

Epaminondas

Relatório Final



Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e
Computação

Programação em Lógica

Grupo 13:

Hugo Ari Rodrigues Drumond - 201102900

João Alexandre Gonçalinho Loureiro - 200806067

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

10 de Novembro de 2013

Resumo

O jogo de tabuleiro que iremos desenvolver chama-se Epaminondas. As peças presentes no tabuleiro são todas iguais e movem-se como o Rei no xadrez, exceto quando se agrupam numa linha, coluna ou diagonal. Trata-se do 1º. trabalho de grupo, da disciplina de Programação em Lógica, realizado pelo Grupo 13.

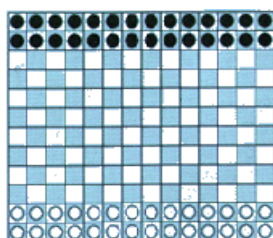
1 Introdução

O fim deste trabalho é criar um jogo de tabuleiro com base na matéria exposta nas aulas teóricas e teórico-práticas. Este trabalho é interessante porque obriga-nos a pensar de um ponto de vista puramente lógico, isto é, cria-se uma base de dados lógica através de cláusulas, sem controlo de fluxo (explícito), e depois procuramos por possíveis soluções. Este paradigma de programação chama-se programação declarativa.

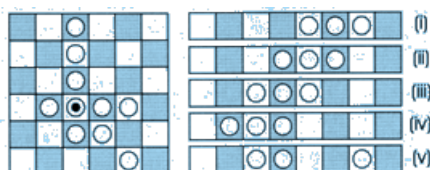
2 O Jogo Epaminondas

Este jogo, inicialmente chamado *Crossings* e jogado num tabuleiro 8x8, foi descrito pela primeira vez no Livro *A Gamut of Games* em 1969. Depois da publicação deste livro, Bob Abbott, reconfigurou o tabuleiro de 8x8 para 14x12, de modo a tornar o jogo nas diagonais mais importante, e renomeou o nome para Epaminondas como homenagem ao general TheBan que inventou a phalanx, uma formação de guerra usada em 371 a.c para derrotar o exército espartano.[1, 2]

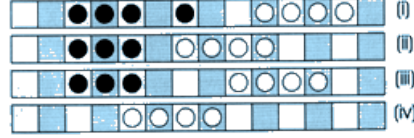
Neste jogo, são posicionadas 28 peças iguais em cada lado maior do tabuleiro, com cores diferentes. São as brancas a iniciar o jogo, e depois joga-se alternadamente.[2]



Os movimentos de cada peça são iguais ao do Rei do xadrez, pode andar uma casa em qualquer direção. Podem ser criados inúmeros grupos de peças, phalanxes, estas só se podem deslocar um número igual ou menor ao número de peças do grupo, na direção da linha formada pelo grupo. Uma peça pode fazer parte de várias phalanxes. Uma phalanx pode ser dividida, mas só pode andar um número de casas igual ou menor ao número de elementos da phalanx que se irá mover.[2]



Não é permitido passar um jogada, em cada jogada é obrigatório mover uma phalanx ou uma peça. Todas as peças têm de estar contidas no tabuleiro. Não podem haver peças sobrepostas ou na mesma posição, exceto quando a cabeça de uma phalanx captura outra peça ou phalanx.



Para capturar é necessário atacar com um grupo com mais elementos que o adversário na linha de ataque. A peça/phalanx capturada é removida do tabuleiro para sempre. Um jogador ganha quando, na sua vez de jogar, tiver mais peças que o adversário na linha mais afastada de si.[2]

3 Arquitetura do Sistema

O jogo é iniciado com um predicado chamado start: mostra o tabuleiro inicial; lê as coordenadas através de READ\1 e chama o predicado recursivo play(E,XS,YS,XD,YD). Este predicado faz o seguinte: na parte inicial testa os valores XS,YS,XD,YD de modo a ver se são válidos; caso sejam chama atualizarEstado(E,XS,YS,XD,YD,N); caso este objetivo seja completado, é mostrado o novo tabuleiro; são lidos os novos valores de XS,YS,XD,YD e é chamado play com os novos valores.

A função de atualizarEstado(E,XS,YS,XD,YD,N) é "corrigir" os valores de XS,YS,XD,YD visto que são mostrados de forma diferente; verificar se a jogada é válida através de canimovethere(E,XS1,YS1,XD1,YD1); se sim, mudarestado(E,XS1,YS1,XD1,YD1,N) e depois mudar a vez.

