Eximo

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Eximo_2:

João Pedro Milano da Silva Cardoso - 200900579
Diogo Alexandre Soares Gomes - 201106586

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, s/n, 4200-465 Porto, Portugal

9 de Novembro de 2014

Resumo

Foi-nos proposta a realização de um trabalho que consiste na programação de um jogo de tabuleiro na linguagem declarativa Prolog, com o objectivo de aprofundar os nossos conhecimentos nesta linguagem, útil no campo de pesquisa de Inteligência Artificial.

Devido à nossa experiência em linguagens procedimentais como C, C++ e Java, o trabalho foi aproximado, em parte, de um modo procedimental.

Começamos por desenvolver a representação e impressão do tabuleiro, desenvolvendo seguidamente o ciclo principal de jogada, ou turno, passando para alterações simples do tabuleiro, os movimentos básicos de moviemento, salto e captura. A seguir foram desenvolvidos aspectos mais complexos como jogadas obrigatórias e recuperação de peças, completando o modo de jogo manual.

Foi então desenvolvido o primeiro e segundo nível de dificuldade do computador, o qual aje de modo aleatório na escolha de peças e jogadas e no primeiro nível e prioriza a captura no segundo nível, sendo de resto aleatório nas suas escolhas.

No entanto, depois de várias tentativas, não conseguimos desenvolver o terceiro nível de dificuldade, que dependeria do computador escolher a melhor jogada possível no seu turno, sendo este o único aspecto do trabalho não implementado com sucesso.

É possível concluir o funcionamento correcto de todos os aspectos implementados.

Índice

Contents

1.Introdução

O presente relatório foi desenvolvido na âmbito da unidade curricular de Programação Lógica do Curso de Mestrado Integrado em Engenharia Informática e de Computação.

O trabalho consiste na programação de um jogo de tabuleiro na linguagem declarativa Prolog, com o objectivo de aprofundar o nosso conhecimento nessa linguagem.

Como o Prolog utiliza certas técnicas de programação consideradas difíceis ou avançadas, a destacar entre estas recursão, aprender Prolog pode ser um bom modo de melhorar a percepção de técnicas como recursão, procura em árvores e programação de restrição lógica.

O relatório está dividio em três partes principais:

1. Explicação das regras do jogo, com exemplos ilustrativos para os vários tipos de jogadas e posições possíveis
2. Explicação da lógica do jogo, que inclui representação e visualização do tabuleiro, obtenção de jogadas válidas e a sua execução, condições de final de jogo e a lógica do computador
3. Interface de texto para o utilizador

2. O Jogo Eximo

Eximo é um jogo de tabuleiro da familia das damas que foi lançado a 1 de Fevereiro de 2013 por Matteo Perlini[1].

O tabuleiro do jogo é uma matriz de 8x8 em que cada jogador começa com 16 peças idênticas.

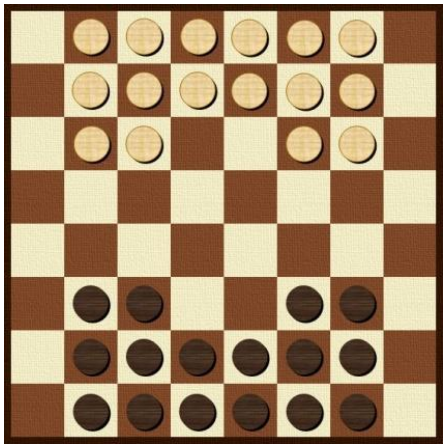

Fig. 1 - Tabuleiro do jogo

Objectivo

Capturar todas as peças do oponente ou chegar a um impasse de forma a que o oponente não possa movimentar as suas peças.

Turnos

Em cada turno o jogador pode movimentar uma peça sua ou capturar peça(s) inimigas.

[1] http://www.boardgamegeek.com/boardgamedesigner/41103/matteo-perlini

Movimento (sem captura)

Uma peça pode mover-se em três direções durante um movimento: frente, diagonal direita e diagonal esquerda.

