Programação em Lógica  
IRIS 4

short line

Leonor Martins de Sousa 201705377

Sílvia Jorge Moreira da Rocha 201704684

MIEIC 3 - Turma 1

Prof. Nuno Fonseca

Prof. Daniel Castro Silva

Prof. Rui Camacho

**Índice**

[Introdução 2](#_Toc24835829)

[O Jogo 3](#_Toc24835830)

[História 3](#_Toc24835831)

[Material 3](#_Toc24835832)

[Regras 3](#_Toc24835833)

[Pontuação e Vencedor 4](#_Toc24835834)

[A Lógica 5](#_Toc24835835)

[Representação do Estado do Jogo 5](#_Toc24835836)

[Visualização do Tabuleiro 7](#_Toc24835837)

[Lista de Jogadas Válidas 8](#_Toc24835838)

[Execução de Jogadas 8](#_Toc24835839)

[Final do Jogo 8](#_Toc24835840)

[Avaliação do Tabuleiro 8](#_Toc24835841)

[Jogada do Computador 8](#_Toc24835842)

[Conclusões 9](#_Toc24835843)

[Bibliografia 10](#_Toc24835844)

# 

# 

# Introdução

O primeiro projeto da unidade curricular Programação em Lógica (PLOG) tem âmbito o desenvolvimento de um jogo na linguagem Prolog. O nosso grupo selecionou, de entre as opções disponíveis, o jogo Iris, descrito na secção seguinte.

Este jogo foi implementado na sua totalidade, tendo várias funcionalidades: visualização do seu estado, movimentação das peças de cada jogador, verificação do estado final do jogo e cálculo do seu vencedor.

O jogo foi implementado segundo 3 métodos: humano contra humano, humano contra computador e computador contra computador. Os jogadores simulados por computadores são simulados de acordo com 4 níveis (fácil, médio, difícil e *hardcore*).