Na verificação das regras canimovethere(E,XS,YS,XD,YD): chama regras(E,XS,YS,XD,YD) e caso esta falhe é chamado badMoveFriend(E) cujo objetivo é pedir novamente que se jogue. A cláusula regras(E,XS,YS,XD,YD) contém 3 regras que verificam: se é a minha vez; se existe sobreposicao; se se pode mover uma phalanx(uma peça é um phalanx com um só elemento) ou se se pode capturar uma peça; e se estamos em condição de acabar o jogo. Se estivermos, o jogo acaba e é indicado quem ganhou.

Foram construídas várias regras úteis: getSizeTilColour('tipodelinha'); headPhalanx('tipodelinha'); checkColour(); changechar(,r); findEnemyHead('tipodelinha'); movePhalanxUP(,); movePhalanxLEFT(,).Cujos objetivos são bastante óbvios.

4 Lógica do Jogo

4.1 Representação do Estado do Jogo

Iremos criar um estado inicial, uma lista de listas da forma:
[[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]]

Através da regra de interface `play` iremos chamar as `size` e `atualizarestado` que testa as regras do jogo e atualiza o estado, caso tal respeite as regras do jogo. A regra `atualizarestado` chama pelas cabeças `canimovethere`, `mudarestado` e `turnchange`. Se a regra `canimovethere` não corresponder isso implica o falhanço do "if" em prolog e portanto o estado não é alterado nem a vez, sendo pedida outra vez um movimento. Se o objetivo `canimovethere` for bem sucedido, então podemos proceder à alteração do estado através da regra `mudarestado`. Restando mudar a vez do jogador, através de por exemplo `turnchange(C)` (através de `retract` e `asserta`). Sendo este ciclo repetido novamente, até chegarmos a uma situação de fim de jogo.

```

109 start :-
110     estadoInicial(E),
111     show(E),
112     read(XS), read(YS), read(XD), read(YD), %le desta forma XS.YS.XD.YD.
113     play(E, XS, YS, XD, YD).
114
115 play( E, XS, YS, XD, YD) :-
116     xDIFy(XS,YS,XD,YD),
117     size(XS,YS),
118     size(XD,YD),
119     atualizarEstado(E,XS,YS,XD,YD,N),
120     show(N),
121     read(XS), read(YS), read(XD), read(YD), % le desta forma XS.YS.XD.YD.
122     play( N, XS, YS, XD, YD).
123 %If sizes are not allowed retry.
124 play(E,_,_,_,_) :- badMoveFriend(E).
125
126 xDIFy(XS,YS,XD,YD) :-
127     XS \= XD; YS \= YD.
128
129 atualizarEstado( E, XS, YS, XD, YD, N) :-
130     XS1 is XS - 1,
131     YS1 is 12 - YS, % Porque na verdade o que aparece no tabuleiro tem o índice trocado.
132     canimovethere(E,XS,YS,XD,YD),
133     mudarestado(E,XS,YS,XD,YD,N),
134     turnchange(C).
135 atualizarEstado(E,_,_,_,_) :- badMoveFriend(E).

```

O estado vai sendo atualizado consoante as jogadas de cada jogador, o 0 significa que não há nenhuma peça nessa posição. O 1 representa uma peça branca e o 2 uma preta. Quando se imprime o 1 passa a W e o 2 a B.

Um estado com posições intermédias.

12		B	B	B	B	B	B	B	B	B	B	B	B	B	B
11		B	B	B	B	B	B	B	B	B	B	B	B	B	B
10	B														
9	B														
8															
7															
6															
5															
4															
3	W														
2		W	W	W	W	W	W	W	W	W	W	W	W	W	W
1	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	

$$\begin{aligned} & [[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],[0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2],[2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], \\ & [2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], \\ & [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], \\ & [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1] \end{aligned}$$

Um estado com posições finais, qualquer configuração em que existam mais peças do adversário na minha linha mais recuada, na vez dele jogar. E vice-versa.