- Movimento ordinário: a peça move-se para um quadrado adjacente vazio (nas três direções possíveis).

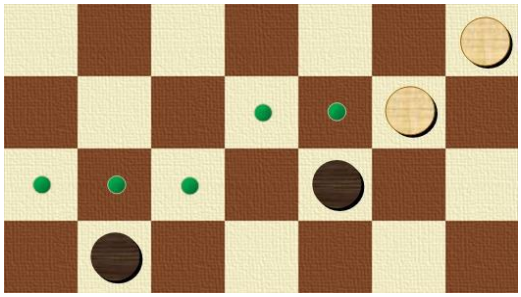

Fig. 2 - Movimento ordinário

- Salto: a peça salta por cima de uma peça amiga (nas três direções possíveis) se o quadrado adjacente a esta está vazio (na mesma direção do salto), colocando a peça que fez o salto no quadrado. Se a peça pode continuar a saltar então é obrigada a fazê-lo. Durante o salto a peça não pode capturar. (Não é obrigatório escolher o caminho mais longo)

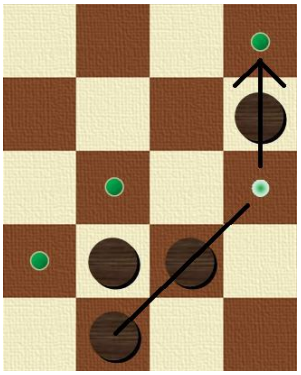

Fig.3 - Salto

Captura

A peça pode capturar em cinco direções: frente, diagonal direita, diagonal esquerda, direita e esquerda.

- Captura: a peça salta por cima de uma peça inimiga (nas cinco direções possíveis) se o quadrado adjacente a esta está vazio (na mesma direção do salto), colocando a peça que fez a captura no quadrado. A peça capturada é retirada do tabuleiro. Se a peça pode continuar a capturar então é obrigada a faze-lo. A captura é obrigatória e deve continuar a capturar dentro do possível. (Não é obrigatório capturar o maior número de peças)

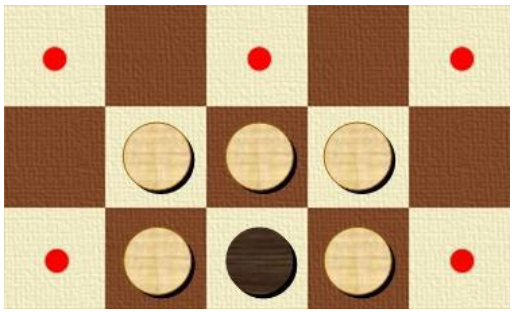

Fig. 4 - Captura (Pontos vermelhos indicam a posição final da peça preta após a captura)

A última linha

Quando uma peça chega ao fim do tabuleiro ela é retirada imediatamente e o jogador que a controla recebe duas peças que deve colocar imediatamente num quadrado livre nas suas primeiras duas linhas, exceto nos quatro quadrados dos lados.

Locais possíveis:

- Preto: b1, c1, d1, e1, f1, g1, b2, c2, d2, e2, f2, g2;
- Branco: b8, c8, d8, e8, f8, g8, b7, c7, d7, e7, f7, g7.

Se a peça chegar à ultima linha e não existe espaço para colocar peças o jogador perde essa peça. Se existir apenas um quadrado livre então o jogador recebe só uma peça.

