LEAR

Relatório Final

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

12 de Novembro de 2017

Resumo

Com este trabalho pretendemos demonstrar como é possível implementar o jogo Lear na linguagem Prolog.
De modo a abordar corretamente e organizadamente o jogo, separamo-lo em vários módulos diferentes para implementar: a lógica do jogo, os ciclos jogo e, interface (cada um com o seu respetivo ficheiro) .
A nossa maior dificuldade foi em saber como implementar o sistema de verificação das peças, sendo que o que nos deu mais problemas foi conseguir saber quando é que elas podiam ser viradas.
Acabamos por optar por uma verificação à frente e outra atrás, no caso das linhas e uma acima e outra abaixo para as colunas, do sítio onde a peça foi jogada.
Outro grande desafio foi implementar uma inteligência artificial não totalmente aleatória, embora rudimentar, para colocar um jogador a jogar contra uma máquina.
Perante este problema, a solução passou por achar peças vulneráveis a serem viradas, e colocar a máquina a jogar a peça numa posição que virasse essas peças.

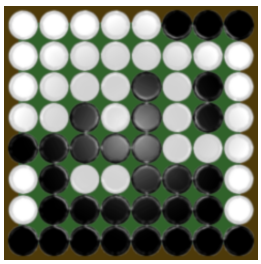
Conteúdo

1 Introdução

Na realização deste projeto tivemos como finalidade, ao aplicar todo o conhecimento que possuímos na área de paradigmas lógicos,conseguir recriar um jogo cuja origem é muito conhecida na área dos jogos de tabuleiro (**o jogo Go**), devido á sua dificuldade e necessidade de várias estratégias (por causa do elevado número de jogadas possíveis).
O que nos deu maior motivação e que nos despertou mais interesse foi poder aplicar conhecimentos de prolog nas consequências da movimentação de peças em Lear.

Neste relatório álem de abordarmos o funcionamento do jogo, delineamos o seu processo de implementação na linguagem Prolog, começando por descrever como o tabuleiro e as jogadas são processadas, seguido pelos métodos e estratégias aplicadas pelo computador quando este participa no jogo, e concluindo com uma breve demonstração da interface aplicada no jogo.

2 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 ímpar 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.

Exemplo:

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).

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

De modo a apresentar o tabuleiro em Prolog, por predefinição, usaremos a proporção 8x8, para não variar o tamanho do tabuleiro.
Para representar o tabuleiro será utilizada uma lista de listas, sendo que os vértices serão representados utilizando a letra v (de vértice), as peças pretas x, e as brancas o.
Dado que, em Lear, a colocação de peças faz-se nos vértices e não nos quadrados, para um tabuleiro 7x7 ,isso dá um total de 8X8 vértices, 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.

Representação do tabuleiro no início do jogo

```
tabuleiroVazio(Tab):- Tab =
[['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'] ,
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'],
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'],
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'],
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'],
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'],
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v'],
['v', 'v', 'v', 'v', 'v' ,'v' ,'v','v']].
```

v v v v v v v v
v v v v v v v v
v v v v v v v v
v v v v v v v v
v v v v v v v v
v v v v v v v v
v v v v v v v v
v v v v v v v v

Representação possível do tabuleiro a meio do jogo

```
[['o', 'v', 'o', 'o', 'o' ,'o' ,'o','o'] ,
['v', 'x', 'o', 'o', 'x' ,'v' ,'x','x'],
['v', 'x', 'x', 'o', 'o' ,'v' ,'x','x'],
['o', 'x', 'o', 'o', 'x' ,'x' ,'o','x'],
['x', 'x', 'v', 'o', 'x' ,'o' ,'v','v'],
['x', 'x', 'v', 'v', 'x' ,'o' ,'o','x'],
['o', 'v', 'x', 'o', 'x' ,'o' ,'v','x'],
['v', 'x', 'o', 'o', 'x' ,'o' ,'x','v']].
```

o v o o o o o o
v x o o x v x x
v x x o o v x x
o x o o x x o x
x x v o x o v v
x x v v x o o x
o v x o x o v x
v x o o x o x v

