

**Programação em Lógica**

**Relatório Final**

**Barragoon 4**

****

**Leonardo Manuel Gomes Teixeira – up201502848**

**Maria Eduarda Santos Cunha – up201506524**

# Resumo

Este projeto é uma possível abordagem ao jogo Barragoon, descrito em detalhe na secção O Jogo Barragoon, em PROLOG.

Todo o processo de desenvolvimento verificou-se bastante difícil dada a novidade do paradigma da linguagem de programação em questão para nós, aliado ao facto de o próprio jogo ter uma lógica bastante complexa, pelo seu número elevado de regras e restrições.

Conseguimos implementar com sucesso todas as regras do jogo e completar os três modos de jogo – humano contra humano, humano contra computador e computador contra computador. No entanto, acabamos por não criar diferentes níveis de dificuldade.

Apesar do volume de regras, pensamos que a versão final é bastante *user friendly* e intuitiva de jogar.

De forma geral, o nosso feedback é bastante positivo. Este trabalho foi fundamental para a consolidação dos conteúdos lecionados nas aulas práticas e teóricas e sentimo-nos capazes de realizar os novos níveis ainda que não tenhamos tido tempo para tal.

Índice

[1. Resumo 2](#_Toc498286024)

[2. O Jogo Barragoon 4](#_Toc498286025)

[2.1. Tabuleiro e Peças 4](#_Toc498286026)

[2.2. Objetivo 4](#_Toc498286027)

[2.3. Movimentos 5](#_Toc498286028)

[2.4. Regras 5](#_Toc498286029)

[3. Lógica do Jogo 7](#_Toc498286030)

[3.1. Representação do Estado de Jogo 7](#_Toc498286031)

[3.2. Visualização do Tabuleiro 9](#_Toc498286032)

[3.3. Lista de Jogadas Válidas 11](#_Toc498286033)

[3.4. Execução de Jogadas 11](#_Toc498286034)

[3.5. Avaliação do Tabuleiro 12](#_Toc498286035)

[3.6. Final do Jogo 12](#_Toc498286036)

[3.7. Jogada do Computador 12](#_Toc498286037)

[4. Interface com o Utilizador 13](#_Toc498286038)

[5. Conclusões 15](#_Toc498286039)

[6. Bibliografia 15](#_Toc498286040)

[7. Anexos 16](#_Toc498286041)

# O Jogo Barragoon

O Barragoon é um jogo de estratégia sem qualquer fator de aleatoriedade para 2 jogadores. Foi publicado pela primeira vez a 3 de março de 2014 pela companhia WiWa Spiele UG. As suas regras foram atualizadas pela última vez a 30 de março de 2016.

## Tabuleiro e Peças

O jogo realiza-se num tabuleiro de 9x7 células e os jogadores jogam sempre à vez.

Existem 2 tipos principais de peças: as telhas de cada jogador, brancas ou castanhas, e os barragoons.

* As telhas dos jogadores possuem na sua face um símbolo com 2, 3 ou 4 círculos, relativo ao número de células que podem andar num só movimento (fig.1). Cada jogador começa com 7 telhas: 2 de 2 círculos, 3 de 3 círculos e 2 de 4 círculos.
* O barragoon é a peça central do jogo. É uma peça cúbica, em que cada uma das suas faces possui um símbolo que indica a permissão do jogador de mover a sua peça pela célula em que o barragoon se encontra (fig.2). O jogo começa com 8 barragoons, mas existem 32.



**Sem passagem**

**Uma direção**

**Duas direções**

**Virar à direita**

**Virar à esquerda**

**Todas as direções**

Figura 1: Faces de um Barragoon



**2 células**

**3 células**

**4 células**

Figura 2: Telhas Brancas ou Castanhas

## Objetivo

Ambos os jogadores têm de recorrer às suas aptidões táticas para mover as suas telhas e dispor os barragoons de forma a que lhes seja permitido capturar todas as telhas do outro jogador ou, pelo menos, impedir o seu progresso.

O jogo acaba quando um dos jogadores já não consegue mover telhas, porque não possui nenhuma ou por as que tem se encontrarem limitadas por barragoons. O outro é o vencedor.

## Movimentos

Existem 2 tipos de movimentos: full moves e short moves.

* Os full moves correspondem a percorrer x células, de acordo com o número de círculos na telha do jogador (2, 3 ou 4), respetivamente (fig.3);
* Os short moves correspondem a percorrer x-1 células, de acordo com o número de círculos na telha do jogador (2, 3 ou 4, logo movimentos de 1, 2 ou 3 células), respetivamente (fig.4).