Estado inicial

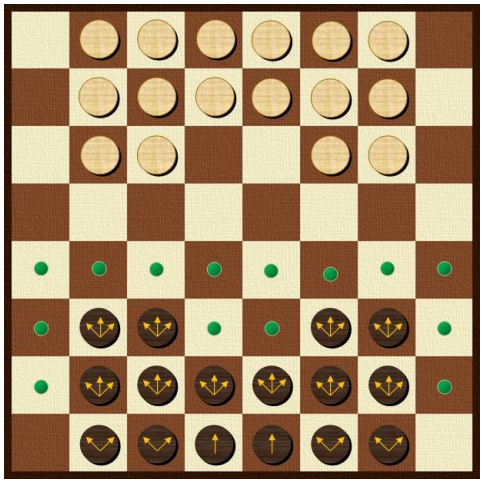

Fig. 5 - Movimentos iniciais possíveis das peças preta

3. Lógica do Jogo

3.1. Representação do Estado de Jogo

A representação do estado do tabuleiro é efetuada usando uma lista que contém oito elementos, cada um contém por sua vez uma lista com oito números. Estes números, 64 no total, indicam que peça se encontra num determinado espaço, bem como se o espaço está vazio. O número 0 indica um espaço vazio, o número 1 indica um espaço ocupado por uma peça branca e o número 2 indica um espaço ocupado por uma peça preta. Os dois últimos números representam as peças pretas capturadas e peças brancas capturadas.

O tabuleiro, com as peças na sua posição inicial, pode então ser representado por:

```
boardInitial([[0,1,1,1,1,1,1,0],
              [0,1,1,1,1,1,1,0],
              [0,1,1,0,0,1,1,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,2,2,0,0,2,2,0],
              [0,2,2,2,2,2,2,0],
              [0,2,2,2,2,2,2,0]]-0-0).
```

Fig. 6 - Representação do tabuleiro com peças em posição inicial

```
boardInitial([[0,0,1,1,2,1,1,0],
              [0,0,1,0,0,1,1,0],
              [0,1,0,0,0,1,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,0,0,0,0,0],
              [0,2,0,0,0,2,2,0],
              [0,2,2,0,2,0,2,0],
              [0,2,0,2,2,0,2,0]]-4-7).
```

Fig.7 - Tabuleiro com 7 peças brancas capturadas e 4 peças pretas capturadas, com uma peça preta na linha final do lado branco. O jogador preto pode agora trocar uma peças por duas.

```
boardInitial([[0,0,1,1,1,1,1,0],
              [0,0,1,0,0,1,1,0],
              [0,0,0,1,0,1,0,0],
              [0,0,0,0,0,0,0,0],
              [0,0,0,1,0,0,0,0],
              [0,2,2,0,0,2,2,0],
              [0,2,2,0,2,0,2,0],
              [0,2,0,2,2,0,2,0]]-4-5).
```

Fig.8 - Tabuleiro com 5 peças brancas capturadas e 4 peças pretas capturadas. A peça preta na posição C6 pode capturar as peças brancas nas posições D5 e D3.

3.2. Visualização do Tabuleiro

A visualização do tabuleiro é efetuado com o seguinte código:

```
% MAIN PRINTING FUNCTION
printBoard(Board-CapByW-CapByB) :- nl, printCaptured(CapByW), nl, nl,
                                   printHeader, nl,
                                   printlists(Board,0),
                                   printCaptured(CapByB), nl.
% PRINTS THE CAPTURED PIECES
printCaptured(Number) :- write('Captured Pieces: '), write(Number).

% PRINTS THE BOARD
printlists([],_Count):- printLine, nl.
printlists([FirstList|OtherLists],Count) :- printLine, nl,
                                            write(Count), Sum is Count+1,
                                            printlist(FirstList),
                                            printlists(OtherLists, Sum).
% PRINTS A LINE OF THE BOARD
printlist([]) :- write('|'), nl.
printlist([FirstElem|OtherElems]) :- write('|'),
                                     printElem(FirstElem),
                                     printlist(OtherElems).

% PRINTS THE HEADER OF THE BOARD
printLine :- write('------------------').
printHeader :- write(' |0|1|2|3|4|5|6|7|').

% PRINT THE TURN OF THE PLAYER
printPlayerTurn(Player) :- Player, !, nl, write('White Turn'), nl.
printPlayerTurn(_Player) :- nl, write('Black Turn'), nl.

% PRINT ELEMENTS OF THE BOARD
printElem(Piece) :- Piece=0, write(' ').
printElem(Piece) :- Piece=1, write('w').
printElem(Piece) :- Piece=2, write('b').
```