# O Jogo

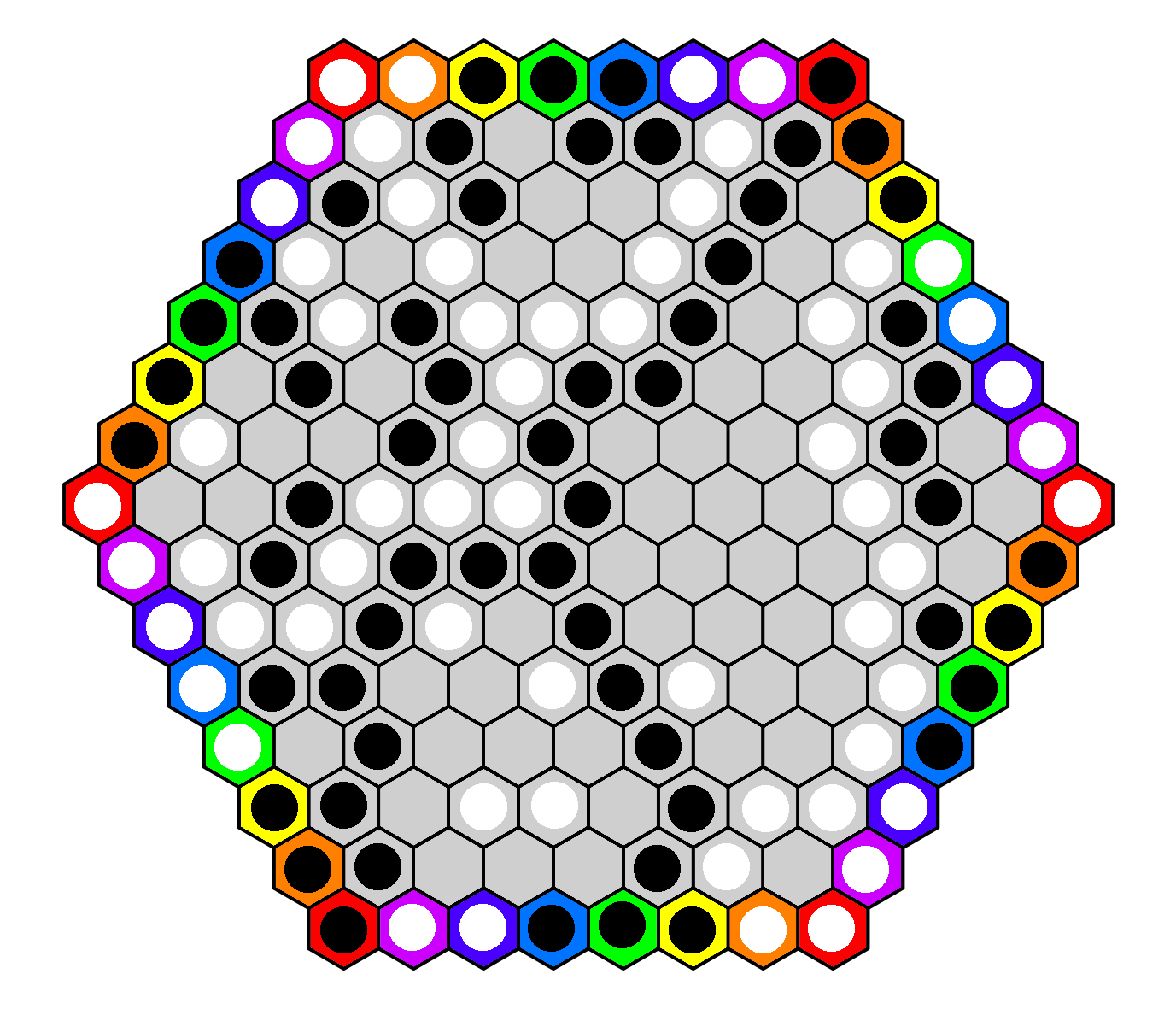
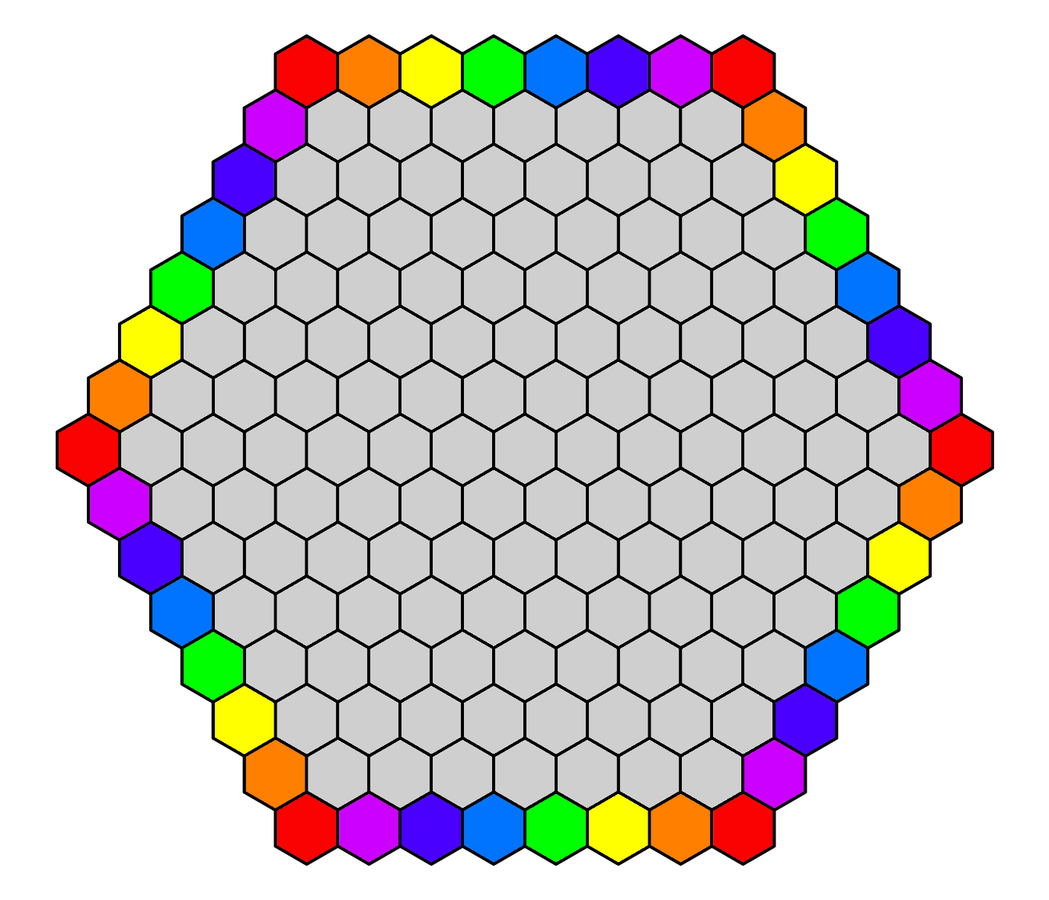
## História

Iris é um jogo que existe desde 2019 e que foi criado por Craig Duncan. Este jogo de tabuleiro enquadra-se na categoria “Estratégia Abstrata” segue um mecanismo de “Construção de Padrões” e é da família “Combinatória”. Deste modo, é um jogo sem tema/enredo, cujo resultado não é dependente da sorte / do acaso. Para além disso, é um jogo desenvolvido de forma a ter 2 jogadores, em que os jogadores alternam os turnos.

