Callanish

Relatório Final

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Callanish_2:

Pedro Dias Faria – ei11167

Rui Filipe de Oliveira Donas-Botto Figueira – ei11021

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

11 de Novembro de 2014

Resumo

Este relatório tem como objectivo mostrar como foi desenvolvido o jogo Callanish em Prolog, no âmbito da cadeira de Programação em Lógica do primeiro semestre do 3º ano do Mestrado Integrado em Engenharia Informática e Computação.

Índice

1 - Introdução

Este projecto teve como principal objectivo melhorar o nosso conhecimento sobre a linguagem de programação lógica Prolog, através da implementação do Callanish, um jogo de tabuleiro para dois jogadores. Jogo esse que, devido a essa natureza, se caracteriza por um conjunto de regras específicas de movimentação de peças e condições de terminação do jogo.
O nosso papel foi o de implementar as respectivas funcionalidades e regras desse mesmo jogo, tendo no final deste projecto uma versão funcional em Prolog do mesmo em que o jogador pode jogar contra um outro jogador.

2 - O Jogo: Callanish

Callanish é uma vila situada no nordeste da ilha de Lewis, na Escócia, onde podemos encontrar algumas das mais antigas estruturas megalíticas da Europa. Conhecida pelas Callanish Stones, que datam a 3000 BC, este conjunto de 13 pedras formam um círculo que se assume ter sido um observatório lunar pré-histórico.

Figura 1 e 2. Callanish Stones, Lewis, Escócia.

É este conjunto de pedras que serviu de inspiração para o nome e funcionamento deste jogo de tabuleiro cuja intenção é o de alinhar um certo número de pedras da mesma cor, horizontal ou verticalmente, nas linhas de um tabuleiro.

Feito para dois jogadores, o jogo decorre num tabuleiro 9x9, onde cada um possui um conjunto de pedras que poderá colocar ao longo do tabuleiro. Um possui pedras brancas e o outro, pedras pretas, sendo que ainda existem quadrados que podem ser colocados para modificar o design do tabuleiro.

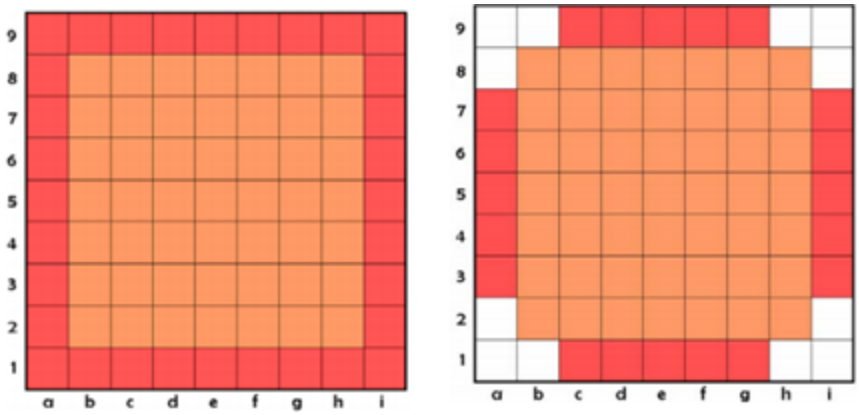


Figura 3 e 4. Exemplos de tabuleiro. (3) Standard; (4) Modificado

Regras:

O jogador com as peças brancas inicia o jogo colocando uma pedra numa posição do tabuleiro, sendo que o outro jogador procede a fazer o mesmo. A partir da primeira jogada, a cada turno, cada jogador deverá colocar duas novas pedras e remover uma das suas antigas do tabuleiro seguindo as seguintes regras:

- Os quadrados nos quais as duas novas pedras são colocadas têm de estar conectadas a um quadrado do qual uma pedra está a ser removida, através de um movimento da peça de cavalo de xadrez (movimento em forma de L).
- Ao serem colocadas de acordo com a regra acima, as pedras podem ser colocadas num quadrado vazio ou em cima de uma pedra do adversário, criando uma pilha, mas não em cima de pedras da mesma cor.
- Pilhas de pedras, não podem ser maiores do que duas pedras.
- A cor da pilha é definida pela pedra que está mais acima.
- Apenas a peça superior da pilha pode ser removida, sendo que a peça que está por baixo dela está fora do jogo temporariamente.

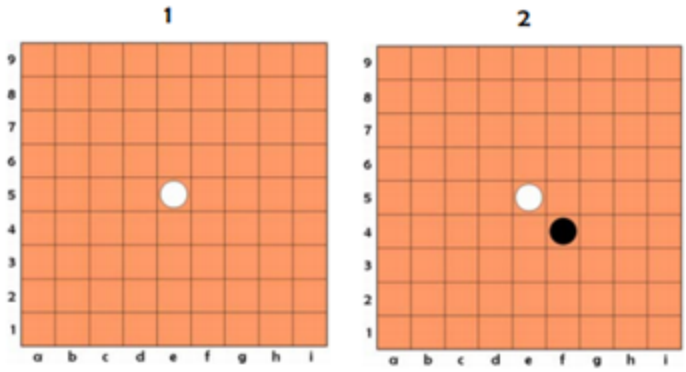


Figura 4 e 5. Primeira jogada de cada jogador:
1- Jogador branco colocou a 1ª peça
2- Jogador preto colocou a 1ª peça