Figura 4: Short Move com Telha de 3 Círculos



Figura 3: Full Move com Telha de 4 Círculos

## Regras

* Uma peça é capturada se a peça do oponente terminar na mesma célula que ela (fig.5);
* Só é possível capturar uma peça durante um full move;
* As telhas com 2 círculos não podem capturar barragoons com o símbolo “todas as direções” virado para cima (fig.6);
* Se um barragoon for capturado, tem de voltar a ser colocado no tabuleiro, numa posição livre à escolha do jogador, com a face levantada para cima que ele preferir;
* Sempre que uma telha é capturada, são adicionados 2 barragoons novos ao tabuleiro, um por cada jogador, e coloca primeiro no tabuleiro o jogador cuja telha foi capturada;
* Nunca se pode mudar um barragoon de posição uma vez colocado;
* Quando se toca numa telha para a mover, não se pode trocar por outra ou voltar atrás;
* Durante um movimento, só se pode efetuar uma mudança de direção uma vez de 90º (fig.7);
* Os movimentos só podem ser verticais ou horizontais, nunca na diagonal.

Figura 6: Peça de 2 Círculos Não Captura Barragoon com Face “Todas as Direções” Voltada para Cima



Figura 5: Captura de um Barragoon

Figura 7: Movimento Impossível com 2 Mudanças de Direção



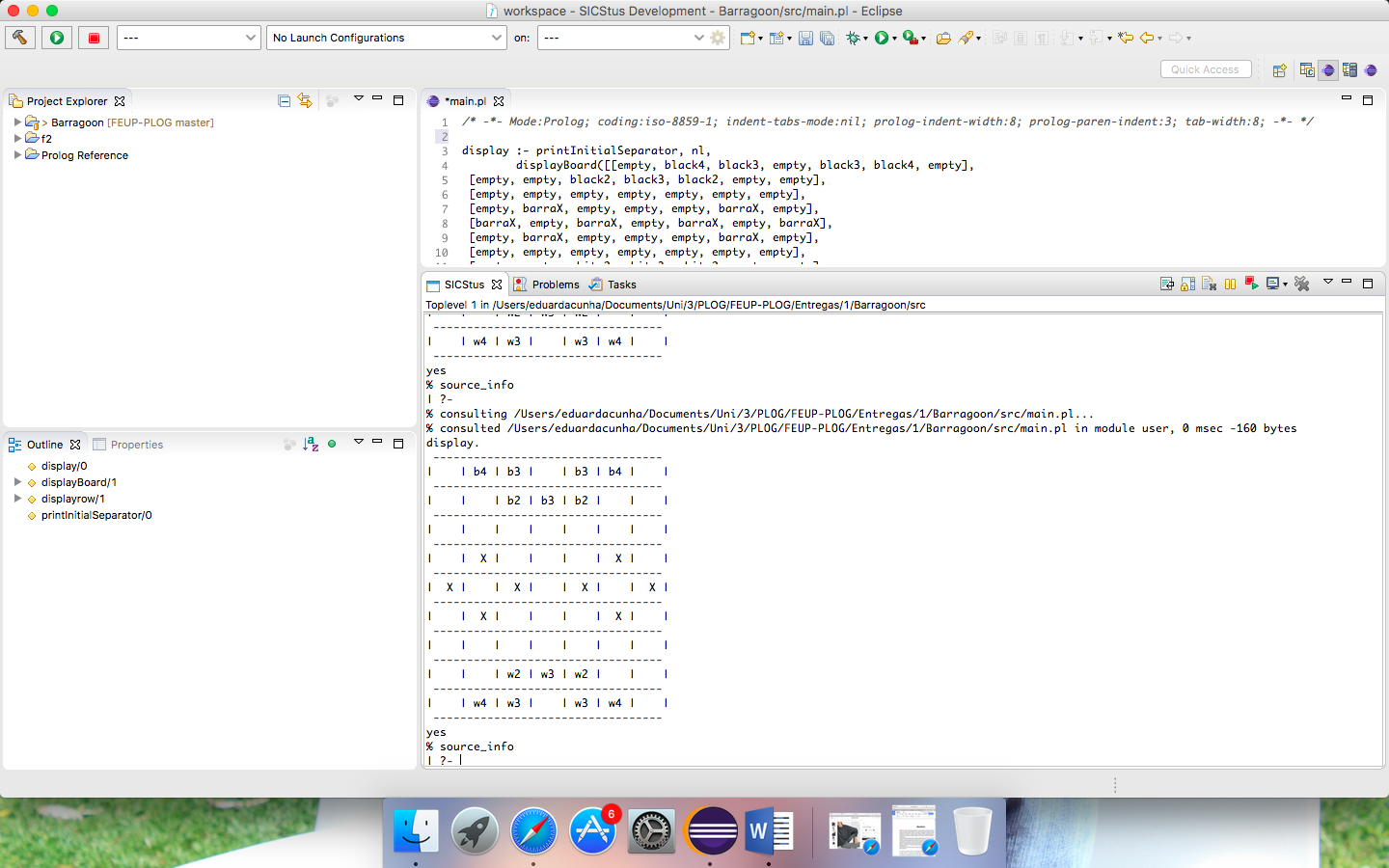
Ainda que só existam 32 barragoons e seria, por consequência, esperado que houvesse uma regra implementada para limitar a inserção de barragoons tendo em conta este valor, dado que sempre que um barragoon é comido é inserido o mesmo de volta e sempre que uma telha é comida são inseridos 2 e só há 7 telhas por jogador, concluímos que a situação em que se tenta inserir mais do que 32 nunca se vai verificar.

