Mestrado Integrado em Engenharia Informática e Computação

PLOG

Lear



Trabalho Prático 1 - Relatório Final

**Grupo Lear\_4**

Leonor Mendes de Freitas 201207603

Marisa Oliveira 201308594

**Resumo**

No âmbito da unidade curricular de programação em lógica enquadrada no 3ºano do plano de estudos do mestrado integrado em engenharia informática e de computação foi proposto aos alunos o desenvolvimento de um jogo de tabuleiro em Prolog.

Prolog é uma linguagem com um paradigma diferente das linguagens mais comuns, sendo que compreender o funcionamento desta linguagem foi desafiante. O problema foi abordado usando listas para a representação do tabuleiro. Posteriormente foram criados predicados para permitir a realização de um jogo de uma forma dinâmica.

O resultado //TODO

// Resumo sucinto do trabalho com 150 a 250 palavras (problema abordado, objetivo, como foi o problema resolvido/abordado, principais resultados e conclusões).

**ÍNDICE**

Introdução 3

O JOGO LEAR 4

LÓGICA DE JOGO 4

REPRESENTAÇÃO DO ESTADO DE JOGO 4

VISUALIZAÇÃO DO TABULEIRO 6

Lista de jogadas válidas 8

Execução de jogadas 8

Avaliação do tabuleiro 8

Final do jogo 8

Jogada do Computador 8

Interface com o utilizador 8

Conclusões 8

Bibliografia 8

Anexos 8

# Introdução

O objetivo do desenvolvimento deste trabalho foi implementar, em linguagem Prolog, um jogo de tabuleiro para dois jogadores com diferentes jogadas e com condições de terminação como derrota, vitória ou empate. O jogo permite três modos de utilização: humano versus humano, humano versus computador e computador versus computador – com níveis de jogo para o computador.

A escolha do grupo recaiu no jogo Lear porque para além de evidenciar ser um jogo interessante, este pareceu também um trabalho bastante apropriado para desenvolver em Prolog.

O relatório está subdividido em duas partes que estão subdividas em pequenas secções. A primeira parte fará uma pequena introdução ao jogo Lear, começando por abordar a sua história, constituição, regras e movimentos possíveis.

A segunda parte incidirá na implementação do jogo Lear em linguagem Prolog, começando por demonstrar como foi representado o estado do jogo, a visualização do tabuleiro, as jogadas possíveis em cada modo de jogo e como será finalizado o jogo.

# O JOGO LEAR

O Lear é um jogo de território, num tabuleiro quadrado com dimensões entre 7x7 e 10x10, para dois jogadores. Os jogadores podem jogar com peças pretas (Black) ou com peças brancas (White). A dimensão escolhida, neste caso, foi 8x8.

O tabuleiro está inicialmente vazio e os jogadores colocam à vez uma peça num ponto no tabuleiro, sendo que Black começa primeiro.

Na sua vez, o jogador deve colocar uma peça da sua cor num ponto vazio. Se o ponto escolhido faz com que a peça passe a fazer parte de uma linha ortogonal contínua de peças, composta por duas peças amigáveis e uma linha contínua de peças inimigas, estas peças inimigas são substituídas por peças da cor oposta.

As duas peças amigáveis podem estar ou as duas juntas numa das pontas da linha ou uma em cada ponta.

x+oo ... +xoo ... +ooox ... +xxooo

Nos primeiros três diagramas acima, jogar um x no local + substitui as peças o por x. No quarto diagrama, tal já não se verifica.

O jogo termina quando o tabuleiro está completamente preenchido. O vencedor é o jogador com a pontuação mais alta. A pontuação de um jogador é igual ao número de peças da sua cor no tabuleiro mais, quando apropriado, um komi.

O komi são pontos que são adicionados à pontuação do jogador que não faz a última jogada. Para evitar empates, o komi deve ser par se a dimensão do tabuleiro for ímpar e vice-versa. Nos tabuleiros ímpares o komi deve ser atribuído ao White e nos tabuleiros pares o komi deve ser atribuído ao Black.

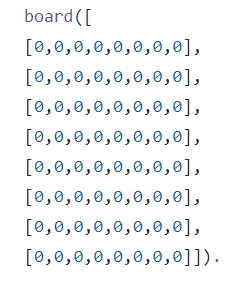
Antes do jogo começar, o primeiro jogador define qual o valor do komi e o segundo jogador escolhe qual a cor com que quer jogar, sendo que o Black é o primeiro a colocar uma peça no tabuleiro.