## Material

Para jogar este jogo, é necessário um tabuleiro “hexhex”, ou seja, um tabuleiro hexagonal com células hexagonais. As células que compõem o perímetro do tabuleiro são coloridas (formando as cores do arco-íris) e as restantes células são de cor cinzenta.

Para além do tabuleiro, são ainda necessárias peças de tamanho inferior ou igual às células do tabuleiro, sendo que algumas tem que ser pretas e outras brancas (pretas para o jogador 1 e brancas para o jogador 2).



**Fig. 1** - Aspeto do tabuleiro de Iris, sem e com peças.

## Regras

Na primeira jogada, o jogador 1 deverá colocar uma única peça preta numa célula cinzenta à escolha. A partir daí, começando o jogador 2, cada jogador deverá colocar 2 peças em cada jogada, seguindo as duas seguintes regras:

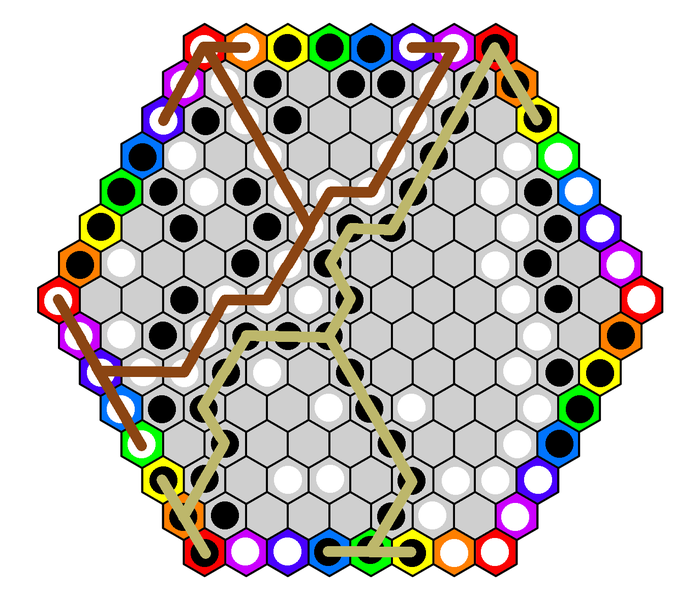
1. Se um jogador coloca a primeira peça numa célula colorida, a segunda peça deverá ser colocada na célula com a mesma cor que se encontra no lado oposto do tabuleiro;
2. Se a primeira peça é colocada numa célula cinzenta, então a segunda peça deverá ser colocada numa célula cinzenta não adjacente à primeira célula. Se já todas as células cinzentas não adjacentes estiverem ocupadas, a segunda peça deverá ser ignorada.

Quando todas as células estão preenchidas ou ambos os jogadores “passam” a sua vez, o jogo termina.

## Pontuação e Vencedor

Cada jogador deverá dividir as suas peças em grupos, sendo que cada grupo é constituído por peças adjacentes. Ganha o jogador que tiver o grupo com maior pontuação. Se os grupos de maior pontuação dos dois jogadores possuírem a mesma pontuação, então comparam-se os grupos com 2ª maior pontuação, e assim consecutivamente, até ocorrer o desempate.

A pontuação de cada grupo é equivalente ao número de células coloridas que ocupa.



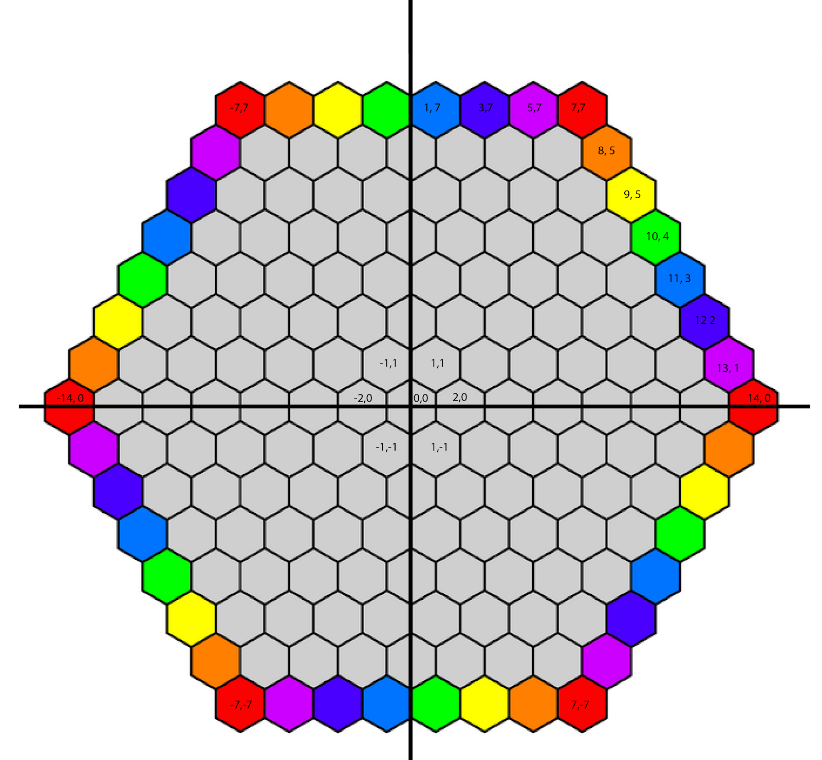
**Fig. 2** - Exemplo de grupos com maior pontuação no final de um jogo.

