U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

ZURERO

Programação em Lógica 2018/2019 – Turma 3

Diogo Filipe da Silva Yaguas – up201606165@fe.up.pt

Eduardo Luís Pinheiro da Silva – up201603135@fe.up.pt

Mestrado Integrado em Engenharia Informática e Computação

ÍNDICE

1. Introdução 3
2. O jogo Zurero 4
   2.1. Regras 5
3. Lógica do jogo 7
   3.1. Representação do estado do jogo 7
   3.2. Visualização do tabuleiro 8
   3.3. Lista de jogadas válidas 10
   3.4. Execução das jogadas 11
   3.5. Final do jogo 12
   3.6. Avaliação do tabuleiro 13
   3.7. Jogada do computador 14
4. Conclusões 15
Bibliografia 15

1. INTRODUÇÃO

Como objetivo principal deste trabalho, foi-nos proposta a realização de um jogo de tabuleiro para dois jogadores em linguagem *Prolog*. As características deste incluem um tabuleiro, peças, as regras de movimentação destas e as condições de vitória/derrota (ou empate). Este jogo deve permitir três modos de utilização distintos:

- Humano vs Humano (PvP);
- Humano vs Computador (PvC);
- Computador vs Computador (CvC);

Nos casos que envolvem o computador, este deve permitir a escolha entre pelo menos dois níveis de dificuldade distintas. Para poder interagir com o sistema, deve ser criada uma interface adequada com o utilizador, em modo de texto.

2. O JOGO ZURERO

Zurero é um jogo de tabuleiro abstrato criado em 2009 por Jordan Goldstein, inspirado no *Gomoku*, outro jogo da mesma categoria.

O material necessário para o jogo é um tabuleiro, estilo *Go*, quadrado com 19x19 espaços, sendo possível jogar com outros tamanhos, e uma grande quantidade de peças brancas e pretas.

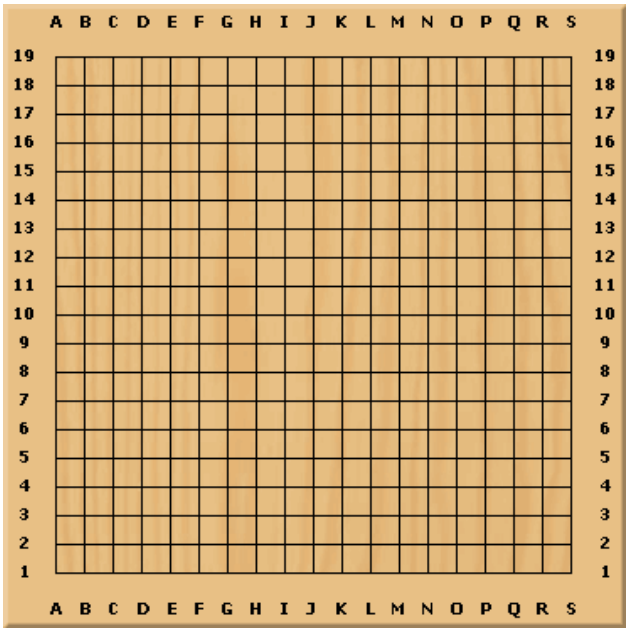

Figura 1 - Tabuleiro do jogo

A condição vencedora é um dos jogadores obter 5 das suas peças em linha, seja esta horizontal, vertical ou diagonal. Esta condição foi deliberadamente selecionada pois é um dos conceitos mais utilizados em jogos clássicos e contemporâneos.

2.1. REGRAS

Inicialmente, o tabuleiro está vazio. Cada jogador tem uma cor associada. O jogador das peças pretas começa por colocar uma peça no centro do tabuleiro (linha 10 e coluna J).