Por exemplo, tendo sido comidos 6 telhas de cada jogador, na situação final em que cada jogador possui 1 telha, quando um comer a do outro, o jogo acaba. E apenas nesse momento é que ocorreria a hipotética inserção do 33º.

# Lógica do Jogo

## Representação do Estado de Jogo

Por questões de simplificação, os barragoons encontram-se aqui representados apenas por, por exemplo, barraX, em vez de bg-'barraX'. Na prática e em todos os fragmentos de código, referimo-nos a cada barragoon por um par, cujo primeiro elemento é bg e o segundo é a face voltada para cima.

**Estado Inicial:**

[[empty, b-4, b-3, empty, b-3, b-4, empty],

[empty, empty, b-2, b-3, b-2, empty, empty],

[empty, empty, empty, empty, empty, empty, empty],

[empty, barraX, empty, empty, empty, barraX, empty],

[barraX, empty, barraX, empty, barraX, empty, barraX],

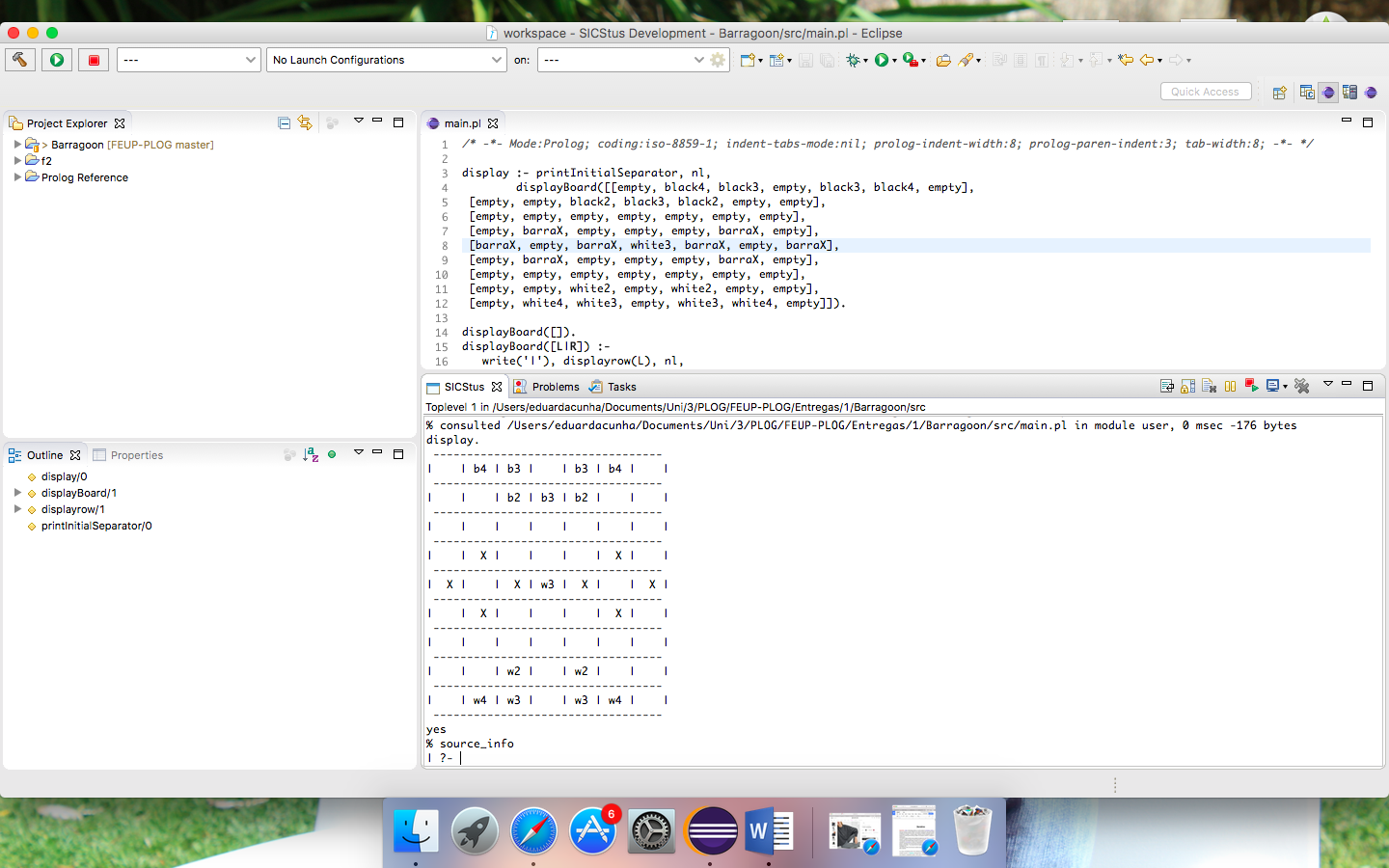
[empty, barraX, empty, empty, empty, barraX, empty],

[empty, empty, empty, empty, empty, empty, empty],

[empty, empty, w-2, w-3, w-2, empty, empty],

 [empty, w-4, w-3, empty, w-3, w-4, empty]]