# A Lógica

## Representação do Estado do Jogo

A representação interna do jogo consiste numa lista de listas de listas. Deste modo o Tabuleiro é representado por uma lista de Linhas. Cada linha é uma lista, cujo *header* é o número da linha e cujos restantes elementos representam as várias células que compõem a linha. Cada célula é representada por uma lista composta por dois elementos: o número da coluna da célula e o estado atual da célula. O estado atual da célula poderá ter 3 valores: ‘B’ se a célula estiver vazia, 1 se a célula estiver preenchida por uma peça do jogador 1 (preta) ou 2 se a célula estiver preenchida por uma peça do jogador 2 (branca).

O sistema de coordenadas utilizado para um tabuleiro 15x15 é o apresentado abaixo.



**Fig. 3** – Sistema de Coordenadas do Jogo.

**Estado Inicial**

[[7, [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B']],

[6, [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B']],

[5, [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B']],

[4, [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B']],

[3, [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B']],

[2, [-12, 'B'], [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B'], [12, 'B']],

[1, [-13, 'B'], [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B'], [13, 'B']],

[0, [-14, 'B'], [-12, 'B'], [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B'], [12, 'B'], [14, 'B']],

[-1, [-13, 'B'], [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B'], [13, 'B']],

[-2, [-12, 'B'], [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B'], [12, 'B']],

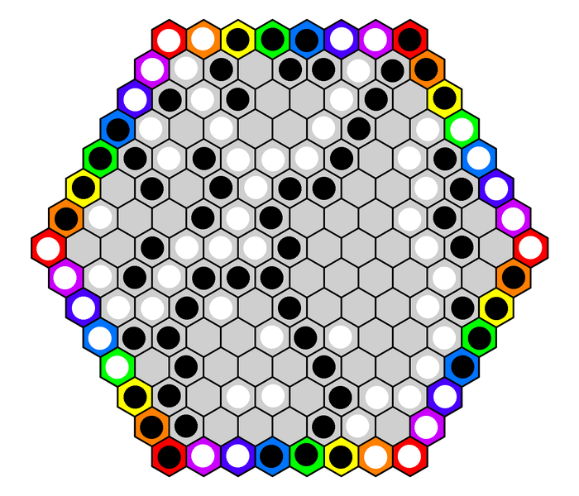
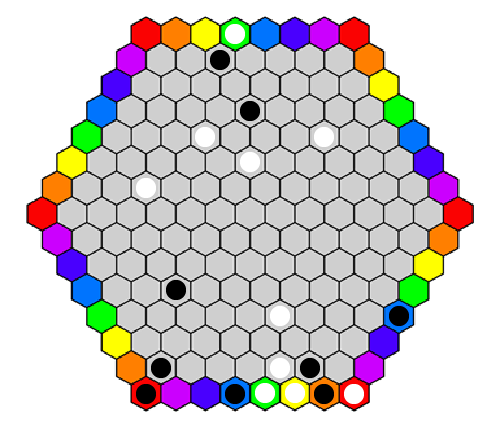
[-3, [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B']],

[-4, [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B']],

[-5, [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B']],

[-6, [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B']],

[-7, [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B']]]



**Fig. 4** – Estado Intermédio do Jogo

**Fig. 5** – Estado Final do Jogo.

**Estado Intermédio**

[[7, [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, '2'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B']],

[6, [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, '1'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B']],

[5, [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B']],

[4, [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, '1'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B']],

[3, [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, '2'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, '2'], [7, 'B'], [9, 'B'], [11, 'B']],

[2, [-12, 'B'], [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, '2'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B'], [12, 'B']],

[1, [-13, 'B'], [-11, 'B'], [-9, 'B'], [-7, '2'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B'], [13, 'B']],