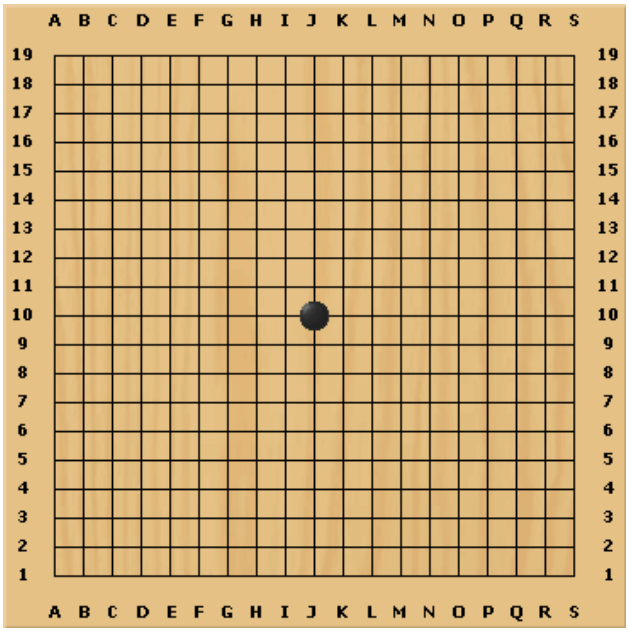

Figura 2.1 - Tabuleiro do jogo com a peça inicial no centro

Depois desse movimento, os jogadores, alternadamente, deslizam uma peça de qualquer borda do tabuleiro ao longo das linhas do tabuleiro. Uma peça desliza até atingir outra peça que já está no tabuleiro. **Não é permitido deslizar uma peça ao longo de uma linha sem que outras peças estejam no caminho.**

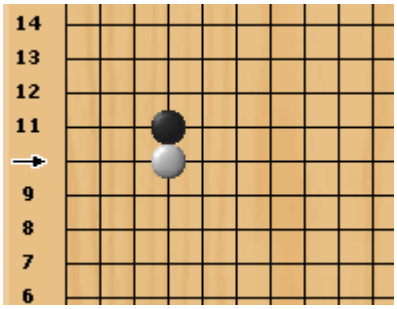

Figura 2.2 – Após as duas jogadas iniciais

Se a peça que já está no tabuleiro é atingida pela peça deslizada e não possuir nada na direção oposta, a peça é "empurrada" para trás um espaço e a peça deslizada move-se para o espaço previamente ocupado pela outra peça.

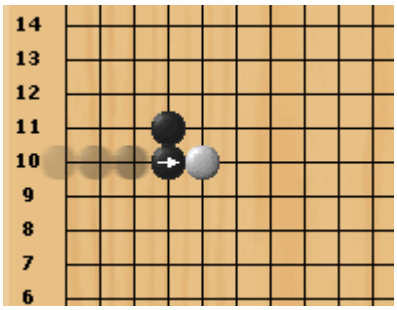

Figura 2.3 - Peça *deslizada* a empurrar a peça branca

Se, ao empurrar uma peça adversária, um jogador colocar 5 peças do adversário em linha, o jogador perde o jogo, a menos que o jogador forme uma combinação de 5 peças com as suas próprias peças no mesmo movimento.