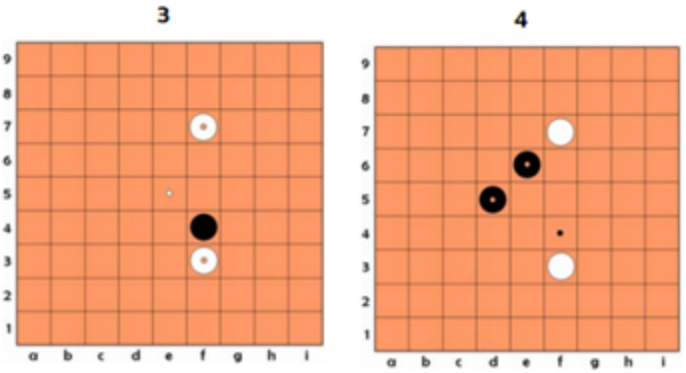


Figura 6 e 7. Segunda jogada dos jogadores:
3- Jogador branco
4- Jogador preto

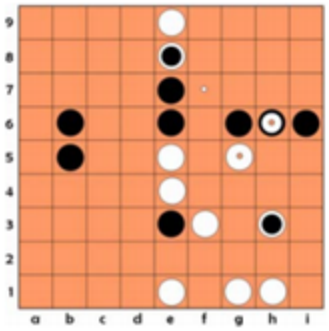


Figura 8. Exemplo de tabuleiro com pilhas de peças:
Pilhas em (e,8) e (h,3) consideradas como peças pretas;
Pilha em (h,6) considerada como peça branca;

Existem duas condições que levam ao final do jogo:

- Um dos jogadores consegue alinhar pelo menos 5 pedras da sua cor ou pilhas (contiguas ou não) ao longo de uma linha do tabuleiro (vertical ou horizontal);
- Se o oponente no turno a seguir á condição citada em cima, não consegue reduzir o número de pedras do adversário contíguas para 4, através da colocação de 1 ou 2 pedras em cima de qualquer uma delas.

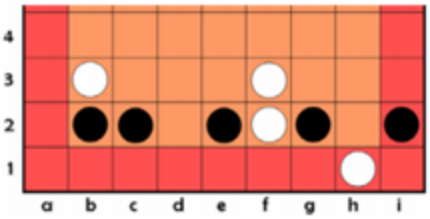


Figura 9. Exemplo de tabuleiro em que o jogador preto consegue ambas as condições de vitória

3 - Lógica do Jogo

3.1 - Representação do Estado do Jogo

Callanish é um jogo no qual as peças em jogo se podem encontrar sozinhas no tabuleiro ou com alguma peça colocada em cima delas, dando origem no máximo a uma pilha de duas pedras na mesma posição. Como tal, para representação do jogo em Prolog optamos por usar uma lista de listas. Ou seja, uma lista que representa o tabuleiro, que por sua vez possui uma lista para cada linha do tabuleiro, onde cada elemento dessa linha é uma célula do tabuleiro.

A matriz correspondente ao estado inicial do jogo é:

```
board([[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]]).
```

Tabuleiro 1. Tabuleiro inicial

Cada posição do jogo é representada por um inteiro. Nesta representação apenas estão 0's contidos nas listas, para representar cada posição como vazia.

Após algumas jogadas, um possível exemplo das posições intermédias de um jogo de Callanish seria:

```
board([[0,0,0,0,2,0,0,0,0],
       [0,0,0,0,3,0,0,0,0],
       [0,0,0,0,1,0,0,0,0],
       [0,1,0,0,1,0,1,4,1],
       [0,1,0,0,2,0,2,0,0],
       [0,0,0,0,2,0,0,0,0],
       [0,0,0,0,1,2,0,3,0],
       [0,0,0,0,0,0,0,0,0],
       [0,0,0,0,2,0,2,2,0]]).
```

Tabuleiro 2. Tabuleiro intermédio

Este estado de jogo representa o tabuleiro da figura 8, onde:

- Células com apenas um valor na cabeça, representam apenas uma pedra;
- 0 representa uma célula vazia;
- 1 representa uma peça preta;
- 2 representa uma peça branca;
- 3 representa uma pilha onde a pedra de cima é branca;
- 4 representa uma pilha onde a pedra de cima é preta;

E por último, uma situação de final de jogo:

```
board([[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],
       [0,2,0,0,0,2,0,0,0],
       [0,1,1,0,1,2,1,0,1],
       [0,0,0,0,0,0,0,2,0]]).
```

Tabuleiro 3. Tabuleiro final, com vencedor.

Representando o tabuleiro da figura 9, onde são cumpridas as condições de final de jogo.

3.2 - Visualização do Tabuleiro

Para visualizar em Prolog o tabuleiro com os vários estados ao longo do jogo foi usado o predicado `draw_board`. Este recebe a lista `board` com o estado do jogo e para fazer o output usa o predicado `draw_line` que por sua vez chama o predicado `draw_board_element`. Isto é feito recursivamente para todas as linhas do tabuleiro.

O código destas funções pode ser consultado no anexo 1.

Figura 10. Visualização gráfica do tabuleiro 1.

Figura 11. Visualização gráfica do tabuleiro 2.

As células do tabuleiro de jogo são representadas da seguinte forma:

- X – peça preta
- O – peça branca
- x – pilha preta
- o – pilha branca

3.3 - Lista de Jogadas Válidas

No processo do jogo, após um jogador escolher uma peça válida para a retirar do tabuleiro, actividade que recorre ao uso do predicado `pick_piece`, são mostradas no ecrã as celulas acessíveis através de um movimento de cavalo de xadrez, ou seja, as jogadas possíveis, usando o predicado `call_available_movements`.

3.4 - Movimentos

No Callanish, não existe movimentação de peças. Em vez disso, pedras são colocadas e retiradas do tabuleiro a cada jogada feita pelos jogadores. Como já foi referido em cima, no primeiro turno cada jogador coloca uma peça, sendo que a partir do segundo, colocam duas peças removendo uma das atualmente colocadas no tabuleiro.