**Estados intermédios:**

**1.**

[[empty, b-4, b-3, empty, b-3, b-4, empty],

[empty, empty, b-2, b-3, b-2, empty, empty],

[empty, empty, empty, empty, empty, empty, empty],

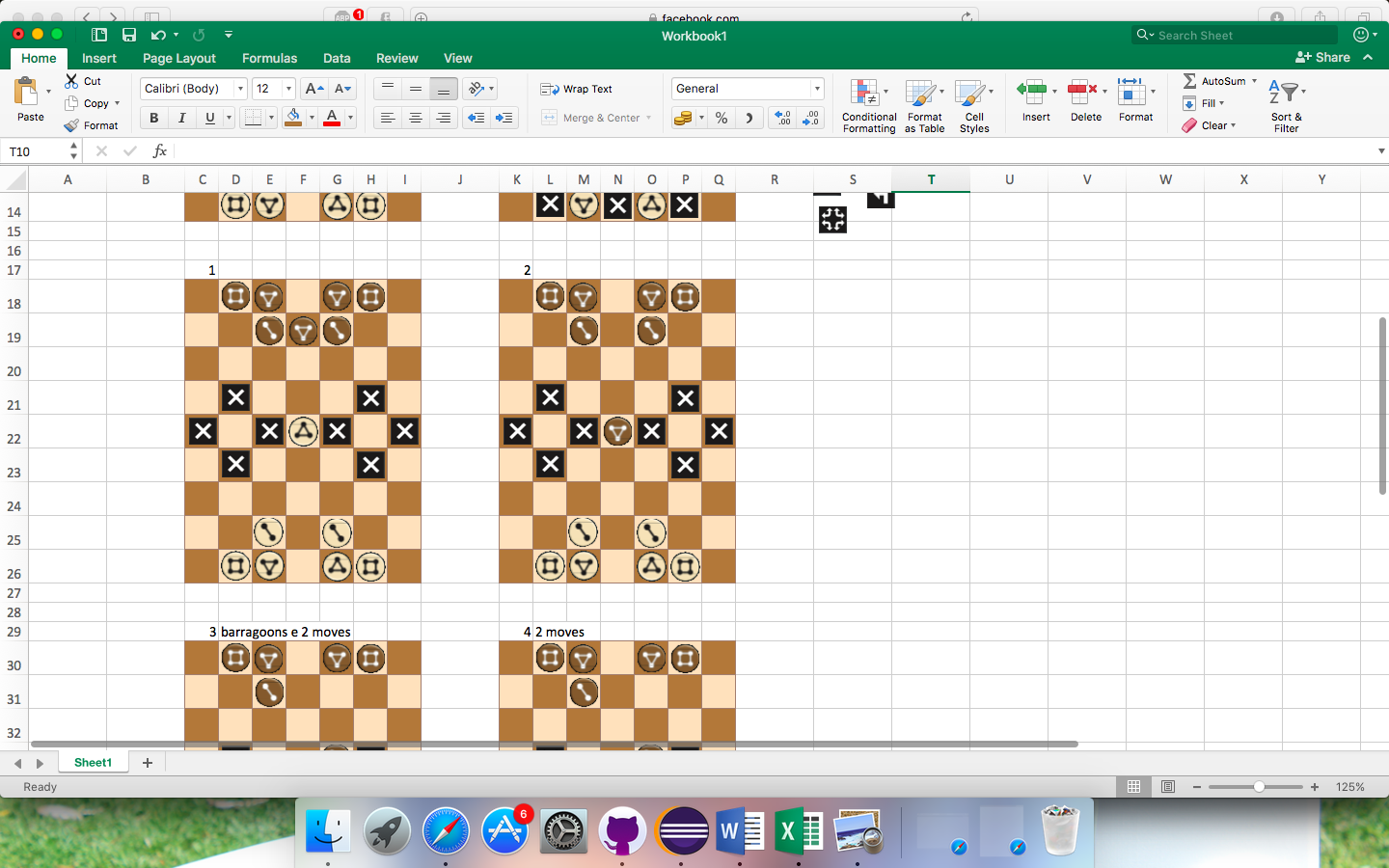
[empty, barraX, empty, empty, empty, barraX, empty],