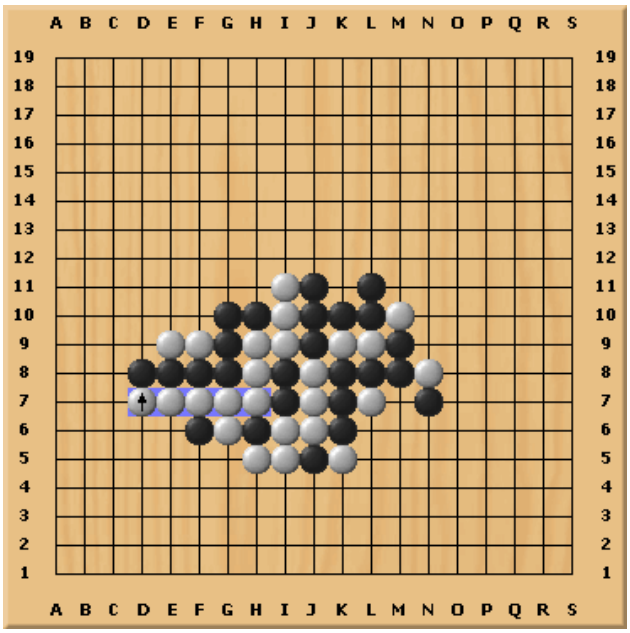

Figura 2.4 – Exemplo de fim do jogo em que o jogador com as peças brancas ganha

3. Lógica do jogo

3.1. Representação do estado do jogo

A representação interna do tabuleiro é feita através de uma lista de listas de tamanho fixo, sendo os espaços vazios identificados por ‘*empty*’, as peças brancas com ‘w’ e as peças pretas com ‘b’. As figuras seguintes representam uma exemplificação do estado interno em situações distintas.

```
table([
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,'b',empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]]).
```

Figura 3.1.1 – Representação interna da situação inicial

```
table([
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,'b',empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,'b','b',w,'b',empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,'b','w',w,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,w,w,'b',empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]]).
```

Figura 3.1.2 – Representação interna da situação intermédia

```
table([
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,'b',empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,'b',empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,'b',empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,'b','b',w,'b',empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,'b','w',w,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,'w','w','w','b',empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]]).
```

Figura 3.1.3 – Representação interna da situação final

3.2. Visualização do tabuleiro

Para a visualização na consola são impressas duas linhas nos extremos verticais do tabuleiro que indicam as colunas além de outras duas colunas de ambos os lados que indicam o número da linha. Os espaços vazios são representados por um ponto '.' enquanto que as peças brancas e pretas são representadas por um carater especial: '○' e '●', respetivamente. As figuras seguintes mostram o tabuleiro impresso na consola em vários momentos do jogo.

```
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
01|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|01
02|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|02
03|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|03
04|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|04
05|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|05
06|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|06
07|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|07
08|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|08
09|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|09
10|.|.|.|.|.|.|.|.|.|●|.|.|.|.|.|.|.|.|.|10
11|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|11
12|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|12
13|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|13
14|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|14
15|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|15
16|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|16
17|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|17
18|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|18
19|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|19
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
```

```
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
01|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|01
02|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|02
03|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|03
04|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|04
05|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|05
06|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|06
07|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|07
08|.|.|.|.|.|.|.|.|●|.|.|.|.|.|.|.|.|.|.|08
09|.|.|.|.|.|.|.|●|●|○|●|.|.|.|.|.|.|.|.|09
10|.|.|.|.|.|.|.|.|●|●|○|.|.|.|.|.|.|.|.|10
11|.|.|.|.|.|.|.|.|○|○|●|.|.|.|.|.|.|.|.|11
12|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|12
13|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|13
14|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|14
15|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|15
16|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|16
17|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|17
18|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|18
19|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|19
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
```

```
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
01|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|01
02|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|02
03|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|03
04|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|04
05|.|.|.|.|.|.|.|.|○|.|.|.|.|.|.|.|.|.|.|05
06|.|.|.|.|.|.|.|.|●|.|.|.|.|.|.|.|.|.|.|06
07|.|.|.|.|.|.|.|.|●|.|.|.|.|.|.|.|.|.|.|07
08|.|.|.|.|.|.|.|.|●|.|.|.|.|.|.|.|.|.|.|08
09|.|.|.|.|.|.|.|●|●|○|●|.|.|.|.|.|.|.|.|09
10|.|.|.|.|.|.|.|.|●|●|○|.|.|.|.|.|.|.|.|10
11|.|.|.|.|.|.|.|○|○|○|●|.|.|.|.|.|.|.|.|11
12|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|12
13|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|13
14|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|14
15|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|15
16|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|16
17|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|17
18|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|18
19|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|.|19
   A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|
```

Figura 3.2.1 – Tabuleiro impresso na consola na situação inicial, numa situação intermédia e numa situação final, respetivamente, da esquerda para a direita.