No início do jogo é declarado um contador `Turn`, que será incrementado no final de cada turno. Este serve inicialmente para fazer a distinção dos primeiros dois turnos de jogo, em que os movimentos são apenas dois `put_piece`, pois cada jogador apenas coloca uma peça nesta fase do jogo. Após estes dois turnos, o `Turn` funciona para a fazer a mudança de turno entre os jogadores. O resto do jogo procede-se normalmente, sendo retiradas e colocadas peças todos os turnos através do predicado `movement`, que representa um turno de jogo e que é chamado no nosso predicado `play`, o nosso predicado principal.

3.5 - Validação de Jogadas

Para realizar jogadas, são pedidas as coordenadas de uma peça para a realização do tipo de movimento a ser feito (pick/put) nessa fase do turno. Este input é pedido através dos predicados `ask_for_pick` e `ask_for_put`, que após a obtenção das coordenadas, as avalia conforme o jogador atual e verifica se estas são possíveis para realizar uma jogada válida, através dos predicados `evaluate_pick` e `evaluate_put`.

`evaluate_pick` verifica se existe uma peça do jogador deste turno na celula escolhida pelo mesmo, através do predicado `get_cell`. Caso exista, chama-se duas vezes o predicado `ask_for_put`, que será utilizada para colocar duas peças no tabuleiro de jogo, avaliando a

validez das posições escolhidas através de `evaluate_put`, que verifica se estamos a escolher uma célula de destino acessível através de um movimento de cavalo de xadrez usando o predicado `evaluate_horse_movement`.

3.6 - Execução de Jogadas

Se as jogadas feitas forem válidas o tabuleiro é actualizado conforme as mesmas com o recurso ao uso dos predicados `refresh_remove` e `refresh_put`. Ambos percorrem todo o tabuleiro fazendo alterações no mesmo, sendo que o primeiro retira a peça selecionada em cada jogada pelo jogador e o segundo coloca as novas peças resultantes de cada nova jogada.

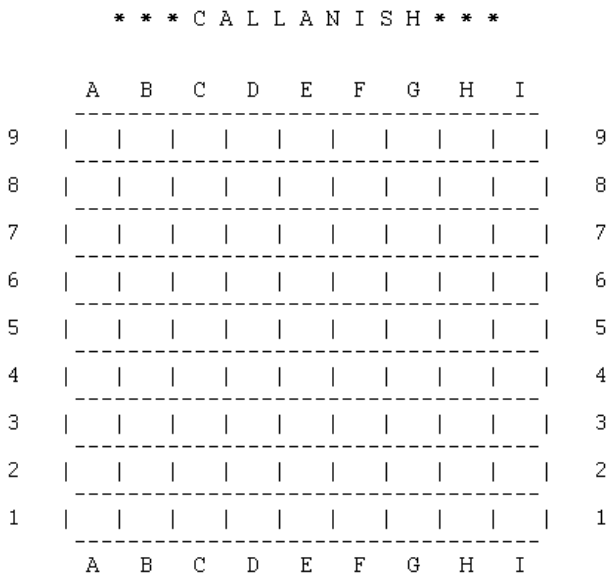
3.7 - Avaliação do Tabuleiro e final de jogo

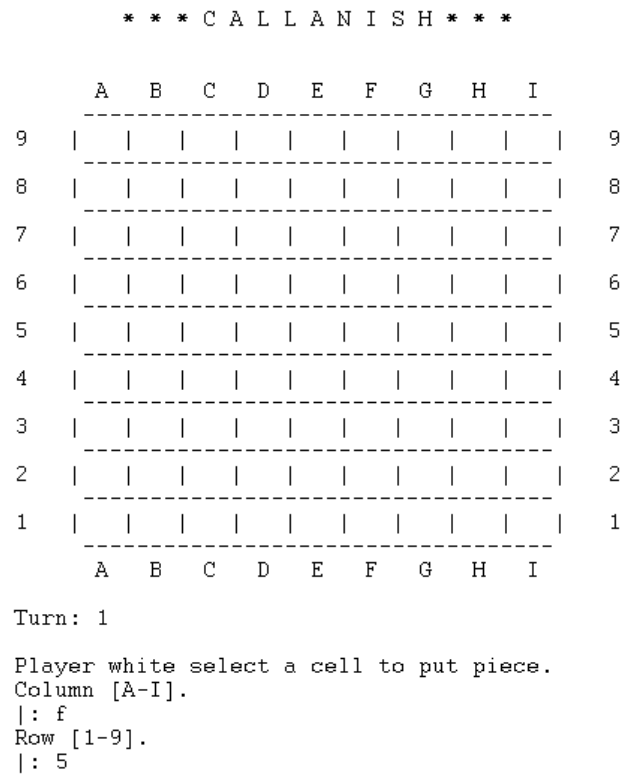
No final de cada turno é chamado o predicado check_end_game, onde e avaliada se existem 5 peças alinhadas, ou em linha (`fiveInARow`), ou em coluna (`fiveInACol`), a partir do predicado `fiveAlign`.

Estas verificações mudam o valor do parâmetro de verificação de fim de jogo `End`. Este será 2 ou 4 se um jogador estiver em Check ou 1 e 3 se estiver em Checkmate, terminando assim o jogo. Caso não aconteça o término do jogo, este continua, sendo mudado o turno ao fim de cada jogada, com o predicado `change_turn` e recomeçando um novo turno por parte do outro jogador.