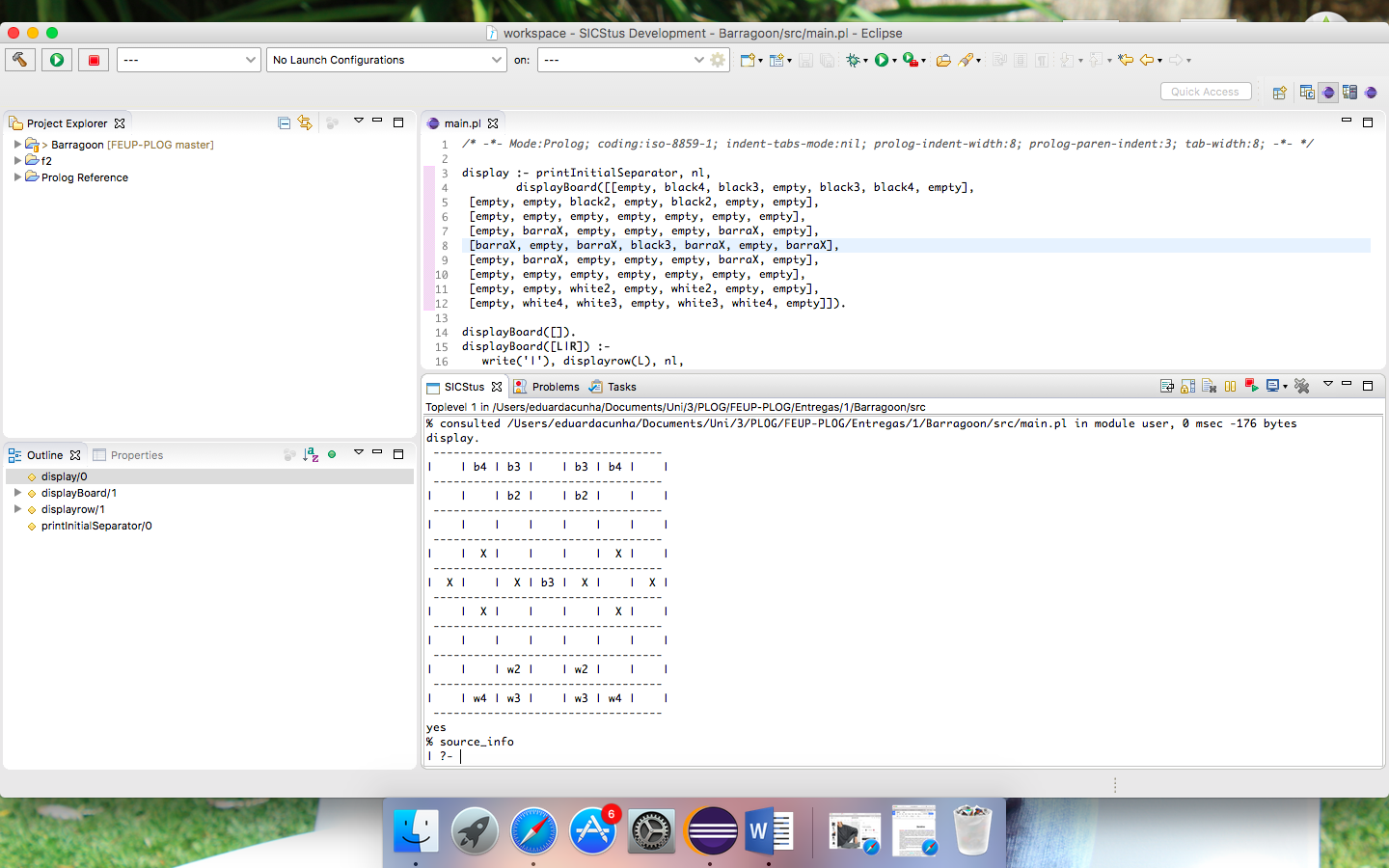
[barraX, empty, barraX, w-3, barraX, empty, barraX],

[empty, barraX, empty, empty, empty, barraX, empty],

[empty, empty, empty, empty, empty, empty, empty],

[empty, empty, w-2, empty, w-2, empty, empty],

 [empty, w-4, w-3, empty, w-3, w-4, empty]]