O predicado responsável pela impressão denomina-se ***display_game(+Board, +Player)***, recebendo como argumentos o tabuleiro e o próximo jogador. Esta irá chamar os predicados ***show_player(+Player)***, que mostra no ecrã qual o próximo jogador**, e** ***print_board(+Board)***, que irá percorrer o tabuleiro e imprimir as peças nele contido além das informações adicionais de cada lado. O predicado principal para a impressão do tabuleiro, ***view_tab(+Board, +N)***, é chamada dentro desta.

Nota: Aconselha-se o uso da font "Consolas", regular, na consola para uma melhor visualização do tabuleiro.

```
% Prints the board as well as its headers.
print_board(Board) :-
    print_table_header,
    view_tab(Board, 1),
    print_table_header.

% Prints a list of lists (board) as well as the line numbers on each side.
view_tab([], 20) :-
    !.

view_tab([H|T], N) :-
    print_format_number(N),
    write('|'),
    print_list(H),
    print_format_number(N),
    nl,
    Next is (N + 1),
    view_tab(T, Next).

% Prints a list, replacing certain characters with custom unicode ones.
print_list([]) :-
    !.

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

Figura 3.2.2 – Predicados responsáveis pela impressão do tabuleiro na consola

3.3. LISTA DE JOGADAS VÁLIDAS

De forma a obter uma lista de jogadas válidas é chamada o predicado ***valid_moves(+Board, +Player, -ListOfMoves)***. Este irá por sua vez chamar o predicado ***get_moves(+Board, +Moves, +Index, +Symbol, +Player, -ListOfMoves)***. Este predicado recebe como argumentos o tabuleiro, uma lista auxiliar (para efeitos totalmente corretos esta deve estar vazia), um índice para a linha/coluna (na primeira chamada este deve ser 1), o símbolo da jogada (‘L’ ou ‘C’ se for jogada na linha ou coluna, respetivamente) e o símbolo do jogador ativo.

Uma jogada é adicionada à lista a retornar caso haja pelo menos uma peça na linha/coluna indicada e caso o primeiro e segundo elementos desta (a partir da direção indicada) não estejam simultaneamente ocupados. Esta irá retornar no último argumento a lista de jogadas possíveis numa determinada direção. Deste modo, no predicado ***valid_moves***, o predicado ***get_moves*** irá ser chamado duas vezes, uma com o tabuleiro na orientação normal e outra com ele transposto. As listas obtidas em ambas as chamadas são depois concatenadas, ficando assim com uma lista de todas as jogadas possíveis.

```
% Goes through the board a gets a list of the valid moves
get_moves([ ], Moves, _, _, _, Moves).

get_moves([Line | RestOfBoard], Moves, Index, Symbol, Player, ListOfMoves) :-
        has_pieces(Line), % Make sure the line has pieces
        % Used for the last element special case
        check_last_element_and_add_play(Line, Moves, Index, Symbol, Player, NewMoves),
        % Used for the first element special case
        check_first_element_and_add_play(Line, NewMoves, Index, Symbol, Player, NewMoves2),
        NextIndex is (Index + 1),
        get_moves(RestOfBoard, NewMoves2, NextIndex, Symbol, Player, ListOfMoves).