[0, [-14, 'B'], [-12, 'B'], [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B'], [12, 'B'], [14, 'B']],

[-1, [-13, 'B'], [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B'], [13, 'B']],

[-2, [-12, 'B'], [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, 'B'], [4, 'B'], [6, 'B'], [8, 'B'], [10, 'B'], [12, 'B']],

[-3, [-11, 'B'], [-9, 'B'], [-7, 'B'], [-5, '1'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B'], [11, 'B']],

[-4, [-10, 'B'], [-8, 'B'], [-6, 'B'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, '2'], [4, 'B'], [6, 'B'], [8, 'B'], [10, '1']],

[-5, [-9, 'B'], [-7, 'B'], [-5, 'B'], [-3, 'B'], [-1, 'B'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, 'B']],

[-6, [-8, 'B'], [-6, '1'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, '2'], [4, '1'], [6, 'B'], [8, 'B']],

[-7, [-7, '1'], [-5, 'B'], [-3, 'B'], [-1, '1'], [1, '2'], [3, '2'], [5, '1'], [7, '2']]]

**Estado Final**

[[7, [-7, '2'], [-5, '2'], [-3, '1'], [-1, '1'], [1, '1'], [3, '2'], [5, '2'], [7, '1']],

[6, [-8, '2'], [-6, '2'], [-4, '1'], [-2, 'B'], [0, '1'], [2, '1'], [4, '2'], [6, '1'], [8, '1']],

[5, [-9, '2'], [-7, '1'], [-5, '2'], [-3, '1'], [1, 'B'], [1, 'B'], [3, '2'], [5, '1'], [7, 'B'], [9, '1']],

[4, [-10, '1'], [-8, '2'], [-6, 'B'], [-4, '2'], [-2, 'B'], [0, 'B'], [2, '2'], [4, '1'], [6, 'B'], [8, '2'], [10, '2']],

[3, [-11, '1'], [-9, '1'], [-7, '2'], [-5, '1'], [-3, '2’], [-1, '2'], [1, '2'], [3, '1'], [5, 'B'], [7, '2'], [9, '1'], [11, '2']],

[2, [-12, '1'], [-10, 'B'], [-8, '1'], [-6, 'B'], [-4, '1'], [-2, '2'], [0, '1'], [2, '1'], [4, 'B'], [6, 'B'], [8, '2'], [10, '1'], [12, '2']],

[1, [-13, '1'], [-11, '2'], [-9, 'B'], [-7, 'B'], [-5, '1'], [-3, '2'], [-1, '1'], [1, 'B'], [3, 'B'], [5, 'B'], [7, '2'], [9, '1'], [11, 'B'], [13, '2']],

[0, [-14, '2'], [-12, 'B'], [-10, 'B'], [-8, '1'], [-6, '2'], [-4, '2'], [-2, '2'], [0, '1'], [2, 'B'], [4, 'B'], [6, 'B'], [8, '2'], [10, '1'], [12, 'B'], [14, '2']],

[-1, [-13, '2'], [-11, '2'], [-9, '1'], [-7, '2'], [-5, '1'], [-3, '1'], [-1, '1'], [1, 'B'], [3, 'B'], [5, 'B'], [7, 'B'], [9, '2'], [11, 'B'], [13, '1']],

[-2, [-12, '2'], [-10, '2'], [-8, '2'], [-6, '1'], [-4, '2'], [-2, 'B'], [0, '1'], [2, 'B'], [4, 'B'], [6, 'B'], [8, '2'], [10, '1'], [12, '1']],

[-3, [-11, '2'], [-9, '1'], [-7, '1'], [-5, 'B'], [-3, 'B'], [-1, '2'], [1, '1'], [3, '2'], [5, 'B'], [7, 'B'], [9, '2'], [11, '1']],

[-4, [-10, '2'], [-8, 'B'], [-6, '1'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, '1'], [4, 'B'], [6, 'B'], [8, '2'], [10, '1']],

[-5, [-9, '1'], [-7, '1’], [-5, 'B'], [-3, '2'], [-1, '2'], [1, 'B'], [3, '1'], [5, '2'], [7, '2'], [9, '2']],

[-6, [-8, '1'], [-6, '1'], [-4, 'B'], [-2, 'B'], [0, 'B'], [2, '1'], [4, '2'], [6, 'B'], [8, '2']],

[-7, [-7, '1'], [-5, '2'], [-3, '2'], [-1, '1'], [1, '1'], [3, '1'], [5, '2'], [7, '2']]]

## Visualização do Tabuleiro

A visualização do tabuleiro *hexhex* de dimensões 15x15 é feita com recurso ao código abaixo apresentado:

drawSpace(0).

drawSpace(N) :- write(' '), N1 is N-1, drawSpace(N1).

displayCell([\_|[P]]):- write(P), write(' ').

displayLineCells([]).

displayLineCells([H|T]) :- displayCell(H), displayLineCells(T).

displayLine([H|T]) :- (H<0 -> write(H); write(' '), write(H)), (H>0 -> N1 is H+1, drawSpace(N1); N1 is -H+1, drawSpace(N1)), (H<0 -> write('\\ '); (H>0 -> write('/ '); write('| '))), displayLineCells(T), (H<0 -> write('/ \n'); (H>0 -> write('\\ \n'); write('| \n'))).

displayBoard([]).