**2.**

[[empty, b-4, b-3, empty, b-3, b-4, empty],

[empty, empty, b-2, empty, b-2, empty, empty],

[empty, empty, empty, empty, empty, empty, empty],

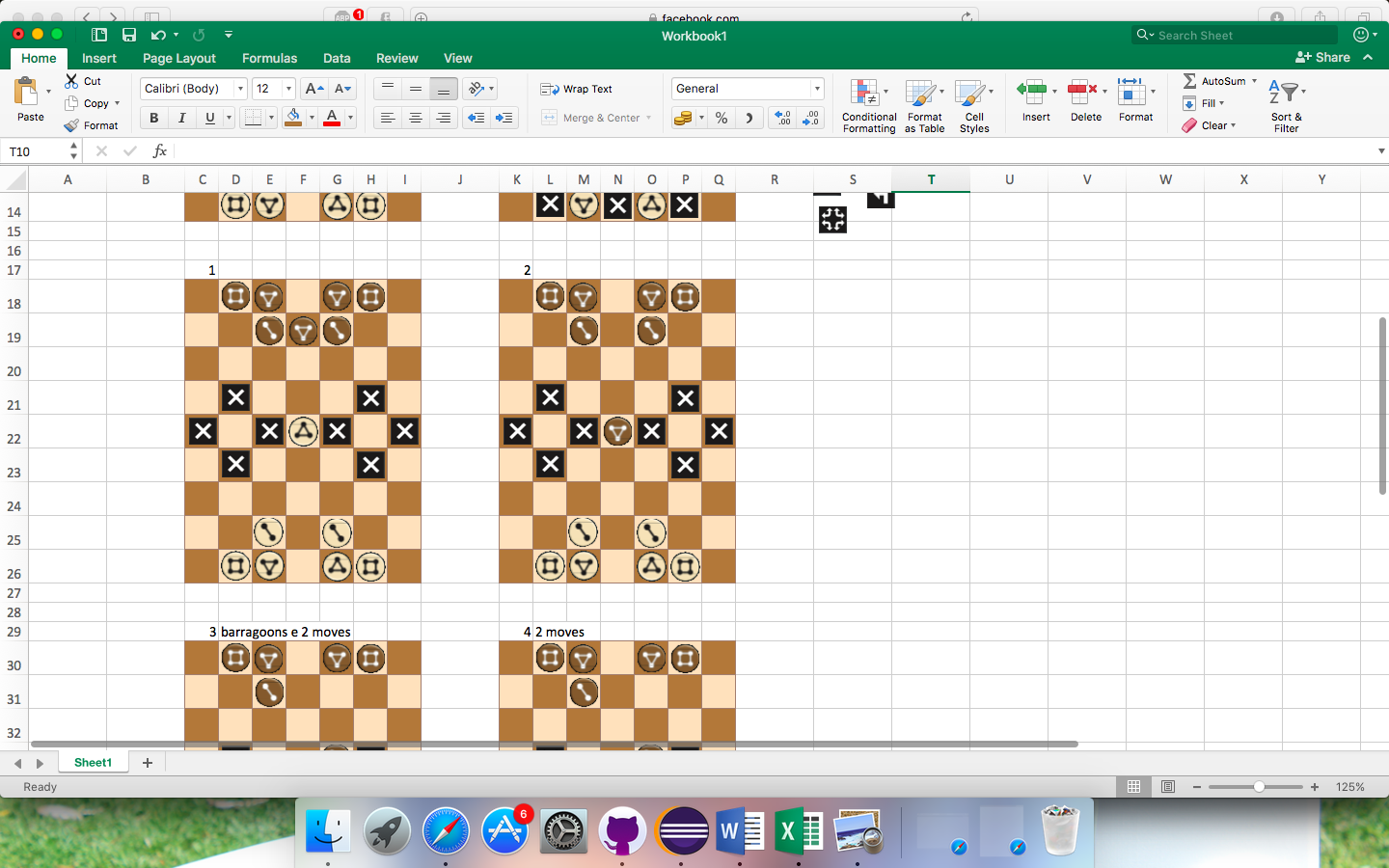
[empty, barraX, empty, empty, empty, barraX, empty],

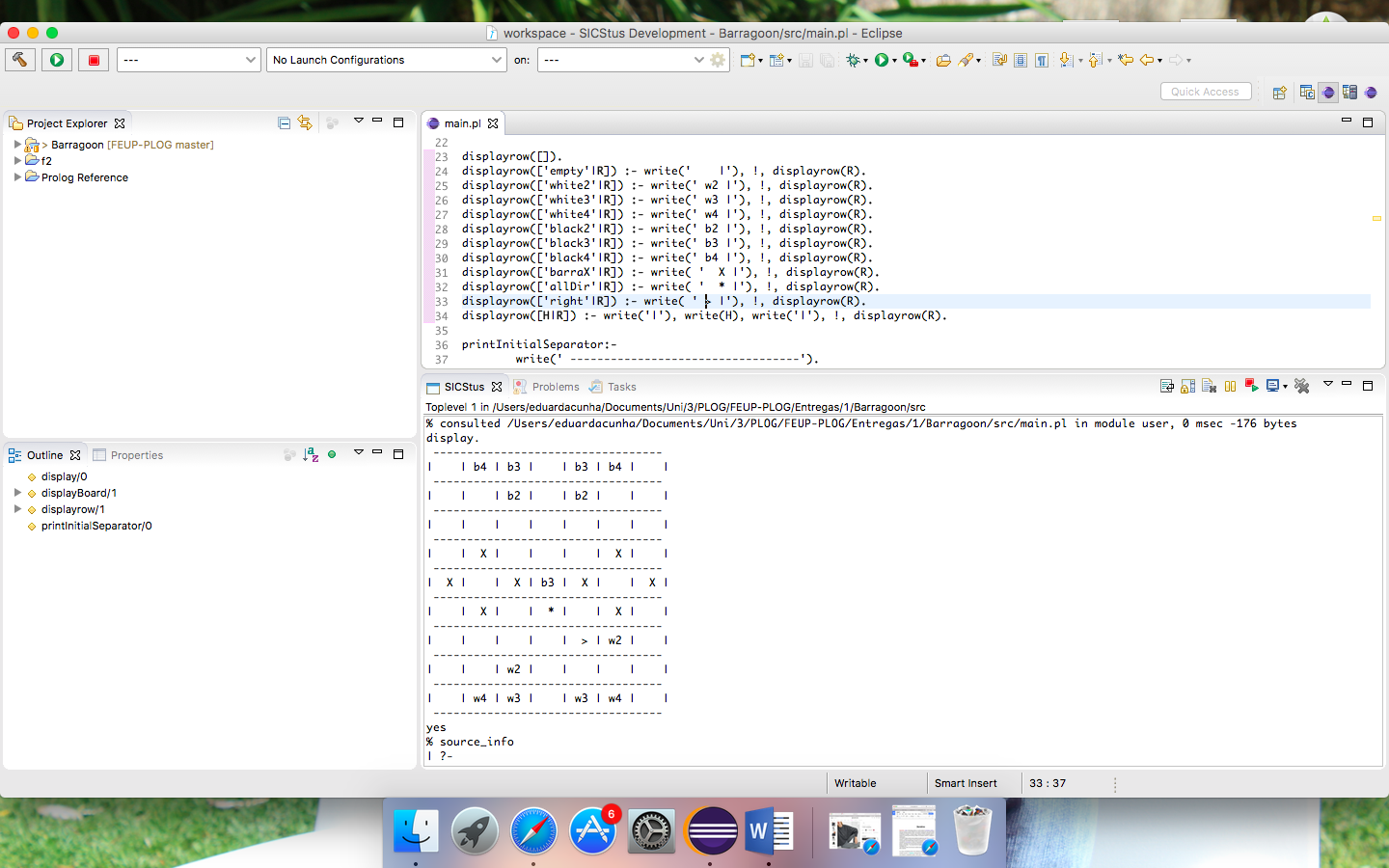
[barraX, empty, barraX, b-3, barraX, empty, barraX],