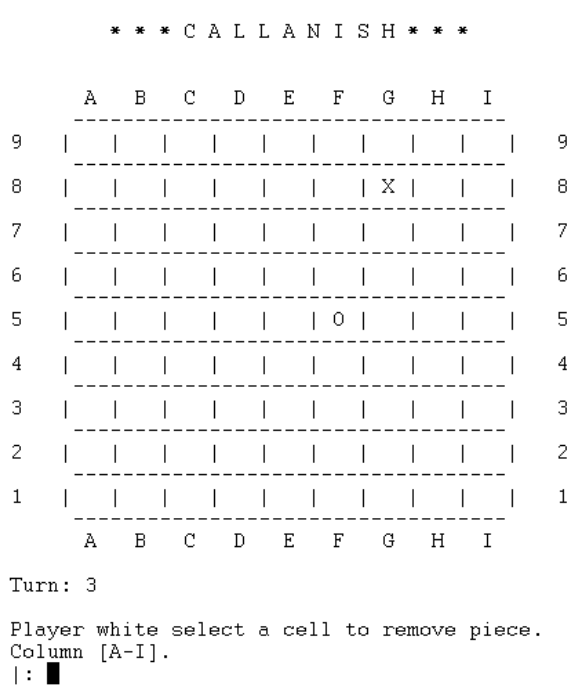
4 - Interface com o Utilizador

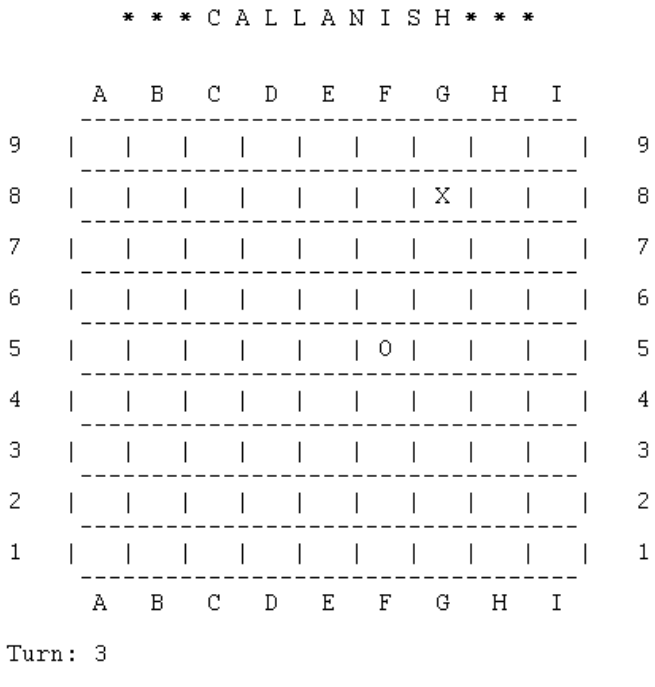
O programa começa por pedir a cada jogador para colocar a sua primeira peça, através do pedido das coordenadas do tabuleiro.



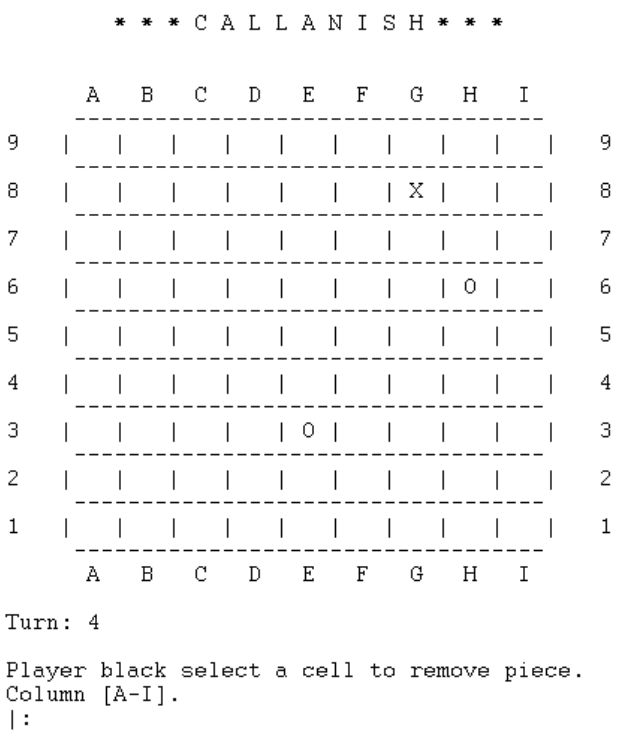


Após os dois primeiros turnos, será sempre pedido a cada jogador as coordenadas da peça a remover, mostrando as celulas possíveis para as novas peças.

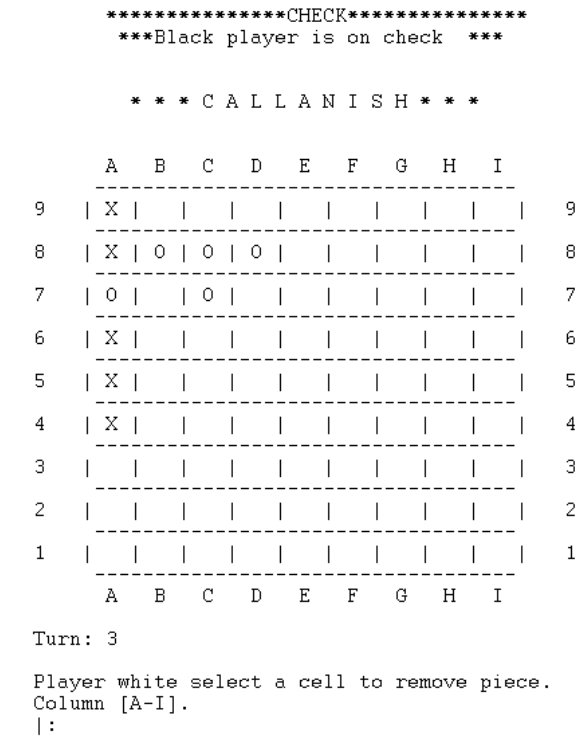




Após a colocação de cada peça, o tabuleiro é atualizado visualmente, mostrando a nova peça colocada. No final da colocação das duas peças e consequentemente no final do turno, dá-se a mudança de jogador.