Fig. 9 - Código de impressão do tabuleiro

Usando o predicado start, o resultado é:

```
Captured Pieces: 0
 |0|1|2|3|4|5|6|7|
-------------------
0| |w|w|w|w|w|w| |
-------------------
1| |w|w|w|w|w|w| |
-------------------
2| |w|w| | |w|w| |
-------------------
3| | | | | | | | |
-------------------
4| | | | | | | | |
-------------------
5| |b|b| | |b|b| |
-------------------
6| |b|b|b|b|b|b| |
-------------------
7| |b|b|b|b|b|b| |
-------------------
Captured Pieces: 0

White Turn
Starting Positions
Select Start Position
```

Fig. 10 - Impressão do tabuleiro inicial

3.3. Lista de Jogadas Válidas

A obtenção de jogadas válidas é efetuada em dois passos.

Primeiro, são apresentadas as todas as peças no tabuleiro que pertencem ao jogador, para que o mesmo escolha uma delas. O predicado responsável pelo primeiro passo, processStartPositions, encontra-se na secção “Process Movement Positions”.

```
processStartPositions(Player, Board, StartPositions, StartRow-StartCol, GameType) :-
    getStartPositions(Player, Board, StartPositions),
    length(StartPositions, Size),
    Size > 0,
    chooseStartingPosition(Player, Board, StartPositions, StartRow-StartCol, GameType).
```

Fig. 11 - processStartPositions, que apresenta e pede ao jogador uma posição initial

O predicado getStartPositions encontra todas as peças percorrendo todos os espaços do tabuleiro procurando pelo número 1 para o jogador branco (true) ou pelo número 2 no caso do jogador preto (false).

As posições inicias são impressas na forma de uma lista.

```
White Turn
Starting Positions
Select Start Position
[0-1,0-2,0-3,0-4,0-5,0-6,1-1,1-2,1-3,1-4,1-5,1-6,2-1,2-2,2-5,2-6]
|:
```

Fig. 12 - Impressão das peças iniciais do jogador branco

Após o jogador escolher uma das peças disponíveis, essa posição inicial é usada para procurar todas as posições finais possíveis nos três tipos de jogada: movimento, salto e captura. Os predicados responsáveis por este segundo passo encontram-se na secção “Process Movement Positions”.

```
processMovePositions(Player, Board, StartRow-StartCol, MovePositions):-
    getAllMovePositions(Player, Board, StartRow-StartCol, MovePositions),
    write('Valid Move Destinations for Piece:'),nl,
    write(MovePositions), nl.
```

Fig. 13 - Predicado que apresenta os destinos possíveis para movimento

Como no caso das posições inicias, as posições finais são também apresentadas como listas, uma para cada tipo de jogada.

```
White Turn
Starting Positions
Select Start Position
[0-1,0-2,0-3,0-4,0-5,0-6,1-1,1-2,1-3,1-4,1-5,1-6,2-1,2-2,2-5,2-6]
|: 1-3.
Valid Move Destinations for Piece:
[2-3,2-4]
Valid Jump Destinations for Piece:
[3-1]
Valide Capture Destinations for Piece:
[]
Select Play EndingRow EndingColumn
|:
```

Fig. 14 - Impressão das jogadas possíveis

3.4. Execução das Jogadas

A leitura e validação da jogada é efectuada usando o predicado readPlay, na secção “Read Plays”.