[empty, barraX, empty, empty, empty, barraX, empty],

[empty, empty, empty, empty, empty, empty, empty],

[empty, empty, w-2, empty, w-2, empty, empty],

**** [empty, w-4, w-3, empty, w-3, w-4, empty]]

**3.**

[[empty, b-4, b-3, empty, b-3, b-4, empty],

[empty, empty, b-2, empty, empty, empty, empty],

[empty, empty, empty, empty, empty, empty, empty],

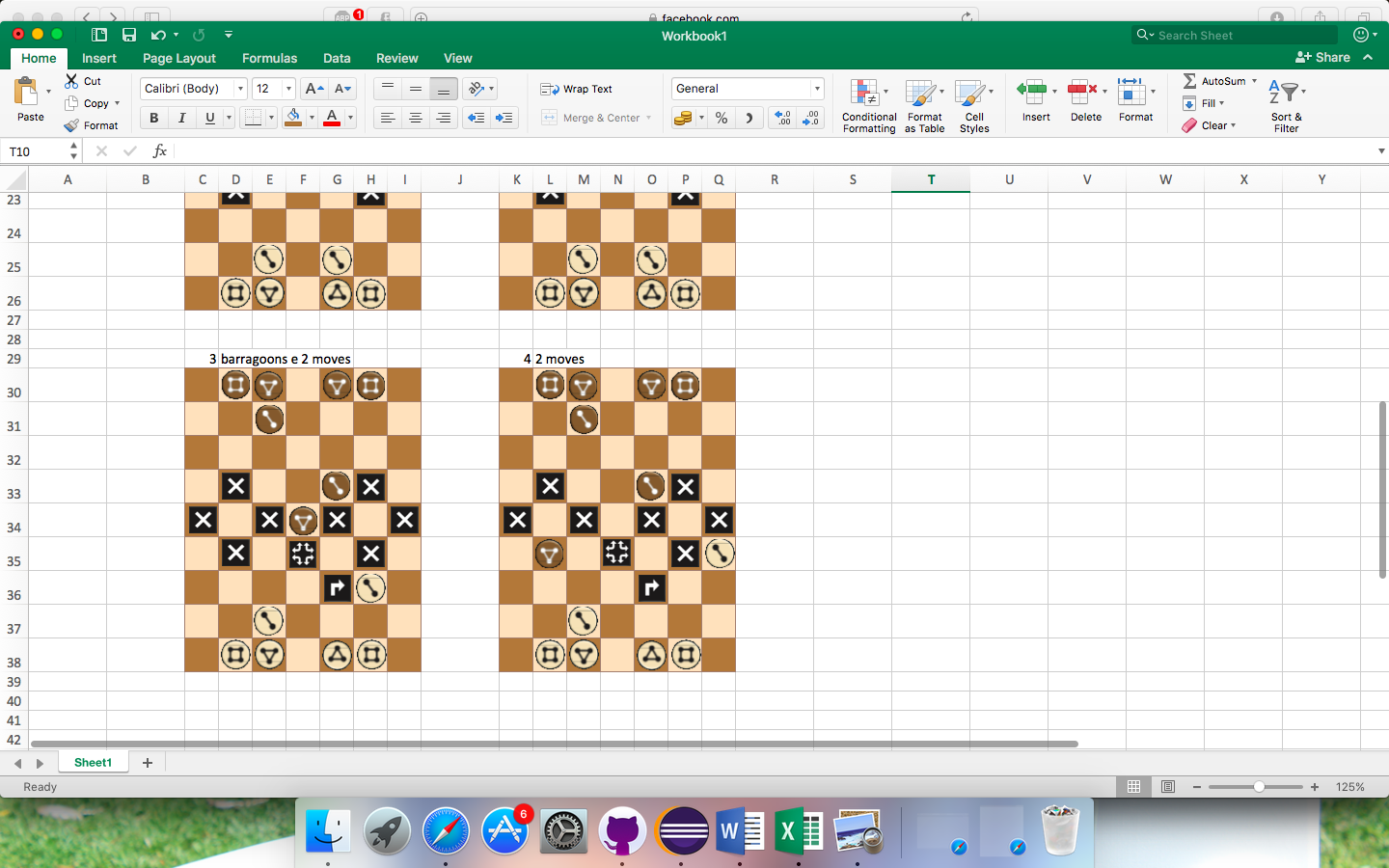
[empty, barraX, empty, empty, b-2, barraX, empty],