Representação possível do tabuleiro no fim do jogo

```
[['o', 'o', 'o', 'o', 'o' ,'o' ,'o','o'] ,
['x', 'x', 'o', 'o', 'x' ,'o' ,'x','o'],
['o', 'x', 'x', 'o', 'o' ,'o' ,'x','o'],
['o', 'x', 'o', 'o', 'x' ,'x' ,'o','o'],
['x', 'x', 'o', 'o', 'x' ,'o' ,'x','o'],
['x', 'x', 'x', 'o', 'x' ,'o' ,'o','o'],
['o', 'x', 'x', 'o', 'x' ,'o' ,'x','o'],
['x', 'x', 'o', 'o', 'x' ,'o' ,'x','o']].
```

o o o o o o o o
x x o o x o x o
o x x o o o x o
o x o o x x o o
x x o o x o x o
x x x o x o o o
o x x o x o x o
x x o o x o x o

3.2 Visualização do Tabuleiro

De modo a poder representar o tabuleiro usamos as regras printBoard, printTabuleiro, e printLinha.

---

printBoard:
A regra printBoard é provada se printTabuleiro conseguir imprimir o tabuleiro.

printBoard(Tab):- printTabuleiro(Tab).

---

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

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

---

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

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

3.3 Lista de Jogadas Válidas

Para se obter uma lista de jogadas possíveis, criaram-se as regras obtemJogadasPossiveis e adicionaJogadas.

---

obtemJogadasPossiveis:
A regra obtemJogadasPossiveis adiciona jogadas ao longo de cada uma das linhas do tabuleiro.

obtemJogadasPossiveis([ ], _ , [ ]).
obtemJogadasPossiveis([Linha Resto], NLinha,Jogadas):-
adicionaJogadas(Linha, NLinha, 1,JogadasAux),
Nl is NLinha + 1 ,
obtemJogadasPossiveis(Resto,Nl,JogadasAux2),
append(JogadasAux,JogadasAux2,Jogadas).

---

adicionaJogadas
A regra adicionaJogadas verifica se o elemento da lista é um espaço vazio 'v', e, em caso afirmativo, adiciona à lista de saída uma lista com duas posições, a linha e a coluna da célula que está vazia.

adicionaJogadas([], _ ,_ ,[]).
adicionaJogadas(['v' |$Resto$], $NLinha$, $NColuna$,
[NLinha,NColuna
$RestoJogadas$]) : $-NcisNColuna + 1$,
$adicionaJogadas(Resto, NLinha, Nc, RestoJogadas)$.
$adicionaJogadas([\_Resto], NLinha, NColuna, Jogadas) :-$
$NcisNColuna + 1$,
$adicionaJogadas(Resto, NLinha, Nc, Jogadas)$.

3.4 Execução de Jogadas

Executar uma jogada em Lear, é simplesmente jogar uma peça numa célula vazia. Na nossa implementação, existe a regra **setPeca**, que faz isso, recebendo um número de linha e de coluna e quando achar a posição certa com as regras **setPecaLinha** e **setPecaColuna**, faz a unificação.
setPecaLinha, procura a linha correta, e setPecaColuna, procura a coluna correta nessa linha.

---

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

---

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

$setPecaColuna(NColuna, [P|Resto], Peca, [P|Mais]) :-$
$PreviousisNColuna - 1$,
$setPecaColuna(Previous, Resto, Peca, Mais)$.
$setPecaColuna(1, [\ |Resto], Peca, [Peca|Resto])$.

3.5 Avaliação do Tabuleiro

Com o predicado verificaJogada, sempre que existe uma jogada nova, é feita a avaliação das consequências dessa jogada, ou seja, se para além de colocar a peça nova no sítio correspondente é necessário virar peças do adversário.
Para isso, avalia-se a linha onde foi jogada a peça com **verificaJogadaLinha**, e a coluna com **verificaJogadaColuna**.
Caso seja necessário virar as peças ao longo de uma linha, usa-se a regra **viraPecasLinha**, e o mesmo raciocínio para as colunas.

