LEAR

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

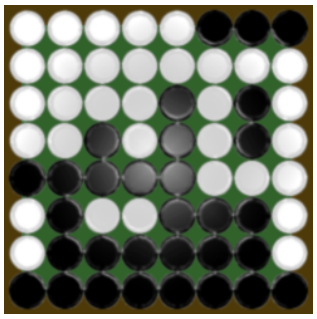
Programação em Lógica

Grupo Lear_1:
Antero Gandra - 201607926
Fernando Fernandes - 201505821

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

15 de Outubro de 2017

1 O Jogo Lear



Categoria: Estratégia

Nº de Jogadores: 2

Tamanho do tabuleiro:

(recomendado) entre 7 por 7 e 10 por 10

Duração: entre 15 a 60 minutos

Baseado no jogo Go, inventado na China antiga há mais de 2500 anos atrás (considerado um dos jogos mais antigos jogado nos dias de hoje), Lear é uma variação no que toca ao tamanho do tabuleiro e simplicidade das regras. O jogo consiste no controlo de território ocupado pelo adversário, sendo que só existe um tipo de peças (pedras redondas) distinguidas pela cor (pretas ou brancas).

Início

Antes do jogo começar é necessário decidir quem joga primeiro, sendo que o primeiro jogador decide que número será usado como *komi*, e o adversário escolhe a cor das peças.
O jogador que decidir ficar com as peças pretas é o que inicia o jogo.

komi

O *komi* consiste na quantidade de pontos adicionados no fim do jogo ao adversário que não pode fazer uma última jogada (devido ao preenchimento total do tabuleiro).
No caso de ser um tabuleiro com número ímpar de vértices, o *komi* deve ser par e ir para o adversário com peças pretas (uma vez que como começa no tabuleiro está automaticamente impossibilitado de fazer a última jogada).Por consequente, no caso de ter número par de vértices, o *komi* deverá ser ímpare para a equipa com peças brancas.

Regras

O jogador no seu turno tem de colocar obrigatoriamente uma peça num vértice desocupado. Se um jogador colocar uma peça sua numa linha sem interrupções, se essa linha já possuir uma peça sua e não tiver mais interrupções na linha feitas por peças suas ou do adversário, o jogador terá de virar as peças do adversário.

Exemplos: **X+OO | +XOO | +OOOX | +XXOOO**
Nos primeiros 3 diagramas, um 'X' jogado no lugar do '+' vira as pedras 'O'. No entanto, no quarto diagrama, as peças 'O' não são viradas.

Fim do jogo

O jogo declara-se finalizado só e apenas quando o tabuleiro estiver totalmente preenchido sendo que o vencedor é o jogador com maior pontuação (número de peças no tabuleiro + *komi* se possuído).

2 Representação do Estado do Jogo

De modo a apresentar o tabuleiro em Prolog, por predefinição, usaremos a proporção 7x7, para não variar o tamanho do tabuleiro.

Para representar o tabuleiro em Prolog será utilizada uma lista de listas, sendo que os vértices serão representados utilizando pontos (.), as peças pretas X, e as brancas O.

No entanto, em Lear, a colocação de peças faz-se nos vértices e não nos quadrados o que dá um total de 8X8, possuindo assim 64 vértices (64 posições onde colocar as peças do jogo).Para efeitos de programação em PROLOG, esta variante não condiciona o desenho e lógica normal de um tabuleiro, já que podemos pensar nos vértices como células e colocar o tabuleiro na mesma como lista de listas (já quando for para desenhar o tabuleiro numa cena gráfica (LAIG) , iremos ter em atenção este pormenor.)