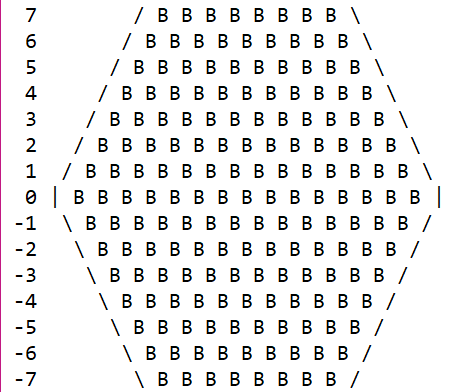
displayBoard([H|T]) :- displayLine(H), displayBoard(T).

display\_game(Board, \_) :- displayBoard(Board).

Desta forma, para se poder visualizar o tabuleiro deve-se ser usado o predicado display\_game(+Board,+Player).

As peças do jogador 1 (peças brancas) são representadas pelo símbolo ‘1’, as peças do jogador 2 (peças pretas) são representadas pelo símbolo ‘2’ e as células vazias (sem nenhuma peça) são representadas pelo símbolo ‘B’.

Usando a lista de listas correspondente ao tabuleiro no estado inicial, no predicado display\_game, já mencionado acima, o resultado obtido é o visível na imagem:



**Fig. 6** – Visualização do Tabuleiro em Modo de Texto

## Lista de Jogadas Válidas

Para obter uma lista com todas as jogadas possíveis para um jogador, dado um determinado estado do jogo, deve ser usado o predicado valid\_moves(+Board, +Player, -ListOfMoves). Este predicado é implementado através do seguinte código:

generateMovesFromLine(\_, [], \_, \_, \_, \_, []).

generateMovesFromLine(Board, [[CellBoard | \_] | T],[Cell | Value], LineBoard, Line, Player, ValidMoves) :- generateMovesFromLine(Board, T, [Cell, Value], LineBoard, Line, Player, ValidMovesAux), (verifyMove(Board, Line, Cell, LineBoard, CellBoard), (Line =:= LineBoard, Cell =:=CellBoard -> fail;!) -> append(ValidMovesAux, [ [Line,Cell, LineBoard, CellBoard] ], ValidMoves); ValidMoves = ValidMovesAux, !).

generateValidMovesCell(\_, [], \_, \_, \_, []).

generateValidMovesCell(Board, Board, [Cell, \_], Line, \_, ValidMoves):- cellColor(Line, Cell), cellEmpty(Board, Line, Cell), Line2 is -Line, Column2 is -Cell, ValidMoves = [[Line, Cell, Line2, Column2]].

generateValidMovesCell(Board, [[LineBoard | Cells ] | T], [Cell, Value], Line, Player, ValidMoves):- generateValidMovesCell(Board, T, [Cell, Value], Line, Player, ValidMovesAux), generateMovesFromLine(Board, Cells, [Cell, Value], LineBoard, Line, Player, ValidFromLine), append(ValidMovesAux, ValidFromLine, ValidMoves).

generateValidMovesLine(\_, \_, [], \_, []).

generateValidMovesLine(Board, Line, [ Cell | T], Player, ValidMoves):- generateValidMovesLine(Board, Line, T, Player, ValidMovesAux), generateValidMovesCell(Board, Board, Cell, Line, Player, ValidInCell), append(ValidMovesAux, ValidInCell, ValidMoves).

generateValidMoves(\_, [], \_, []).

generateValidMoves(Board, [[Line| Cells] | T ], Player, ValidMoves):- generateValidMoves(Board, T, Player, ValidMovesAux), generateValidMovesLine(Board, Line, Cells, Player, ValidInLine), append(ValidMovesAux, ValidInLine, ValidMoves).

valid\_moves(Board, Player, ListOfMoves) :- generateValidMoves(Board, Board, Player, ListOfMoves).

Sílvia explica isto por favor que eu não sei explicar.