# LÓGICA DE JOGO

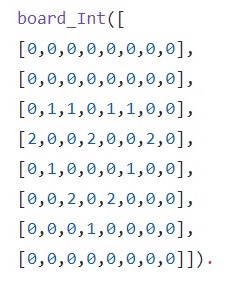
## REPRESENTAÇÃO DO ESTADO DE JOGO

A representação do Lear em Prolog consiste numa lista de listas, sendo que cada uma destas representa uma linha do tabuleiro. Cada linha é composta por símbolos que correspondem a cada ponto do tabuleiro, sendo que o valor do mesmo determina o estado daquela célula (vazia, peça preta ou peça branca). A primeira lista (da lista de listas) corresponde à primeira linha e o primeiro elemento de cada lista corresponde ao ponto mais à esquerda dessa linha.

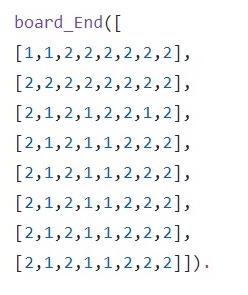
**REPRESENTAÇÃO INICIAL**



**REPRESENTAÇÃO INTERMÉDIA**



**REPRESENTAÇÃO FINAL**



## VISUALIZAÇÃO DO TABULEIRO

A visualização do tabuleiro irá ser feita através de caracteres ASCII impressos na consola, de forma a que cada caracter corresponda a uma célula vazia ou a uma célula ou com uma peça preta ou com uma peça branca. A visualização inicial representa um tabuleiro vazio.

Nas imagens abaixo estão exemplos da visualização do tabuleiro em vários estados, bem como alguns exemplos de código.

**VIZUALIZAÇÃO DO TABULEIRO**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISPLAY BOARD

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

% imprime as coordenadas das

colunas e respetivas barras

view(Board):-

nl,nl, write(' 1 2 3 4 5 6 7 8'), nl,

lines(1,Board),nl,!.

% imprime recursivamente

as linhas do tabuleiro

lines(\_,[]).

lines(N,[L|Ls]):-

write(N), write('. ') ,show\_line(L), nl,

N1 is N+1,

lines(N1, Ls).

% imprime recursivamente os caracteres de cada linha do tabuleiro

show\_line([]).

show\_line([El|Es]):-

translate(El),

show\_line(Es).

% imprime o tabuleiro

printBoard(Board):-

board(Board),

view(Board).

% imprime o tabuleiro

printBoardInit(Board):-

board\_Int(Board),

view(Board).

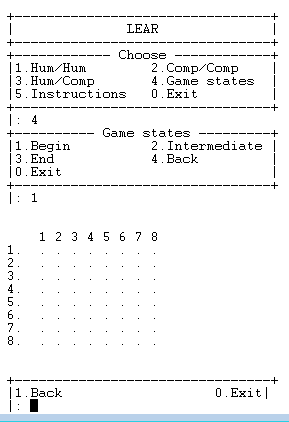
% imprime o tabuleiro

printBoardEnd(Board):-

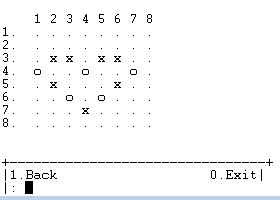
board\_End(Board),

view(Board).

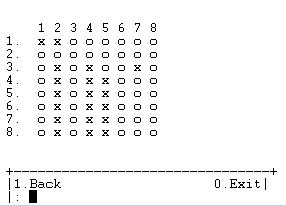
**VISUALIZAÇÃO INICIAL**



**VISUALIZAÇÃO INTERMÉDIA**



**VISUALIZAÇÃO FINAL**



## Lista de jogadas válidas

Para fazer a verificação de quais as jogadas possíveis foi utilizado o predicado validMoves(+Board, +Player, -ListOfMoves) que verifica quais os locais do tabuleiro que ainda não estão ocupados por uma peça, dentro dos limites do tabuleiro, devolvendo uma lista com os mesmos.

## Execução de jogadas

Para a execução de uma jogada é executado o predicado playPvP, no caso do jogo ser humano contra humano.