Inicialização (sem peças):

```
tabuleiro([]).
tabuleiro([X|S]):-
tabuleiro(S).
```

Exemplos das restantes posições:

```
tabuleiro
([
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.']
]).
```

estado inicial:

```
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
```

```
printBoard
([
['.', '.', '.', '.', '.','.' ,'.'] ,
['.', '.', '.', '.', 'X' ,'X' ,'.'],
['.', '.', '.', 'O', '.' ,'.' ,'O'],
['.', '.', '.', 'X', '.' ,'O' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.'],
['.', '.', '.', '.', '.','.' ,'.']
]).
```

estado intermédio:

```
. . . . . . .
. . . . X X .
. . . O . . O
. . . X . O .
. . . . . . .
. . . . . . .
. . . . . . .
```

```
printBoard
([
[ 'X', 'X', 'O', 'O', 'O' ,'X' ,'X'] ,
['X', 'X', 'X', 'O', 'O' ,'X' ,'X'],
['X', 'O', 'O', 'O', 'X' ,'X' ,'O'],
['O', 'O', 'O', 'X', 'X' ,'X' ,'X'],
['O', 'O', 'O', 'X', 'X' ,'O' ,'X'],
['X', 'O', 'X', 'O', 'O' ,'O' ,'O'],
['O', 'O', 'O', 'X', 'X' ,'O' ,'X']
]).
```

estado final:

```
X X O O O X X
X X X O O X X
X O O O X X O
O O O X X X X
O O O X X O X
X O X O O O O
O O O X X O X
```

3 Visualização do Tabuleiro

Instruções PROLOG para criação do tabuleiro:

printBoard:
A regra printBoard retorna **true** se Tab for um tabuleiro, e se printTabuleiro o conseguir imprimir.

```
printBoard(Tab):-
tabuleiro(Tab), printTabuleiro(Tab).
```

---

printTabuleiro:
printTabuleiro tenta imprimir uma linha(printLinha), ou seja uma lista(das listas do tabuleiro), e depois escreve um newline.

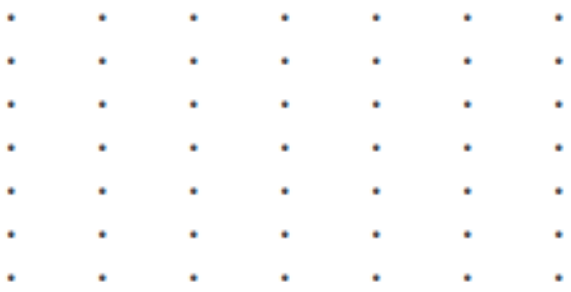
```
printTabuleiro([H|T]):-
printLinha(H), write(' \n'), printTabuleiro(T).
printTabuleiro([]).
```

---

printLinha:
A regra printLinha escreve um elemento de cada vez, separado por um espaço vazio.

```
printLinha([H|T]):-
write(H), write(' '),printLinha(T).printLinha([]).
```

Tabuleiro:



4 Movimentos

Durante uma partida de Lear é apenas necessário colocar peças em células vazias (posições válidas).
A parte mais complicada é programar as consequências que esse posicionamento traduz no tabuleiro.

Instrução para colocar uma peça (setPeca):
Retorna true se for possível colocar a peça nas coordenadas(NLinha,NColuna) e colocar o tabuleiro final em TabOut.
TabOut tem de ser um tabuleiro.

```
setPeca(NLinha, NColuna, Peca,TabIn, TabOut):-
setPecaLinha(NLinha, NColuna,TabIn, Peca, TabOut), tabuleiro(TabOut),
printTabuleiro(TabOut).
```

Instrução para a linha de uma peça (setPecaLinha):
Decrementando NLinha até achar a linha(lista certa), e depois chama setPecaColuna, retornando true se conseguir.

```
setPecaLinha(NLinha, NColuna, [H|T], Peca, [H|R]):-
Previous is NLinha-1, setPecaLinha(Previous, NColuna, T, Peca, R).

setPecaLinha(1, NColuna, [H|T], Peca, [I|T]) :-
setPecaColuna(NColuna,H,Peca,I).
```

Instrução para a coluna de uma peça (setPecaColuna):
SetPecaColuna, na lista fornecida por setPecaLinha, vai decrementando NColuna até achar a coluna certa, e quando achar, coloca a peça nessa coluna.

```
setPecaColuna(NColuna,[P|Resto],Peca,
[P|Mais]) :-
Previous is NColuna-1, setPecaColuna(Previous, Resto,Peca,Mais).

setPecaColuna(1,[_| Resto],Peca,[Peca|Resto]).
```