## Execução de Jogadas

De forma a executar uma jogada, deve-se utilizar o predicado move(+Move, +Board, -NewBoard). O código seguinte implementa uma parte deste mesmo predicado:

verifyMove(Board, Line1, Column1, [], []) :- cellEmpty(Board, Line1, Column1), \+cellColor(Line1, Column1).

verifyMove(Board, Line1, Column1, Line2, Column2) :- cellEmpty(Board, Line1, Column1), cellEmpty(Board, Line2, Column2), cellColor(Line1, Column1), Line2 =:= -Line1, Column2 =:= -Column1.

verifyMove(Board, Line1, Column1, Line2, Column2) :- cellEmpty(Board, Line1, Column1), cellEmpty(Board, Line2, Column2), \+cellColor(Line1, Column1), \+cellColor(Line2, Column2), \+adjacentPieces(Line1, Column1, Line2, Column2).

move([Player,Line1,Column1,Line2,Column2], Board, NewBoard) :- verifyMove(Board, Line1, Column1, Line2, Column2), implement\_moves([Player, Line1, Column1, Line2, Column2], Board, NewBoard).

O predicado cellValue permite obter qual o valor atual de uma determinada célula, dada a sua linha e coluna. O predicado cellEmpty permite descubrir se uma célula está vazia. O predicado adjacentPieces permite descubrir se duas células são adjacentes. O predicado cellColor permite saber se uma célula é colorida. Todos estes predicados são predicados auxiliares mas essenciais para o predicado move (essencialmente para o verifyMove).

O predicado verifyMove permite saber se um determinado movimento é válido, de acordo com as regras do jogo. O predicado implemente\_move implemente um determinado movimento, usando o predicado changeCell para efetivamente mudar o valor da célula em questão. O predicado move é responsável por verificar se uma jogada é possível (verifyMove) e efetuar a jogada (implemente\_moves).

## Final do Jogo

A verificação do estado final do jogo, assim como o cálculo das pontuações e a identificação do vencedor são efetuados pelo predicado game\_over(+Board, -Winner). Este predicado é implementado, em parte, pelo código seguinte:

calculateScore([], 0).

calculateScore([[Line,Column] | T], GroupPoints):- calculateScore(T, GroupPointsAux), (cellColor(Line, Column)-> GroupPoints is GroupPointsAux+1; GroupPoints is GroupPointsAux, !).

calculateGroup(\_, [], \_, InitialGroup, InitialGroup, InitialUsedCells, InitialUsedCells).

calculateGroup(PlayerCells, [[Line, Column]|T], [ColoredLine, ColoredColumn], Igroup, Fgroup, IusedCells, FusedCells) :- (adjacentPieces(Line, Column, ColoredLine, ColoredColumn), \+ member([Line, Column], IusedCells) ->

append(IusedCells, [[Line, Column]], UsedCells1),

append(Igroup, [[Line, Column]], Group1),

calculateGroup(PlayerCells, T, [ColoredLine, ColoredColumn], Group1, Group2, UsedCells1, UsedCells2),

calculateGroup(PlayerCells, PlayerCells, [Line, Column], Group2, Fgroup, UsedCells2, FusedCells);

calculateGroup(PlayerCells, T, [ColoredLine, ColoredColumn], Igroup, Fgroup, IusedCells, FusedCells)).

calculateGroups(\_, [], InitialGroups, InitialGroups, InitialUsedCells, InitialUsedCells).

calculateGroups(PlayerCells, [[Line, Column]|T], Igroups, Fgroups, IusedCells, FusedCells) :-

(member([Line,Column], IusedCells) ->

calculateGroups(PlayerCells, T, Igroups, Fgroups, IusedCells, FusedCells);

append(IusedCells, [[Line, Column]], UsedCells1),

calculateGroup(PlayerCells, PlayerCells, [Line, Column], [[Line, Column]], Group1, UsedCells1, UsedCells2),

append(Igroups, [Group1], Group2),

calculateGroups(PlayerCells, T, Group2, Fgroups, UsedCells2, FusedCells)).

calculatePoints(Board, Player, Points) :-

calculateCellsPlayerLines(Board, Player, PlayerCells),

calculateColored(PlayerCells, ColoredCells),

calculateGroups(PlayerCells, ColoredCells, [], FinalGroups, [], \_),

calculateGroupsScore(FinalGroups, Points).

calculateWinner([], [], 3).

calculateWinner([], \_, 1).

calculateWinner(\_, [], 2).

calculateWinner(PointsP1, PointsP2, Winner):- maxlist(PointsP1, MaxP1), maxlist(PointsP2, MaxP2),

(MaxP1 == MaxP2 ->

deleteElement(PointsP1, MaxP1, [], NewPointsP1),

deleteElement(PointsP2, MaxP2, [], NewPointsP2),

calculateWinner(NewPointsP1,NewPointsP2, Winner);

(MaxP2 > MaxP1 -> Winner = 2; Winner = 1)).

game\_over\_sure(Board, Winner) :- calculatePoints(Board,1,PointsP1), calculatePoints(Board,2,PointsP2), calculateWinner(PointsP1, PointsP2, Winner).

game\_over(Board, Winner) :- (boardFull(Board) -> calculatePoints(Board,1,PointsP1), calculatePoints(Board,2,PointsP2), calculateWinner(PointsP1, PointsP2, Winner); Winner = 0) .