No tipo de jogo 1 - Humano contra Humano, é sempre pedido input ao utilizador através do interface de texto. No tipo de jogo 2 - Humano contra Computador - é pedido input ao utilizador quando é o turno do jogador branco e é pedida uma escolha do Computador quando

é o turno do jogador preto. No tipo de jogo 3 - Computador contra Computador - é sempre pedida um escolha ao Computador.

```
readPlay(_Player, GameType-Dif, PlayType-EndRow-EndCol, MovePositions, JumpPositions, CapturePositions):-
    GameType = 1,
    readPlayHuman(PlayType-EndRow-EndCol, MovePositions, JumpPositions, CapturePositions).

readPlayHuman(PlayType-EndRow-EndCol, MovePositions, JumpPositions, CapturePositions):-
    write('Select Play EndingRow EndingColumn'), nl,
    read(PlayType-EndRow-EndCol),
    checkValidChoiceGeneral(PlayType-EndRow-EndCol, MovePositions, JumpPositions, CapturePositions).
```

Fig. 15 - readPlay com tipo de jogo 1 pede input ao utilizador

A validação da jogada é efetuada pelo predicado checkValidChoicesGeneral. Como são criadas listas com os possíveis destinos da peça este predicado, que se encontra na secção "Choice Validation", apenas verifica se a escolha pertence à lista correspondente à jogada escolhida.

Após apresentação das jogadas possíveis o jogador introduz a sua jogada no formato Play-Row-Column, variável composta passada para o predicado executePlay.

Este predicado, juntamente com os restantes predicados responsáveis pela execução de jogadas encontram-se na secção "Execute Play".

Cada um destes predicados funciona em 3 passos, 2 para o caso de um movimento:

- Modificação do tabuleiro correspondente à jogada
- Verificar se é necessário e possível salvar peças
- Verificar se existem jogadas obrigatórias a serem executadas

Existem três predicados executePlay, um para cada tipo de jogada.

```
% MOVEMENT
executePlay(Player, GameType, PlayType, Board-CapW-CapB, StartRow-StartCol, EndRow-EndCol, NewBoard2-NCapW-NCapB) :-
    PlayType = 7, % movement
    movePiece(Player, Board, StartRow-StartCol, EndRow-EndCol, NewBoard),
    savePiece(Player, GameType, NewBoard-CapW-CapB, EndRow-EndCol, NewBoard2-NCapW-NCapB).

% JUMP
executePlay(Player, GameType, PlayType, Board-CapW-CapB, StartRow-StartCol, EndRow-EndCol, NewBoard3-NCapW2-NCapB2) :-
    PlayType = 8, % jump
    jumpPiece(Player, Board, StartRow-StartCol, EndRow-EndCol, NewBoard),
    getAllJumpPositions(Player, NewBoard-CapW-CapB, EndRow-EndCol, JumpPositions),
    savePiece2(Player, GameType, NewBoard-CapW-CapB, EndRow-EndCol, NewBoard2-NCapW-NCapB, JumpPositions, NewJumpPositions),
    mandatoryJump(Player, GameType, 8, NewBoard2-NCapW-NCapB, EndRow-EndCol, NewJumpPositions, NewBoard3-NCapW2-NCapB2).
```

Fig. 16 - Execute Play para movimento e salto

Numa jogada é colocado no tabuleiro, na posição inicial da peça um 0, e na posição final 1 ou 2, através do predicado movePiece, jumpPiece ou capturePiece. No caso da captura é colocado no espaço intermédio um 0. É então necessário verificar se a peça chegou à primeira linha do oponente, que é feito com o predicado savePiece. Se não estiver na linha final, o turno muda.