4.2 Visualização do Tabuleiro

De modo a imprimir o estado atual do tabuleiro, tem-se de construir um predicado recursivo que escreva o conteúdo de cada linha na consola e depois dê um break line. Tal pode ser feito através das seguintes cláusulas:

```
19 sepforprint([],_,_):-!.
20 sepforprint([H|T],_totalVerticalLines,_totalHorizontalLines,_currentHorizontal):-
21   write(' '),
22   numbers(vert,_currentHorizontal),
23   _currentHorizontal1 is _currentHorizontal - 1,
24   println(H),
25   write(' '),
26   nl,
27   write(' '),
28   write(' '),
29   write(' '),
30   lineseparator(_totalVerticalLines),
31   nl,
32   sepforprint(T,_totalVerticalLines,_totalHorizontalLines,_currentHorizontal1).
```

```
61 println([]).
62 println([H|T]):-
63   write(' '),
64   printchar(H),
65   println(T).
```

Resultando:

12	B	B	B	B	B	B	B	B	B	B	B	B	B	B
11	B	B	B	B	B	B	B	B	B	B	B	B	B	B
10														
9														
8														
7														
6														
5														
4														
3														
2	W	W	W	W	W	W	W	W	W	W	W	W	W	W
1	W	W	W	W	W	W	W	W	W	W	W	W	W	W
	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4.3 Validação das Jogadas

A regra principal do nosso programa é `play(E,XS,YS,XD,YD)`, no início desta são verificados casos básicos de regras: `xDIFy(XS,YS,XD,YD)` verifica se as coordenadas são exatamente iguais se forem falsa, `size(XS,YS)` vê se XS e YS estão dentro do tabuleiro e o mesmo para XD e YD. O resto das regras são verificadas em `canimovethere(E,XS,YS,XD,YD)`, cuja cabeça está em `atualizarEstado(E,XS,YS,XD,YD,N)` que por sua vez está em `play(E,XS,YS,XD,YD)`.

Em `canimovethere(E,XS,YS,XD,YD)` é chamado as regras(`E,XS,YS,XD,YD`). Lá encontram-se as regras restantes: é verificado se a cor na posição XS,YS tem o mesmo valor que a cor de quem deve jogar; temos um outra regra que verifica se XS,YS e XD,YD têm a mesma cor se tiverem falsa; de seguida temos a regra mais complicada que verifica se se pode comer ou mover uma phalanx(engloba todos os tipos de linhas) e finalmente a regra de fim de jogo.

```
115 play( E, XS, YS, XD, YD) :-
116   xDIFy(XS,YS,XD,YD),
117   size(XS,YS),
118   size(XD,YD),
119   atualizarEstado(E,YS,YS,XD,YD,N)
120
121
122
123
124
125
126
127 regras([H|T], XS, YS, XD, YD) :-
128   turn(C),
129   checkcolour(E,XS,YS,C),
130   regra('sobreposicao',E,XS,YS,XD,YD),
131   regra('moverPhalanx',E,XS,YS,XD,YD).
```

4.4 Lista de Jogadas Válidas

Não tivemos tempo para completar esta funcionalidade, mas estava previsto criar um predicado recursivo que percorre-se todos os XS,YS,XD,YD e caso as regras fossem válidas inseríamos numa lista uma lista com os movimentos válidos, por exemplo, $[[1,2,3,4],[1,4,3,7]]$. Este ciclo era repetido até extinguir as possibilidades de XS,YS,XD,YD.

4.5 Execução de Jogadas

4.6 Avaliação do Tabuleiro

4.7 Final do Jogo

4.8 Jogada do Computador

4.9 Comunicação com Visualizador 3D

5 Interface com o Utilizador

6 Conclusões e Perspectivas de Desenvolvimento

O Epaminondas é um jogo cheio de estratégia e regras, tivemos de fazer várias regras de remoção, testes, etc dependendo da natureza da linha de movimentação das phalanx e portanto esta parte foi, na nossa opinião, a mais trabalhosa. Infelizmente, só conseguimos acabar a parte jogador vs jogador, isto foi devido a termos outros projetos em mão. A movimentação das peças que pode ser feita em oito direções e usando só uma peça ou um conjunto de peças, no caso dos phalanxs, sempre tendo em atenção os limites do tabuleiro e as outras peças, quer sejam do próprio jogador o que impossibilita a jogada ou do jogador adversário cujas peças têm de ser removidas do tabuleiro, no caso de ter menos peças no phalanx, ou então serão as peças do próprio a serem removidas, irá ser a parte mais desafiante a implementar.

Gostaríamos de ter acaba o trabalho em tempo útil mas não nos foi possível. Mas na globalidade foi uma boa experiência aprender a pensar em PROLOG.

Bibliografia

- [1] Bob Abbott. Epaminondas. <http://www.logicmazes.com/games/epam.html>,
Revised: December 11, 2010. The website of the game creator.
- [2] Kerry Handscomb. Abstract games issue 3 autumn 2000.
<http://www.logicmazes.com/games/epam/index.html>, 3 Autumn 2000.
Epaminondasm... a game of classical elegance.