Para verificar o estado final do jogo usa-se o predicado BoardFull, que verifica se o tabuleiro está cheio.

O predicado calculateCellsPlayerLines permite calcular as células que estão ocupadas por peças de um determinado jogador. O predicado calculateColored permite calcular, para um dado conjunto de células, quais delas são coloridas e é utilizado, sobretudo, para calcular as células coloridas de um jogador, sendo chamado com o resultado de calculateCellsPlayerLines.

O predicado calculateGroup é responsável por calcular um grupo a partir de uma célula inicial (colorida) que faz parte desse grupo. Este predicado é utilizado no predicado calculateGroups que calcula todos os grupos de um determinado jogador. O predicado calculateScore calcula os pontos de um determinado grupo de células, sendo chamada por calculateGroupsScore que calcula os pontos de todos os grupos de um jogador. O predicado calculatePoints calcula um vetor com as pontuações de todos os grupos de um jogador.

O predicado calculateWinner verifica qual o vencedor, analisando consecutivamente os vários grupos com maiores pontuações, de acordo com as regras do jogo.

O predicado game\_over, após verificar se o board está cheio (BoardFull), calcula as pontuações de cada jogador (calculatePoints) e decide qual o vencedor (calculateWinner).

É ainda de notar o predicado game\_over\_sure que é chamado no caso de o final do jogo ser detetado através da interface do jogo (quando os dois jogadores passam consecutivamente) e não através do estado do jogo.

## Avaliação do Tabuleiro

O predicado value(+Board, +Player, -Value) permite obter o valor de um determindo Board para um determinado jogador, permitindo a comparação entre Boards para decisão de melhores jogadas.

O cálculo do valor de um tabuleiro é calculado com base nas células ocupadas por peças do jogador em questão. O valor/peso de uma célula é calculado com base na sua distância ao centro do tabuleiro/às células coloridas. Desta forma, quando mais longe uma célula estiver do centro (quanto mais perto estiver de uma célula colorida) maior é a sua utilidade, visto que permitirá no futuro construir grupos melhores que, por sua vez, providenciarão uma melhor pontuação. Assim, este predicado é implementado da seguinte forma:

calculateCellWeight([], [], 0).

calculateCellWeight(Line, Column, Weight):- Weight is 2\*abs(Line)+abs(Column).

value(Board, Player, Value) :- calculateCellsPlayerLines(Board, Player, PlayerCells), calculateCellsWeight(PlayerCells, Value).

O predicado calculateCellWeight calcula o peso/valor de uma determinada célula. O predicado calculateCellsWeight, por sua vez, calcula o peso de um determinado conjunto de células. Por fim, o predicado value, calcula as células ocupadas por um determinado jogador (calculateCellsPlayerLines) e, de seguida, calcula o peso total dessas células (calculateCellsWeight).

## Jogada do Computador

Para poder implementar modos de jogo que permitam que um ou dois dos jogadores sejam “simulados” por computadores é necessário que exista um predicado que escolha a melhor jogada possível: choose\_move(+Board, +Level, +Player, -Move).

Este predicado contém uma pequena diferença relativamente ao solicitado choose\_move(+Board, +Level, -Move). Esta diferença deve-se ao facto de, no jogo Iris ser necessário saber qual o jogador para o qual deve ser efetuado a jogada de forma a efetuar a melhor jogada possível.

# Conclusões

Finalizando o projeto e analisando o trabalho desenvolvido, pensamos que conseguimos construir um bom projeto e chegar a um bom resultado, sendo que atingimos os principais requisitos propostos pelos professores da unidade curricular.

No entanto, é de ressaltar que alguns melhoramentos poderiam ser aplicados. Um dos aspetos que poderia ser aperfeiçoado é a *interface* do jogo. Contudo, este não nos parece um ponto muito relevante, visto que a mesma vai ser trabalhada ao longo da cadeira de LAIG.

# Bibliografia

[Board Game Geek](https://www.boardgamegeek.com/boardgame/286792/iris)

[Wikipedia](https://pt.wikipedia.org/wiki/Jogo_de_estrat%C3%A9gia_abstrato)