Para salvar uma peça é necessário que um movimento, salto ou captura coloque a peça na primeira linha do adversário, linhas 0 e 7, e que existam espaços livres na primeira e segunda linha do jogador, entre as colunas 1 e 6. Os predicados responsáveis pela implementação encontram-se na secção “Recovering Pieces”.

```
savePiece(Player, GameType, Board-CapW-CapB, Row-Col, NewBoard2-CapW-NCapB2) :-
    Player,
    Row = 7,
    changeBoard(Board, Row-Col, 0, NewBoard),
    NCapB is CapB + 1,
    getPieceBack(Player, GameType, NewBoard-CapW-NCapB, NewBoard2-CapW-NCapB2,0).

getPieceBack(Player, GameType, Board-CapW-CapB, NewBoard2-CapW-NCapB2, Count) :-
    Player,
    Count < 2,
    CapB > 0,
    getAllFreePositions(Player, Board, FreePositions),
    length(FreePositions, Size), Size > 0,
    chooseSpaceForPieceSave(Player, GameType, FreePositions, Row-Col),
    changeBoard(Board, Row-Col, 1, NewBoard),
    NewCount is Count + 1,
    NCapB is CapB - 1,
    getPieceBack(Player, GameType, NewBoard-CapW-NCapB, NewBoard2-CapW-NCapB2, NewCount).
```

Fig. 17 - savePiece e getPieceBack, no caso do jogador Branco

A peça que chega á ultima fila é retirada do jogo e conta como uma peça capturada.

O predicado getPieceBack verifica se é possível recuperar peças, verificando que existem posições e peças disponíveis, pedindo então ao jogador ou computador para selecionar os espaços para colocar peças.

```
Recovering Piece...
Available Positions For Recovered Piece:
[0-1,0-2,0-4,0-5,0-6]
Choose Position:
|:
```

Fig. 18 - Impressão das posições para recuperar peça

Depois deste passo de recuperação de peças, é necessário verificar se existem ou não jogadas obrigatórias a serem realizadas no caso de saltos ou capturas.

```
mandatoryJump(_,_,_, Board-CapW-CapB, _, [], Board-CapW-CapB).

mandatoryJump(Player, GameType, _PlayType, Board-CapW-CapB, StartRow-StartCol, PositionList, NewBoard3-NCapW2-NCapB2) :-
    GameType = 1,
    printBoard(Board-CapW-CapB),nl,
    mandatoryHuman(PositionList, EndRow-EndCol),
    executePlay(Player, 1, 8, Board-CapW-CapB, StartRow-StartCol, EndRow-EndCol, NewBoard3-NCapW2-NCapB2).
```

Fig. 19 - Predicado de jogada obrigatória no caso de um salto

Termina assim uma jogada completa.

3.4. Avaliação do Tabuleiro

A jogada de maior valor será aquela que capture o maior número de peças, visto que todas as peças são do mesmo valor e que uma das duas condições de vitória é capturar todas as peças do adversário.

Este predicado de avaliação receberia todas as posições de início e devolveria uma lista. Cada elemento desta lista seria ele mesmo uma lista, contendo na primeira posição a posição inicial e nas restantes as posições de captura para cada jogada. A melhor jogada seria então a jogada correspondente à lista mais longa. O número de peças capturadas seria o comprimento da lista menos um.

O predicado de avaliação do tabuleiro não foi implementado com sucesso.

3.5. Final do Jogo

O final do jogo é verificado no início de cada ciclo principal, ou turno, através do predicado canStillPlay, na secção “Can Still Play”.

```
canStillPlay(Board-CapW-CapB, Player) :-
    Player,
    existsOnBoard(Board, 1),
    canStillPlay3(Player, Board-CapW-CapB).
```

Fig. 20 - Predicado canStillPlay para o jogador branco

Este predicado verifica duas condições:

- Existe pelo menos uma peça pertencente ao jogador
- Existe pelo menos uma jogada que o jogador pode realizar

O predicado existsOnBoard apenas verifica se existe pelo menos uma peça do jogador.

O predicado canStillPlay3 cria uma lista, percorrendo todas as peças do jogador, que irá conter todos os destinos possíveis de todas as peças. Nesta altura do turno é irrelevante se as jogadas são de captura, salto ou movimento, apenas é necessário verificar se existe pelo menos uma jogada.

Se o jogador branco não conseguir jogar, é impressa a mensagem “Black Won”.

Se o jogador preto não conseguir jogar, é impressa a mensagem “White Won”.