Quando um jogador entra em Check, é mostrada uma notificação no topo do tabuleiro, dando oportunidade de o jogador seguinte poder fazer uma jogada a contrariar a vitória do adversário.



Caso o estado de jogo ainda se mantenha, ao ponto de o jogador continuar em Check no seu turno, é declarada a vitória e término do jogo.

```
**************CHECKMATE**************
 ***Black player is the winner***

   * * * C A L L A N I S H * * *

     A   B   C   D   E   F   G   H   I
   ---------------------------------
9  | X | O |   |   |   |   |   |   |  9
   ---------------------------------
8  | X | O | O | O |   |   |   |   |  8
   ---------------------------------
7  | O |   |   |   |   |   |   |   |  7
   ---------------------------------
6  | X |   |   |   |   |   |   |   |  6
   ---------------------------------
5  | X | O |   |   |   |   |   |   |  5
   ---------------------------------
4  | X |   |   |   |   |   |   |   |  4
   ---------------------------------
3  |   |   |   |   |   |   |   |   |  3
   ---------------------------------
2  |   |   |   |   |   |   |   |   |  2
   ---------------------------------
1  |   |   |   |   |   |   |   |   |  1
   ---------------------------------
     A   B   C   D   E   F   G   H   I
```

5 - Conclusões

Com este trabalho, acreditamos que aprofundamos o nosso conhecimento na linguagem Prolog e que cumprimos a maior parte dos objetivos propostos.
A grande falha do trabalho foi a não realização dos restantes modos de jogo (Humano vs Computador / Computador vs Computador), apesar da jogabilidade Humano vs Humano estar completa, com todas as verificações necessárias para realizar um jogo corretamente.

O balanço final é positivo, apesar da não implementação da inteligência artificial.