Nesta situação é invocado o predicado makeMove(+Board1, -Board2, +Player1) que irá pedir ao utilizador para introduzir as coordenadas do local onde quer inserir a peça. Dentro deste predicado , após verificada a validade das coordenadas inseridas, são chamados os predicados getPiece(+CurrBoard, +Line, +Col, -Piece), para verificar que ainda não existe nenhuma peça naquele local, e insertPiece(+CurrBoard, +Line, +Col, +Player, -NewBoard), que insere a peça correspondente ao jogador no tabuleiro.

## Avaliação do tabuleiro

A avaliação do tabuleiro será feita quando a jogada é feita pelo computador. Sendo assim, após verificar quais são as jogadas possíveis e a partir das mesmas testar quais são os locais onde, ao ser colocada a peça, irá haver um maior número de flips, de forma a ter o maior número possível de peças “friendly” no tabuleiro.

## Final do jogo

O jogo termina quando o tabuleiro está completamente preenchido, condição que é verificada através de contadores que guardam o número de cada tipo de peças. Quando o jogo termina é invocado o predicado gameOver(+Komi, +Cb, +Cw) que irá calcular as pontuações finais, somando o Komi definido no início do jogo ao jogador que não foi o último a colocar uma peça no tabuleiro (no caso do tabuleiro 8x8 o último jogador irá ser sempre o White uma vez que quem começa é o Black e o tabuleiro é par).

## Jogada do Computador

Nos casos em que as jogadas são efetuadas pelo computador, são chamados os predicados playPvC, no caso humano contra computador e playCvC, no caso computador contra computador.

No nível fácil, existe o predicado botRandom(-R,-C) que gera aleatoriamente as coordenadas onde vai ser inserida a peça, verificando que é um local onde a jogada é possível, e o makeMoveRandomBot(+CurrBoard, -NewBoard, +Player, +Line, +Col) que insere a peça no local gerado.

//falta o nível hard

No nível difícil será feita uma avaliação do tabuleiro, de forma a que o computador jogue no local que lhe permite aumentar o máximo possível a sua pontuação.

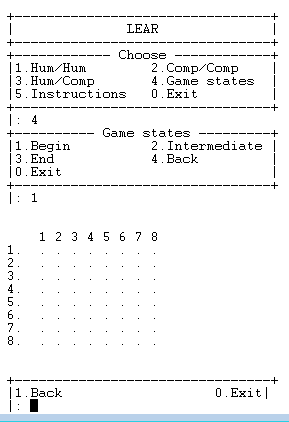
# Interface com o utilizador

A presente implementação do Lear pode ser jogada através do SICStus Prolog, executando o predicado play.

A execução deste predicado faz com que seja apresentado no ecrã o menu inicial, no qual são apresentadas 6 opções:

* Hum/Hum: permite iniciar um jogo com dois utilizadores
* Comp/Comp: avança para um menu que permite escolher a dificuldade para depois iniciar um jogo entre dois computadores
* Hum/Comp: avança para um menu que permite escolher a dificuldade para depois iniciar um jogo entre um jogador humano e um computador
* Game States: avança para um menu que permite escolher ao utilizador qual dos exemplos de visualização de tabuleiro quer ver
* Instructions: permite visualizar as instruções do jogo
* Exit: permite sair do jogo.

Quando a modalidade do jogo envolve um jogador humano, irá aparecer, quando for a sua vez, texto que irá pedir ao utilizador para introduzir as coordenadas onde quer colocar a peça.

//FALTAM AS OUTRAS IMAGENS

# 

# Conclusões

A implementação deste projeto trouxe algumas dificuldades, dando especial ênfase à implementação do *flip* das peças.

Esta componente mostrou-se bastante complexa, uma vez que a sua implementação seria extremamente extensa e exaustiva ou iria exigir um encadeamento sucessivo de recursividade, pela natureza iterativa da questão.

A extensão do projeto gerou ainda alguma dificuldade de organização do código desenvolvido, apesar do mesmo estar organizado em vários ficheiros diferentes.

Implementar este jogo permitiu contactar com uma linguagem que promove um tipo diferente de pensamento lógico, em relação às linguagens mais habitualmente utilizadas pelos alunos (C++, por exemplo).

O código desenvolvido teria ainda mais espaço para melhorar e poderia estar mais completo.

No final, este foi um projeto desafiante, muito pertinente para poder pôr em prática os conhecimentos em Prolog.

# Bibliografia

# Anexos