get_moves([_ | RestOfBoard], Moves, Index, Symbol, Player, ListOfMoves) :-
    NextIndex is (Index + 1),
    get_moves(RestOfBoard, Moves, NextIndex, Symbol, Player, ListOfMoves).
```

Figura 3.3 – Predicado get_moves(+Board, +Moves, +Index, + Symbol, +Player, -ListOfMoves)

3.4. EXECUÇÃO DAS JOGADAS

Para a validação e execução de jogadas é usado o predicado ***move(+Move, +MoveList, +Board, -NewBoard)***. Este predicado recebe como argumentos a jogada a validar e executar, a lista de jogadas possíveis naquele instante e o tabuleiro do jogo. Ela irá retornar um novo tabuleiro já com a jogada executada (se esta for válido, caso contrário irá falhar).

Para efeitos de validação, o predicado verifica se a jogada recebida faz parte da lista das jogadas possíveis. Em caso afirmativo, ela procede em chamar o predicado ***check_play_type(+Symbol, +RestOfPlay, +Board, -NewBoard)***, que analisa o tipo de jogada (linha ou coluna), e que irá por sua vez chamar uma de duas funções para analisar a direção da jogada e executá-la, ou numa linha (***check_line_play_direction(+ Board, +Play, +LineNumber, -NewBoard)***) ou numa coluna (***check_column_play_direction(+Board, +Play, -NewBoard)***).

Estas irão então executar algumas operações (ou não) de acordo com a direção escolhida e com a jogada em si antes de finalmente chamarem os predicados respetivos que irão executar a jogada. Estes são os predicados ***play_line(+Board, +Player, -NewBoard)*** e ***play_column(+Board, +Column, +Player, -NewBoard)***. Embora estes funcionem de maneiras distintas, o princípio é o mesmo: eles irão percorrer a linha/coluna respetiva (na direção indicada) até encontrarem uma peça. Se esta não possuir nenhuma peça na casa ao lado, essa casa fica a ser ocupada pela peça encontrada e o lugar desta pela peça do jogador. Caso contrário a peça do jogador fica a ocupar a casa anterior à peça encontrada.

```
% Analyses the next two pieces (if they exist) in the line and decides what to do
play_line_decide([Head | RestOfLine], Player, NextPiece, [NewHead | Remainder]) :-
    length(RestOfLine, L), L >= 2, !,
    (
        nth1(2, RestOfLine, SecondPiece),
        (
            % If there is another piece after the one we found, just add the player's piece to the current slot in the line
            (SecondPiece \= empty, NewHead = Player, Remainder = RestOfLine) ; true,

            % Else move the piece we found one spot and place the player's piece in its place
            replace_element(2, RestOfLine, NextPiece, NewRestOfLine),
            replace_element(1, NewRestOfLine, Player, Remainder),
            NewHead = Head
        )
    ).
```

Figura 3.4.1 – Predicado play_line_decide(+Board, +Player, +NextPiece, -NewBoard), chamado dentro do predicado play_line.

```
% Analyses the next two pieces (if they exist) in the column and decides what to do.
play_column_decide([Line | RestOfBoard], Column, Player, NextLine, Piece, [Head | Remainder]) :-
    length(RestOfBoard, L), L >= 2, !, % If there are at least two more pieces/lines after the current one
    (
        nth1(2, RestOfBoard, SecondLine),
        get_piece_in_column(Column, SecondLine, NextPiece),
        (
            % If there is another piece after the one we found, just add the player's piece to the current slot in the
            % column
            (NextPiece \= empty, replace_element(Column, Line, Player, Head), Remainder = RestOfBoard); true,