Anexo 1: Código em Prolog - callanish.pl

```prolog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     CALLANISH                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%
%Grupo 2                         %
%Turma 5                         %
%----------------------%
%Pedro Faria          ei11167%
%Rui Botto            ei11021%
%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%
%Static New Boards%
%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%
%Empty Initial Board%
%%%%%%%%%%%%%%%%%%%%%

board([[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]]).

%%%%%%%%%%%%%
%Test Boards%
%%%%%%%%%%%%%

board2([[0,0,0,0,0,0,0,0,0],
        [1,2,2,2,0,0,0,0,0],
        [2,0,0,0,0,0,0,0,0],
        [1,0,0,0,0,0,0,0,0],
```

```
        [1,0,0,0,0,0,0,0,0],
        [1,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0]]).

board3([[0,0,0,0,0,0,0,0,0],
        [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]]).

board4([[0,0,0,0,0,0,0,0,0],
        [2,0,0,0,0,0,0,0,0],
        [2,0,0,0,0,0,0,0,0],
        [2,0,0,0,0,0,0,0,0],
        [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]]).

%%%%%%%%%%%%%%
%Print Boards%
%%%%%%%%%%%%%%

printTitle:- nl, write('            * * * C A L L A N I S H * * *'),nl,nl.
draw_empty_line:-write('      ----------------------------------').
draw_board(B):- printTitle, nl,write('          A     B     C     D     E     F     G     H     I'), nl,
draw_empty_line, nl, draw_lines(9,B), write('         A  B  C  D  E  F  G  H  I'), nl.
draw_lines(_,[]).
draw_lines(N,[H|T]):-  write(N),  write('       '),  draw_line(H),  write('|       '),  write(N),  nl,
draw_empty_line, nl, N2 is N-1, draw_lines(N2,T).
draw_line([]).
draw_line([He|Ta]):- draw_board_element(He), draw_line(Ta).
draw_board_element(0):-write('|   '). %empty cell
draw_board_element(1):-write('| X '). %black piece
```

```
draw_board_element(2):-write('| O '). %white piece
draw_board_element(3):-write('| o '). %white stack
draw_board_element(4):-write('| x '). %black stack

%%%%%%%%%%%%
%Game Logic%
%%%%%%%%%%%%

%%%%%%%%%%%%
%Test games%
%%%%%%%%%%%%

callanish2:- board2(Board),
             play(1, Board, white, 0).

callanish3:- board3(Board),
             play(1, Board, white, 0).

callanish4:- board4(Board),
             play(1, Board, white, 0).

%%%%%%%%%%
%New Game%
%%%%%%%%%%

callanish:- board(Board),
            play(1, Board, white, 0).

%%%%%%%%%%%%
%Game cicle%
%%%%%%%%%%%%

play(Turn, Board, Player, 1):-      nl,write('
**********CHECKMATE**********'),
                                                                  nl,write('
***White player is the winner***'),nl,nl,

draw_board(Board),nl.

play(Turn, Board, Player, 2):-      nl,write('
**********CHECK**********'),
```

```
                                                                    nl,write('
***White player is on check  ***'),nl,nl,

draw_board(Board),nl,
                                                                    write('Turn:  '),
write(Turn),nl,nl,

movement(Turn, Board, Player, NewBoard),

check_end_game(Turn, NewBoard, Player, 2, End2),

change_turn(Turn, NextTurn, Player, NextPlayer),

play(NextTurn, NewBoard, NextPlayer, End2).


play(Turn, Board, Player, 3):-          nl,write('
**********CHECKMATE**********'),
                                                                     nl,write('
***Black player is the winner***'),nl,nl,

draw_board(Board),nl.

play(Turn, Board, Player, 4):-          nl,write('
**********CHECK**********'),
                                                                    nl,write('
***Black player is on check  ***'),nl,nl,

draw_board(Board),nl,
                                                                    write('Turn:  '),
write(Turn),nl,nl,

movement(Turn, Board, Player, NewBoard),

check_end_game(Turn, NewBoard, Player, 4, End2),

change_turn(Turn, NextTurn, Player, NextPlayer),

play(NextTurn, NewBoard, NextPlayer, End2).

play(Turn, Board, Player, End):-
```

```
                                        draw_board(Board),nl,
                                        write('Turn: '), write(Turn),nl,nl,
                                        movement(Turn, Board, Player,
NewBoard),
                                        check_end_game(Turn, NewBoard,
Player, End, End2),
                                        change_turn(Turn, NextTurn, Player,
NextPlayer),
                                        play(NextTurn, NewBoard,
NextPlayer, End2).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%First and second movements%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

movement(1, B, P, NB):-  play_movement(B, P, NB).

movement(2, B, P, NB):-  play_movement(B, P, NB).

play_movement(B, P, NB):- repeat,
                                        ask_for_put(X, Y, P),
                                        evaluate_put(B, P, X, Y),
                                        put_piece(B, P, X, Y, NB),!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%All other movements%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

movement(T, B, P, NB):-  repeat,
                                        pick_piece(B, P, Xi, Yi, NB1),
                                        repeat,
                                        picked_from(Xi, Yi),
                                        call_available_movements(Xi, Yi),
                                        play_movement(NB1, P, Xi, Yi,
NB2),draw_board(NB2),
                                        repeat,
                                        picked_from(Xi, Yi),
                                        call_available_movements(Xi, Yi),
                                        play_movement(NB2, P, Xi, Yi, NB).

pick_piece(B, P, Xi, Yi, NB):-  ask_for_pick(Xi, Yi, P),
                                        evaluate_pick(B, P, Xi, Yi),
```

```
                                                    remove_piece(B, P, Xi, Yi, NB).

play_movement(B, P, Xi, Yi, NB):- ask_for_put(Xf, Yf, P),
                                                    evaluate_put(B, P, Xi, Yi, Xf,
Yf),
                                                    put_piece(B, P, Xf, Yf, NB).

picked_from(Xi, Yi):-          num_to_alpha(Xi, Xalfa),
                                                    nl,write('Picked                    from:
'),write(Xalfa),write(','),write(Yi),nl,nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Available cells to move%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

call_available_movements(Xi, Yi):-          write('Available cells: '),nl,
                                                    available_movements(Xi,  Yi,
1),nl.

available_movements(Xi,  Yi,  1):-  Xf  is  Xi+2,  Yf  is  Yi+1,  Xf>=1,  Xf=<9,  Yf>=1,  Yf=<9,
num_to_alpha(Xf,        Xalfa),write('X:        '),write(Xalfa),write('        |Y:        '),write(Yf),nl,
available_movements(Xi, Yi, 2).