---

verificaJogada(Tabuleiro, NLinha, NColuna, Peca, TabOut):-
avancaParaLinha(Tabuleiro, NLinha, L),
verificaJogadaLinha(L, NColuna, Peca),
viraPecasLinha(Tabuleiro, NLinha, Peca, TabAux),
verificaJogadaColuna(TabAux, NLinha, NColuna,Peca),
viraPecasColuna(TabAux,Peca, NColuna, TabOut).

---

verificaJogada(Tabuleiro, NLinha, NColuna, Peca, TabOut):-
verificaJogadaColuna(Tabuleiro, NLinha, NColuna,Peca),
viraPecasColuna(Tabuleiro,Peca, NColuna, TabOut).

---

verificaJogada(Tabuleiro, NLinha, NColuna, Peca, TabOut):-
avancaParaLinha(Tabuleiro, NLinha, L),
verificaJogadaLinha(L, NColuna, Peca),
viraPecasLinha(Tabuleiro, NLinha, Peca, TabOut).
verificaJogada(Tabuleiro, , , , Tabuleiro).

3.6 Final do Jogo

O fim do jogo verifica-se apenas quando o tabuleiro não tem mais posições vazias.
Após a verificação dessa condição, é enviada uma mensagem para o ecrã de quem é o vencedor, seguida das pontuações de cada jogador **após a soma do komi**.
A regra que verifica o fim do jogo é a **verificaFim**, que vê para cada linha se existe algum 'v'. Caso seja fim de jogo, vai verificar o vencedor com a regra **verificaVencedor,** contando as peças de cada um, e adiciona o komi.

---

komi(X):- X is 1.

verificaFim(Tabuleiro):-verificaFimDoJogo(Tabuleiro),
pecaJogadorUm(PecaJogadorUm),
pecaJogadorDois(PecaJogadorDois),
verificaVencedor(Tabuleiro, PecaJogadorUm, PecaJogadorDois).

verificaFimDoJogo([]).
verificaFimDoJogo([Linha $|T$]) : −
$verificaFimDoJogoLinha(Linha), verificaFimDoJogo(T)$.
$verificaFimDoJogoLinha([])$.

3.7 Jogada do Computador

Para podermos começar a aplicar os conceitos básicos de Inteligencia Aetificial, optamos desenvolver 2 dificuldades distintas: modo fácil e modo difícil. Estes dois modos estão disponíveis no menu de escolha, presente no início de uma partida contra computador.

Modo Fácil

Para a dificuldade fácil, escolheu-se o computador gerar uma gerada aleatória a partir de uma lista de jogadas possíveis, com o predicado **jogaPC**, assim como simular o tempo de "pensar a jogada"com a regra sleep, que espera um determinado número de segundos.

```
jogaPC(Peca,Tabuleiro,TabOut):-
sleep(1),
obtemJogadasPossiveis(Tabuleiro,1,Jogadas),
numeroJogadasPossiveis(Jogadas, Numero),
NumeroJogadas is Numero + 1,
random(1,NumeroJogadas,Jogada),
selecionaJogada(Jogadas, Jogada, [NLinha | NCol]),
NColuna is NCol,
jogadaPC(NLinha,NColuna),
setPeca(NLinha, NColuna, Peca,Tabuleiro, TabOut2),
verificaJogada(TabOut2, NLinha, NColuna, Peca, TabOut),!.
```

Modo Difícil

Para a dificuldade difícil, a regra **jogaPCIA**, verifica se alguma das jogadas consegue virar peças do adversário.
Se conseguir joga na primeira que encontrar, caso contrário, joga numa posição aleatória

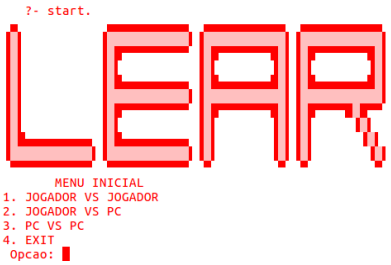
```
jogaPCIA(Peca,Tabuleiro,TabOut):-
sleep(1),
obtemJogadasPossiveis(Tabuleiro,1,Jogadas),
tentaViraPecas(Tabuleiro, Jogadas, Peca, NLinha,NColuna),
jogadaPC(NLinha,NColuna),
setPeca(NLinha, NColuna, Peca,Tabuleiro, TabOut2),
verificaJogada(TabOut2, NLinha, NColuna, Peca, TabOut),!.
```

---

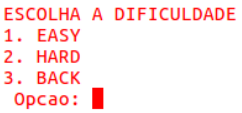
4 Interface com o Utilizador

Ínicio do Jogo

No ínicio do jogo,após uma mensagem de boas vindas, é perguntado ao utilizador que modo de jogo é que pretende,ou seja,computador vs ser humano, computador vs computador ou ser humano vs ser humano.