            % Else move the piece we found one spot and place the player's piece in its place
            replace_element(Column, SecondLine, Piece, NewBottomLine),
            replace_element(2, RestOfBoard, NewBottomLine, NewRestOfBoard),
            replace_element(Column, NextLine, Player, NewLine),
            replace_element(1, NewRestOfBoard, NewLine, Remainder),
            Head = Line
        )
    ).
```

Figura 3.4.2 – Predicado play_column_decide(+Board, +Column, +Player, +NextLine, +Piece, -NewBoard), chamado dentro do predicado play_column.

3.5. FINAL DO JOGO

Na verificação de final do jogo é usada o predicado ***game_over(+Board, -Winner, +PieceIndex)***. Esta recebe como argumentos o tabuleiro e o número da coluna da casa a ser analisada, devolvendo o símbolo correspondente ao jogador que ganhou (caso alguém tenha de facto ganhe). No predicado, o tabuleiro é percorrido de linha a linha, sendo cada uma destas percorrida coluna a coluna. Se a casa a ser analisada não estiver vazia, inicia-se a verificação das 5 peças em linha, sendo primeiro verificada a linha atual (ou seja, na horizontal, para a direita), de seguida a verificação da coluna (vertical, para baixo) e por fim verificam-se as diagonais esquerda e direita para baixo. Se nenhuma destas validar, passa-se à casa seguinte.

```
% Goes trough the board, verifying if there are 5 pieces in a row of the same player.
game_over([[] | []], _) :-
    false.

% Checks for five equal pieces in a row
game_over([[Piece | RestOfLine] | RestOfBoard], Winner, PieceIndex) :-
    RestOfLine = [], game_over(RestOfBoard, Winner, 0);
    Piece \= empty, % Only analyses if the current slot is occupied by a player's piece
    % Horizontal Search
    check_game_over_horizontal(RestOfLine, Piece, 1), Winner = Piece;
    % Vertical Search
    check_game_over_vertical(RestOfBoard, PieceIndex, Piece, 1), Winner = Piece;
    % Left Diagonal Search
    PreviousPieceIndex is PieceIndex - 1,
    check_game_over_diagonal_left(RestOfBoard, PreviousPieceIndex, Piece, 1), Winner = Piece;
    %Right Diagonal Search
    NextPieceIndex is PieceIndex + 1,
    check_game_over_diagonal_right(RestOfBoard, NextPieceIndex, Piece, 1), Winner = Piece.

game_over([[_ | RestOfLine] | RestOfBoard], Winner, PieceIndex) :-
    NextIndex is (PieceIndex + 1),
    append([RestOfLine], RestOfBoard, CutBoard),
    game_over(CutBoard, Winner, NextIndex).
```

Figura 3.5 – Predicado game_over(+Board, -Winner, +PieceIndex)

3.6. AVALIAÇÃO DO TABULEIRO

Na avaliação do estado do jogo é usada o predicado ***value(+Board, +Player, -Value).*** Este recebe como argumentos o tabuleiro, o jogador atual, devolvendo a avaliação.

```
% Rate the game state
value(Board, Player, Value) :-
    switch_players(Player, Opponent),
    count_rows(Board, Player, 0, [], NumberPlayer),
    count_rows(Board, Opponent, 0, [], NumberOpponent),
    Value is NumberPlayer - NumberOpponent.
```

Figura 3.6.1 – Predicado value(+Board, +Player, -Value)

No predicado, utiliza-se dois predicados auxiliares. Nestes, o tabuleiro é percorrido e contado o número de peças que o jogador tem em linha. Todos os valores são guardados e no fim, é devolvido o máximo número de peças em linha. O mesmo é feito para o adversário. Por último, é feito uma subtração do máximo de peças em linha do adversário ao máximo de peças em linha do jogador atual. Este valor calculado corresponde ao *Value*, avaliação do estado do jogo. Para valores negativos, significa que o jogador atual está em desvantagem para com o adversário, visto que tem menos peças em linha. Para valores positivos, o jogador está em vantagem para com o adversário.