available_movements(Xi, Yi, 1):- available_movements(Xi, Yi, 2).
available_movements(Xi,  Yi,  2):-  Xf  is  Xi+2,  Yf  is  Yi-1,  Xf>=1,  Xf=<9,  Yf>=1,  Yf=<9,
num_to_alpha(Xf,        Xalfa),write('X:        '),write(Xalfa),write('        |Y:        '),write(Yf),nl,
available_movements(Xi, Yi, 3).
available_movements(Xi, Yi, 2):- available_movements(Xi, Yi, 3).
available_movements(Xi,  Yi,  3):-  Xf  is  Xi-2,  Yf  is  Yi+1,  Xf>=1,  Xf=<9,  Yf>=1,  Yf=<9,
num_to_alpha(Xf,        Xalfa),write('X:        '),write(Xalfa),write('        |Y:        '),write(Yf),nl,
available_movements(Xi, Yi, 4).
available_movements(Xi, Yi, 3):- available_movements(Xi, Yi, 4).
available_movements(Xi,  Yi,  4):-  Xf  is  Xi-2,  Yf  is  Yi-1,  Xf>=1,  Xf=<9,  Yf>=1,  Yf=<9,
num_to_alpha(Xf,        Xalfa),write('X:        '),write(Xalfa),write('        |Y:        '),write(Yf),nl,
available_movements(Xi, Yi, 5).
available_movements(Xi, Yi, 4):- available_movements(Xi, Yi, 5).
available_movements(Xi,  Yi,  5):-  Xf  is  Xi+1,  Yf  is  Yi+2,  Xf>=1,  Xf=<9,  Yf>=1,  Yf=<9,
num_to_alpha(Xf,        Xalfa),write('X:        '),write(Xalfa),write('        |Y:        '),write(Yf),nl,
available_movements(Xi, Yi, 6).
available_movements(Xi, Yi, 5):- available_movements(Xi, Yi, 6).
```

```
available_movements(Xi, Yi, 6):- Xf is Xi+1, Yf is Yi-2, Xf>=1, Xf=<9, Yf>=1, Yf=<9, num_to_alpha(Xf, Xalfa),write('X: '),write(Xalfa),write(' |Y: '),write(Yf),nl,
available_movements(Xi, Yi, 7).
available_movements(Xi, Yi, 6):- available_movements(Xi, Yi, 7).
available_movements(Xi, Yi, 7):- Xf is Xi-1, Yf is Yi+2, Xf>=1, Xf=<9, Yf>=1, Yf=<9, num_to_alpha(Xf, Xalfa),write('X: '),write(Xalfa),write(' |Y: '),write(Yf),nl,
available_movements(Xi, Yi, 8).
available_movements(Xi, Yi, 7):- available_movements(Xi, Yi, 8).
available_movements(Xi, Yi, 8):- Xf is Xi-1, Yf is Yi-2, Xf>=1, Xf=<9, Yf>=1, Yf=<9, num_to_alpha(Xf, Xalfa),write('X: '),write(Xalfa),write(' |Y: '),write(Yf),nl.
available_movements(Xi, Yi, 8).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Transform numerical to alphabetical%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

num_to_alpha(X, Xalfa):- X1 is X+64, char_code(Xalfa, X1).

%%%%%%%%%
%Inputs%
%%%%%%%%%

ask_for_put(X, Y, P):-      repeat,
                                    write('Player '), write(P),
                                    write(' select a cell to put piece.'),nl,
                                    write('Column [A-I].'),nl,
                                    read_X(X),
                                    write('Row [1-9].'),nl,
                                    read_Y(Y),!.


ask_for_pick(X, Y, P):-     repeat,
                                    write('Player '), write(P),
                                    write(' select a cell to remove piece.'),nl,
                                    write('Column [A-I].'),nl,
                                    read_X(X),
                                    write('Row [1-9].'),nl,
                                    read_Y(Y),!.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check correct positions%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%

read_X(X2):- repeat,get_code(X),get_code(_), conv_col(X,X2),!.
%only accepts one valid input
read_Y(Y2):- repeat,get_code(Y),get_code(_), conv_row(Y,Y2),!.

conv_col(X,X2):- uppercase(X), X2 is X-64.
        %converts letters into numbers
conv_col(X,X2):- lowercase(X), X2 is X-96.
conv_col(X,X2):- write('That column doesnt exist, choose again.'),nl,fail.

conv_row(Y, Y2):- num(Y), Y2 is Y-48.
conv_row(Y, Y2):- write('That row doesnt exist, choose again.'),nl,fail.

uppercase(X):- X>=65, X=<73.                % accepts upper and lowercase letters
lowercase(X):- X>=97, X=<105.
num(Y):- Y>=49, Y=<57.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Outputs - board refreshing%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Putting a piece on board%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

put_piece(B, P, X, Y, NB):-  refresh_put(B, P, X, Y, NB).

refresh_put([BoardH|BoardT], P, X, 9, [NboardH|BoardT]):-        refresh_put2(BoardH,  P,  X,
NboardH).

refresh_put([BoardH|BoardT], P, X, Y, [BoardH|NboardT]):-        Y1          is          Y+1,
refresh_put(BoardT, P, X, Y1, NboardT).

refresh_put2([0|BoardT], white, 1, [2|BoardT]).
refresh_put2([1|BoardT], white, 1, [3|BoardT]).
refresh_put2([0|BoardT], black, 1, [1|BoardT]).
refresh_put2([2|BoardT], black, 1, [4|BoardT]).

refresh_put2([BoardH|BoardT], P, X, [BoardH|NboardT]):- X1 is X-1, refresh_put2(BoardT, P,
X1, NboardT).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Removing a piece from board%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

remove_piece(B, P, Xi, Yi, NB):- refresh_remove(B, P, Xi, Yi, NB).

refresh_remove([BoardH|BoardT], P, X, 9, [NboardH|BoardT]):- refresh_remove2(BoardH, P,
X, NboardH).

refresh_remove([BoardH|BoardT], P, X, Y, [BoardH|NboardT]):- Y1 is Y+1,
refresh_remove(BoardT, P, X, Y1, NboardT).

refresh_remove2([2|BoardT], white, 1, [0|BoardT]).
refresh_remove2([3|BoardT], white, 1, [1|BoardT]).
refresh_remove2([1|BoardT], black, 1, [0|BoardT]).
refresh_remove2([4|BoardT], black, 1, [2|BoardT]).

refresh_remove2([BoardH|BoardT], P, X, [BoardH|NboardT]):- X1 is X-1,
refresh_remove2(BoardT, P, X1, NboardT).
%%%%%%%%%%%%%%%
%Evaluations%
%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check if player is puting in valid position%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%
%For the 1st move%
%%%%%%%%%%%%%%%%%%%%%

evaluate_put(B, P, X, Y):-     get_cell(B, X, Y, Piece),
                               evaluate_put_piece_player(Piece, P, 1).

%%%%%%%%%%%%%%%%%%%%%%%%
%For all other moves%
%%%%%%%%%%%%%%%%%%%%%%%%

evaluate_put(B, P, Xi, Yi, Xf, Yf):- get_cell(B, Xf, Yf, Piece),
```

```
evaluate_put_piece_player(Piece, P, 0),

evaluate_horse_movement(Xi, Yi, Xf, Yf).


evaluate_put_piece_player(Piece, Player, First):- Piece = 0.
evaluate_put_piece_player(Piece, white, First):- Piece = 1.
evaluate_put_piece_player(Piece, black, First):-    (

        Piece = 2, First = 0;

        write('1st movement of black cant be put on white piece'),nl,fail

). %1st movement of black cant be put on white piece

evaluate_put_piece_player(Piece, black, First):- Piece = 1, write('Black player, please put on white piece or blank cell.'), nl, fail.
evaluate_put_piece_player(Piece, black, First):- Piece = 3, write('Black player, please put on white piece or blank cell.'), nl, fail.
evaluate_put_piece_player(Piece, black, First):- Piece = 4, write('Black player, please put on white piece or blank cell.'), nl, fail.

evaluate_put_piece_player(Piece, white, First):- Piece = 2, write('White player, please put on black piece or blank cell.'), nl, fail.
evaluate_put_piece_player(Piece, white, First):- Piece = 3, write('White player, please put on black piece or blank cell.'), nl, fail.