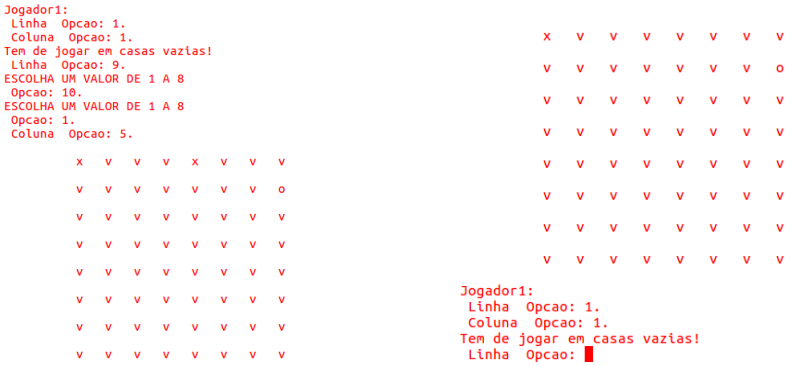


No caso de ser selecionado computador vs ser humano, pergunta também que dificuldade pretende que o computador tenha.



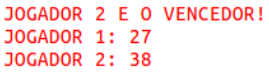
Tabuleiro

Depois de representar o tabuleiro no ecrã, é perguntado ao utilizador (utilizadores no caso de serem dois seres humanos)que jogada pretendem fazer e para que posição específica.
No caso da posição onde se quiser jogar já esteja ocupada, é enviada uma mensagem a avisar ao utilizador que náo pode fazer essa jogada.
Esta questão vai ser feita de jogada a jogada, até que seja impossível fazer mais jogadas, ou seja, até o tabuleiro encher.



Fim

Quando se verificar que o tabuleiro já se encontra totalmente ocupado com peças, é enviada a mensagem de "Game Over", seguida pelas pontuações de cada jogador.
Para acabar é questionado ao utilizador se este pretende jogar mais uma vez ou encerrar o programa.



5 Conclusões

Com este projeto conseguimos implementar de uma maneira interativa e interessante os conceitos de prolog aprendidos em aula.
Aplicamos várias dos diversos tipos de abordagens a problemas abstratos na área da Lógica e empregamos estruturas de dados complexas.
Adquirimos também conhecimentos na área da lógica existente nos jogos de tabuleiro e capacidades para os representar e implementar em prolog.

Alterações face o relatório intercalar

Algo que teve que ser alterado foi o símbolo de peça vazia, que antes era um ponto (`.`), que representa formalmente a definição de vértice, uma vez que para representar carateres para além de letras são necessárias plicas, e isso ia tornar o nosso código ilegivel e com aspeto desorganizado.
Para solucionar esse problema alteramos o símbolo ponto para a letra v (de ***vértice***) de maneira a continuar a representar corretamente o conceito de vértice.

Melhorias por implementar

De modo a melhorar o trabalho desenvolvido, seria necessário mais tempo disponível quer para nos reunirmos em grupo, quer para podermos implementar mais detalhes no jogo.

Bibliografia

Livros:

The Art of Prolog (Second Edition)
Leon S. Sterling and Ehud Y. Shapiro
ISBN: 9780262193382

Clause and Effect
William Clocksin,
ISBN:9783540629719

Websites:

Documentação PROLOG:

https://www.metalevel.at/prolog
http://www.swi-prolog.org

Regras do jogo:

https://boardgamegeek.com/boardgame/209777/lear
https://senseis.xmp.net/?Komi

O Código Prolog implementado encontra-se devidamente comentado e anexado na pasta source.