```
% Counts the number of consecutive pieces
count_rows([[] | []], _, _, Rows, NumberPlayer) :-
    Rows = [], NumberPlayer = 0;
    max_member(Number, Rows),
    NumberPlayer is Number.

count_rows([[Piece | RestOfLine] | RestOfBoard], Player, Index, Rows, NumberPlayer) :-
    RestOfLine = [], count_rows(RestOfBoard, Player, 0, Rows, NumberPlayer);
    Piece = Player,
    (
        count_horizontal(RestOfLine, Piece, 1, CounterHorizontal), append([CounterHorizontal], Rows, Rows1),
        count_vertical(RestOfBoard, Piece, Index, 1, CounterVertical), append([CounterVertical], Rows1, Rows2),
        count_diagonal_right(RestOfBoard, Piece, Index, 1, CounterDiagonalRight), append([CounterDiagonalRight], Rows2, Rows3),
        count_diagonal_left(RestOfBoard, Piece, Index, 1, CounterDiagonalLeft), append([CounterDiagonalLeft], Rows3, FinalRow),

        NextIndex is Index + 1,
        append([RestOfLine], RestOfBoard, CutBoard),
        count_rows(CutBoard, Player, NextIndex, FinalRow, NumberPlayer)
    );
    NextIndex is Index + 1,
    append([RestOfLine], RestOfBoard, CutBoard),
    count_rows(CutBoard, Player, NextIndex, Rows, NumberPlayer).
```

Figura 3.6.2 – Predicado count_rows(+Board, +Player +Index, +Rows, -NumberPlayer)

3.7. JOGADA DO COMPUTADOR

De forma a que seja possível jogar contra o computador, é utilizado o predicado ***choose_move(+Board, +Player, -Move, +Level, +ListOfMoves)***. Este recebe como argumentos o tabuleiro, o jogador atual, o nível de dificuldade e a lista de movimentos possíveis, e devolve o movimento ideal. Conforme o nível de dificuldade, o nível de complexidade da escolha do movimento aumenta.

Quando o nível de dificuldade é igual a 1 (Modo Fácil), o computador irá escolher um movimento aleatório, dentro da lista de movimentos possíveis.

```
% Choose a bot's move
choose_move(_, _, Move, 1, ListOfMoves) :-
    random_move(ListOfMoves, Index),
    nth1(Index, ListOfMoves, Move),
    nth1(1, Move, Symbol),
    write_move(Symbol, Move).
```

Figura 3.7.1 – Predicado choose_move(+Board, +Player, -Move, +Level, +ListOfMoves) no Modo Fácil

Por outro lado, quando o nível de dificuldade é igual a 2 (Modo Inteligente), o computador irá aplicar todas as jogadas possíveis, verificar qual a avaliação do estado do jogo, depois desse movimento ter sido aplicado, e guardar o respetivo *Value*. Em último, é selecionado o movimento que proporciona o maior *Value*.

```
choose_move(Board, Player, Move, 2, ListOfMoves) :-
    random_permutation(ListOfMoves, SortedMoves),
    choose_best_play(Board, Player, Move, SortedMoves, [], Index),
    nth0(Index, SortedMoves, Move),
    nth1(1, Move, Symbol),
    write_move(Symbol, Move).
```

Figura 3.7.1 – Predicado choose_move(+Board, +Player, -Move, +Level, +ListOfMoves) no Modo Inteligente

4. CONCLUSÕES

O presente trabalho exigiu um grande empenho por parte de ambos os elementos do grupo, tendo sido uma experiência recompensadora.

O projeto teve como principal objetivo aplicar o conhecimento adquirido nas aulas teóricas e práticas.

Consideramos que o nosso conhecimento de programação em lógica foi amplamente aumentado, tendo sido alcançado tudo o que objetivamos. Ao longo do desenvolvimento do trabalho, foram encontradas dificuldades, nomeadamente, o pensamento recursivo e o melhor forma de desenvolver certos predicados.

Em suma, apesar de *Prolog* se modelar por um paradigma diferente do que estamos habituados, rapidamente nos acostumamos e aprendemos a apreciar as facilidade e dificuldade que este apresenta, tendo culminado num trabalho no qual nos orgulhamos.

BIBLIOGRAFIA

- https://boardgamegeek.com/boardgame/41145/zurero
- http://www.iggamecenter.com/info/en/zurero.html