evaluate_put_piece_player(Piece, white, First):- Piece = 4, write('White player, please put on black piece or blank cell.'), nl, fail.

evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi+2, Y1 is Yi+1, Xf=X1, Yf=Y1.
evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi+2, Y1 is Yi-1, Xf=X1, Yf=Y1.

evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi-2, Y1 is Yi+1, Xf=X1, Yf=Y1.
evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi-2, Y1 is Yi-1, Xf=X1, Yf=Y1.

evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi+1, Y1 is Yi+2, Xf=X1, Yf=Y1.
evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi+1, Y1 is Yi-2, Xf=X1, Yf=Y1.

evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi-1, Y1 is Yi+2, Xf=X1, Yf=Y1.
evaluate_horse_movement(Xi, Yi, Xf, Yf):- X1 is Xi-1, Y1 is Yi-2, Xf=X1, Yf=Y1.
```

```
evaluate_horse_movement(Xi, Yi, Xf, Yf):- write('Please choose a valid cell (chess horse movement).'),nl,fail.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check if player is picking his piece%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

evaluate_pick(B, P, Xi, Yi):- get_cell(B, Xi, Yi, Piece),
                                    evaluate_pick_piece_player(Piece, P).

evaluate_pick_piece_player(Piece, white):- Piece = 2.
evaluate_pick_piece_player(Piece, white):- Piece = 3.
evaluate_pick_piece_player(Piece, black):- Piece = 1.
evaluate_pick_piece_player(Piece, black):- Piece = 4.

evaluate_pick_piece_player(Piece, Player):- Piece = 0, write('Cant pick empty cell.'), nl, fail.
evaluate_pick_piece_player(Piece, black):- Piece = 2, write('Black player, please pick black piece.'), nl, fail.
evaluate_pick_piece_player(Piece, black):- Piece = 3, write('Black player, please pick black piece.'), nl, fail.
evaluate_pick_piece_player(Piece, white):- Piece = 1, write('White player, please pick white piece.'), nl, fail.
evaluate_pick_piece_player(Piece, white):- Piece = 4, write('White player, please pick white piece.'), nl, fail.

%%%%%%%%%%%%%%%%%%%%%%%%
%Get cell contents%
%%%%%%%%%%%%%%%%%%%%%%%%

get_cell([BoardH|BoardT], X, 9, Piece):-
                                    get_cell2(BoardH, X, 9, Piece).

get_cell([BoardH|BoardT], X, Y, Piece):- Y1 is Y+1,
                                    get_cell(BoardT, X, Y1, Piece).

get_cell2([Piece|BoardT], 1, 9, Piece).
```

```
get_cell2([BoardH|BoardT], X, 9, Piece):- X1 is X-1,

get_cell2(BoardT, X1, 9, Piece).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%Change turn and player%
%%%%%%%%%%%%%%%%%%%%%%%%%%

change_turn(Turn, NextTurn, Player, black):- Player = white, NextTurn is Turn+1.
change_turn(Turn, NextTurn, Player, white):- Player = black, NextTurn is Turn+1.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check end game conditions%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%
%If game is on check%
%%%%%%%%%%%%%%%%%%%%%%

check_end_game(Turn, Board, Player, 2, End2):-   (

fiveAlign(Board, white), End2 is 1;

End2 is 0
                                                                    ).

check_end_game(Turn, Board, Player, 4, End2):-   (

fiveAlign(Board, black), End2 is 3;

End2 is 0
                                                                    ).

%%%%%%%%%%%%%%%%%%%%%%%%%%
%Game ends on checkmate%
%%%%%%%%%%%%%%%%%%%%%%%%%%

check_end_game(Turn, Board, Player, End, End2):-          Player == white,

(
```

```
        fiveAlign(Board, Player), End2 is 2;

        End2 is End

).

check_end_game(Turn, Board, Player, End, End2):-        Player == black,

(

        fiveAlign(Board, Player), End2 is 4;

        End2 is End

).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
%Confirm if there is a sequence of 5 pieces to end the game%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%

fiveAlign(Board, Player):-
                                        fiveInARow(9, Board, Player);
                                        fiveInACol(9, Board ,Player).
%%%%%%%%%%%
%In a row%
%%%%%%%%%%%

fiveInARow(Row, Board, Player):-
                                        Row >= 0,
                                        Row1 is Row-1,
                                    (
                                        validateFiveInARow(Row, Board, Player);
                                        fiveInARow(Row1, Board, Player)
                                    ).

validateFiveInARow(Row, Col, Board, Acc, Player):-
                                    Col >= 0,
                                    Col1 is Col-1,
```

```
                                        get_cell(Board, Col, Row, Piece),
                                        (
                                                pieceIsOwnedByPlayer(Piece, Player) ->
                                                        Acc1 is Acc + 1;
                                                Acc1 is Acc
                                        ),
                                        (
                                                Acc1 >= 5;
                                                validateFiveInARow(Row,    Col1,    Board,
Acc1, Player)
                                        ).

validateFiveInARow(Row, Board, Player):-
                                        validateFiveInARow(Row, 9, Board, 0, Player).



%%%%%%%%%%%%%
%In a column%
%%%%%%%%%%%%%

fiveInACol(Col, Board, Player):-
                                                Col >= 0,
                                                Col1 is Col-1,
                                        (
                                                validateFiveInACol(Col, Board, Player);
                                                fiveInACol(Col1, Board, Player)
                                        ).

validateFiveInACol(Col, Board, Player):-
                                        validateFiveInACol(9, Col, Board, 0, Player).

validateFiveInACol(Row, Col, Board, Acc, Player):-
                                        Row >= 0,
                                        Row1 is Row-1,
                                        get_cell(Board, Col, Row, Piece),
                                        (
                                                pieceIsOwnedByPlayer(Piece, Player)->
                                                        Acc1 is Acc + 1;
                                                Acc1 is Acc
                                        ),
```

```
                                        (
                                             Acc1 >= 5;
                                             validateFiveInACol(Row1,    Col,    Board,
Acc1, Player)
                                        ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Check if the piece on selected cell, belongs to the current player%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pieceIsOwnedByPlayer(Piece, Player):-
     ((Piece =:= 1; Piece =:= 4), Player == black);
     ((Piece =:= 2; Piece =:= 3), Player == white).
```