. Construtor.

   - $faz\text{-}posicoes\text{-}de\text{-}jogada :$
     $lista\ de\ posições \times lista\ de\ posições \mapsto posições\ de\ jogada$
     $faz\text{-}posicoes\text{-}de\text{-}jogada(pc, pl)$ devolve o elemento do tipo posições de jogada cujas posições de peças capturadas são as da lista pc, e cujas posições livres são as da lista pl.

2. Selectores.

   - $pecas\text{-}capturadas : posições\ de\ jogada \mapsto lista\ de\ posições$
     $pecas\text{-}capturadas(pj)$ devolve a lista de posições de peças capturadas de pj.
   - $posicoes\text{-}livres : posições\ de\ jogada \mapsto lista\ de\ posições$
     $posicoes\text{-}livres(pj)$ devolve a lista de posições livres de pj.

Deve também ser disponibilizada a seguinte operação de alto nível:

[2] Ou, eventualmente, se se tratar de um jogador inexperiente, de o jogador não encontrar nenhuma jogada para fazer, embora a jogada exista.

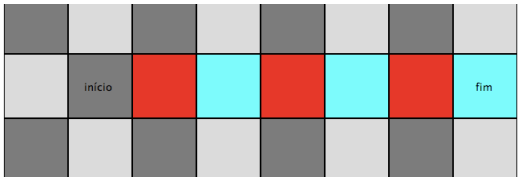

Figura 1: Posições entre o início e o fim de uma jogada. A vermelho, as posições com as peças do adversário que serão capturadas. A azul, as posições que devem estar livres.

- $posicoes\text{-}de\text{-}jogada : jogada \mapsto posições\ de\ jogada$

  $posicoes\text{-}de\text{-}jogada(j)$ devolve as posições de jogada da jogada j. Se j não for uma jogada, ou o início e o fim da jogada não estiverem na mesma direcção, ou ainda se a distância entre o início e o fim não for um inteiro par, $posicoes\text{-}de\text{-}jogada(j)$ devolve a lista vazia. Em caso contrário:

  – As posições de peças do adversário são a posição imediatamente a seguir ao início da jogada, e as restantes, de duas em duas posições, até ao fim da jogada, exclusive.

  – As posições livres são a posição duas posições a seguir ao início da jogada, e as restantes, de duas em duas posições, até ao fim da jogada, inclusive.

  Na figura 1 apresenta-se um exemplo. Supondo que, nesta figura, a posição de início da jogada é a posição $(1,1)$, temos
  $posicoes\text{-}de\text{-}jogada(faz\text{-}jogada(faz\text{-}pos(1,1), faz\text{-}pos(1,7))) =$
  $(((1,2)(1,4)(1,6))((1,3)(1,5)(1,7)))$.

5 Classificação

A nota da primeira parte do projecto será baseada nos seguintes aspectos:

1. Execução correcta (40%).
2. Facilidade de leitura, nomeadamente abstracção procedimental, abstracção de dados, nomes bem escolhidos, paragrafação correcta, qualidade (e não quantidade) dos comentários[3] (25%).
3. Relatório (30%).
4. Estilo de programação (5%).

Os pesos das notas das duas partes na nota final do projecto são os seguintes:

1. Primeira parte - 30%.
2. Segunda parte - 70%.

[3]Todos os comentários do seu programa devem ser feitos utilizando a opção “Comment Out with Semicolons". Programas que utilizem a opção “Comment Out with a Box” serão penalizados com três valores.

6 Condições de realização e prazos

O projecto deve ser realizado em grupos de 2 ou 3 alunos. Os alunos de um grupo podem pertencer a qualquer turno das aulas práticas. Os alunos devem proceder à inscrição do grupo através do sistema Fénix.

A primeira parte do projecto deve ser entregue até às 15:00 horas do dia **22 de Novembro de 2010**, na Reprografia do DEI ou na portaria do Tagus (consoante o campus que corresponda ao grupo do projecto), e deverá constar de um relatório, incluindo uma listagem do código. Um modelo de relatório, que podem/devem adaptar de acordo com as vossas necessidades, será disponibilizado no sistema Fénix. Projectos em atraso serão aceites durante a semana de 22 a 26 de Novembro, sendo penalizados com 0,5 valores por cada dia de atraso. A partir das 15:00 horas do dia 26 de Novembro de 2010 não se aceitam quaisquer projectos, seja qual for o pretexto. O relatório deve ser entregue dentro de uma capa ou encadernado, apresentando visivelmente o número do grupo e número e nome dos seus autores, na capa. Projectos que não sejam entregues nestas condições serão penalizados com três valores.

Para além disto, a submissão do código por via electrónica, *através do sistema Fénix*, é *obrigatória* e deverá ser feita nos mesmos prazos que a entrega do relatório. Se o código e o relatório forem entregues em dias diferentes, o desconto será o correspondente ao dia da última entrega. O código do projecto deve estar contido num único ficheiro. Se durante o desenvolvimento usaram vários ficheiros devem, no final, antes de submeter o código, colocar todo o código num único ficheiro, chamado `FP1011-parte1-grupo`n`.scm`, em que n é o número do grupo. Por exemplo, o ficheiro do grupo nº 5 deverá chamar-se `FP1011-parte1-grupo5.scm`. O ficheiro de código deve conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo.

Durante o desenvolvimento do programa é importante não se esquecer da *Lei de Murphy*:

1. Todos os problemas são mais difíceis do que parecem;
2. Tudo demora mais tempo do que nós pensamos;
3. Se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis.

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

Projectos iguais, ou muito semelhantes, serão considerados cópias. O corpo docente da cadeira será o único juiz do que se considera ou não copiar no projecto.