3.6. Computador

Nos tipos de jogos 2 e 3, é necessário pedir ao computador para escolher uma jogada. O computador tem dois níveis de dificuldade. No nível um, o computador escolhe uma peça aleatória e uma jogada aleatória. No nível 2 escolhe uma peça e prioriza a captura de peças. Como a avaliação do tabuleiro não foi implementada, não é garantido que o computador faça uma escolha óptima.

Ao escolher uma posição inicial, é verificado se a peça tem pelo menos uma jogada possível, de modo a que o computador não fique trancado com uma peça que não se possa mover.

```
computerChoosePlay(Dif, PlayType-EndRow-EndCol, MovePositions, JumpPositions, CapturePositions):-
    Dif = 1,
    repeat,
        random(7,10,PlayType),
        computerGetPlay(PlayType, EndRow-EndCol, MovePositions, JumpPositions, CapturePositions),
        write('Computer Chose: '), write(PlayType),
        write('  Moving To: '), write(EndRow-EndCol),nl.

computerGetPlay(_PlayType, _EndRow-_EndCol, [], [], []).

computerGetPlay(PlayType, EndRow-EndCol, MovePositions, _JumpPositions, _CapturePositions):-
    PlayType = 7,
    computerChoose(MovePositions, EndRow-EndCol),
    checkValidChoice(EndRow-EndCol, MovePositions).

computerChoose([],_).
computerChoose(Positions, Row-Col) :-
    length(Positions, Length),
    random(0, Length, Index),
    nth0(Index, Positions, Row-Col).
```

Fig. 21 - Predicado de escolha de jogada para dificuldade 1 e escolha do computador para um movimento

4. Interface com o Utilizador

O jogo é iniciado usando o predicado start. No início do jogo é pedido ao jogador que escolha o tipo de jogo e a dificuldade.

```
| ?- start.
Select Game Mode and Difficulty
1 - H/H 2 - H/C 3- C/C
|:
```

Fig. 22 - Escolha do tipo de jogo e dificuldade

A para a selecção da jogada o jogador insere a jogada no formato Jogada-Linha-Coluna. A linha e a coluna são a posição final. Existem três valores válidos para a Jogada:

- 7, no caso de ser Movimento
- 8, no caso de ser Salto
- 9, no caso de ser Captura

Excepto no caso da escolha da jogada, em que se lêem três números, o input é sempre feito no mesmo formato em que é impresso no ecrã, Linha-Coluna.

Conclusão

O jogo foi implementado com sucesso em modo manual. Todas as regras estão a ser observadas e cumpridas. No entanto, a implementação do computador não foi completamente executada, ficando este incapaz de selecionar sempre a jogada ótima. Esta avaliação do tabuleiro é o único aspecto não implementado do trabalho.

É possível que existam predicados que consigam ser melhorados do ponto de vista de performance, visto que a optimização do programa não foi nosso alvo.

É possível também concluir que em determinados casos conhecimento de linguagens procedimentais pode influenciar a estrutura de um projecto de Prolog, uma linguagem de natureza declarativa, no entanto podendo esta última ajudar a reforçar conceitos de linguagens procedimentais como a recursão.

Bibliografia

Boardgamegeek.com, (2013). *Matteo Perlini | Board Game Designer | BoardGameGeek*. [online] Available at: http://www.boardgamegeek.com/boardgamedesigner/41103/matteo-perlini

Perlini, M. (2013). *Eximo | Board Game | BoardGameGeek*. [online] Boardgamegeek.com. Available at: http://www.boardgamegeek.com/boardgame/137916/eximo

Sicstus.sics.se, (2014). *SICStus Prolog*. [online] Available at: https://sicstus.sics.se/sicstus/docs/latest4/html/sicstus.html/

Stackoverflow.com, (2010). *Random items in Prolog*. [online] Available at: http://stackoverflow.com/questions/2261238/random-items-in-prolog