[barraX, empty, barraX, b-3, barraX, empty, barraX],

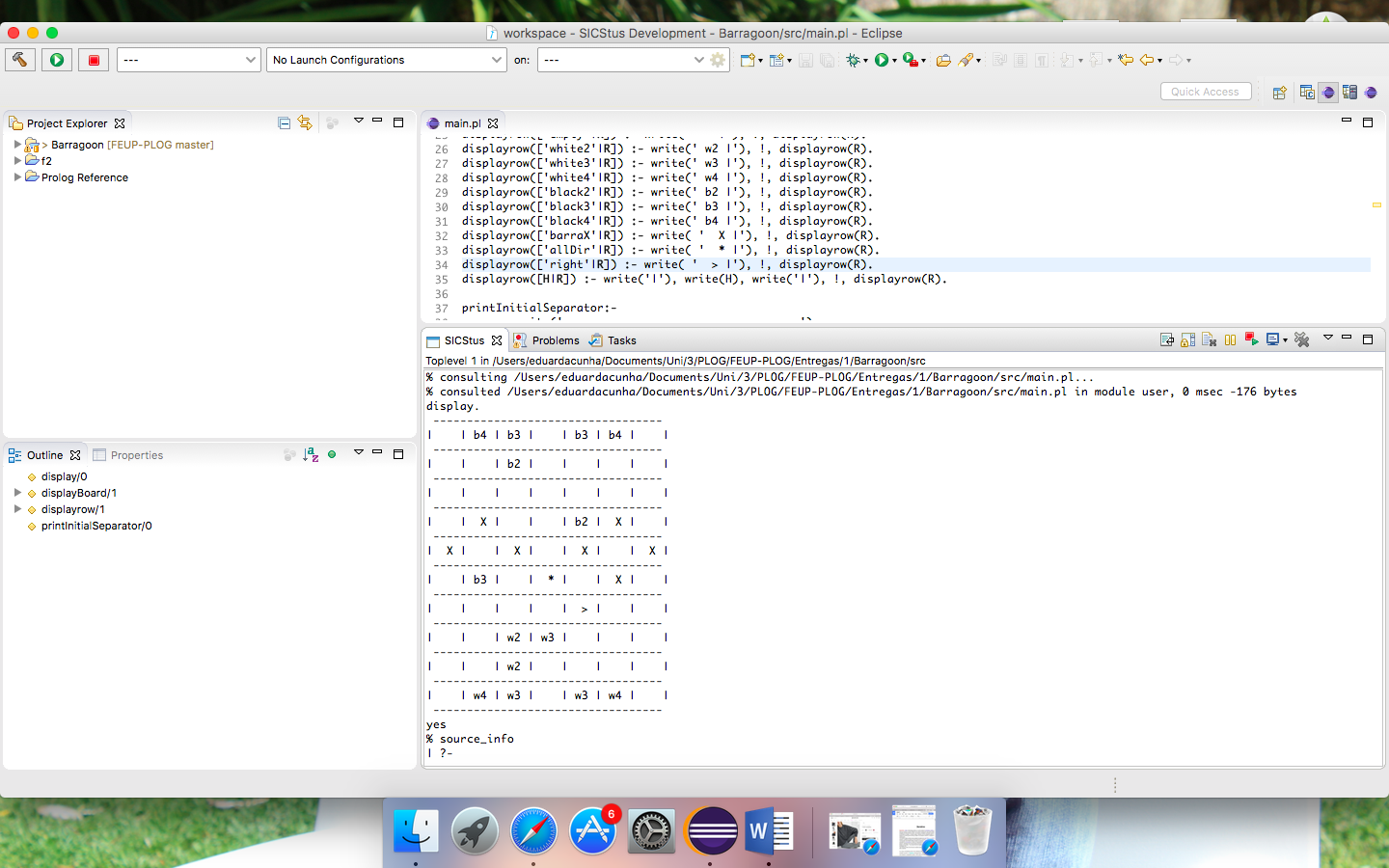
[empty, barraX, empty, allDir, empty, barraX, empty],

[empty, empty, empty, empty, right, w-2, empty],

[empty, empty, w-2, empty, empty, empty, empty],

 [empty, w-4, w-3, empty, w-3, w-4, empty]]

**4.**

****[[empty, b-4, b-3, empty, b-3, b-4, empty],

[empty, empty, b-2, empty, empty, empty, empty],

[empty, empty, empty, empty, empty, empty, empty],

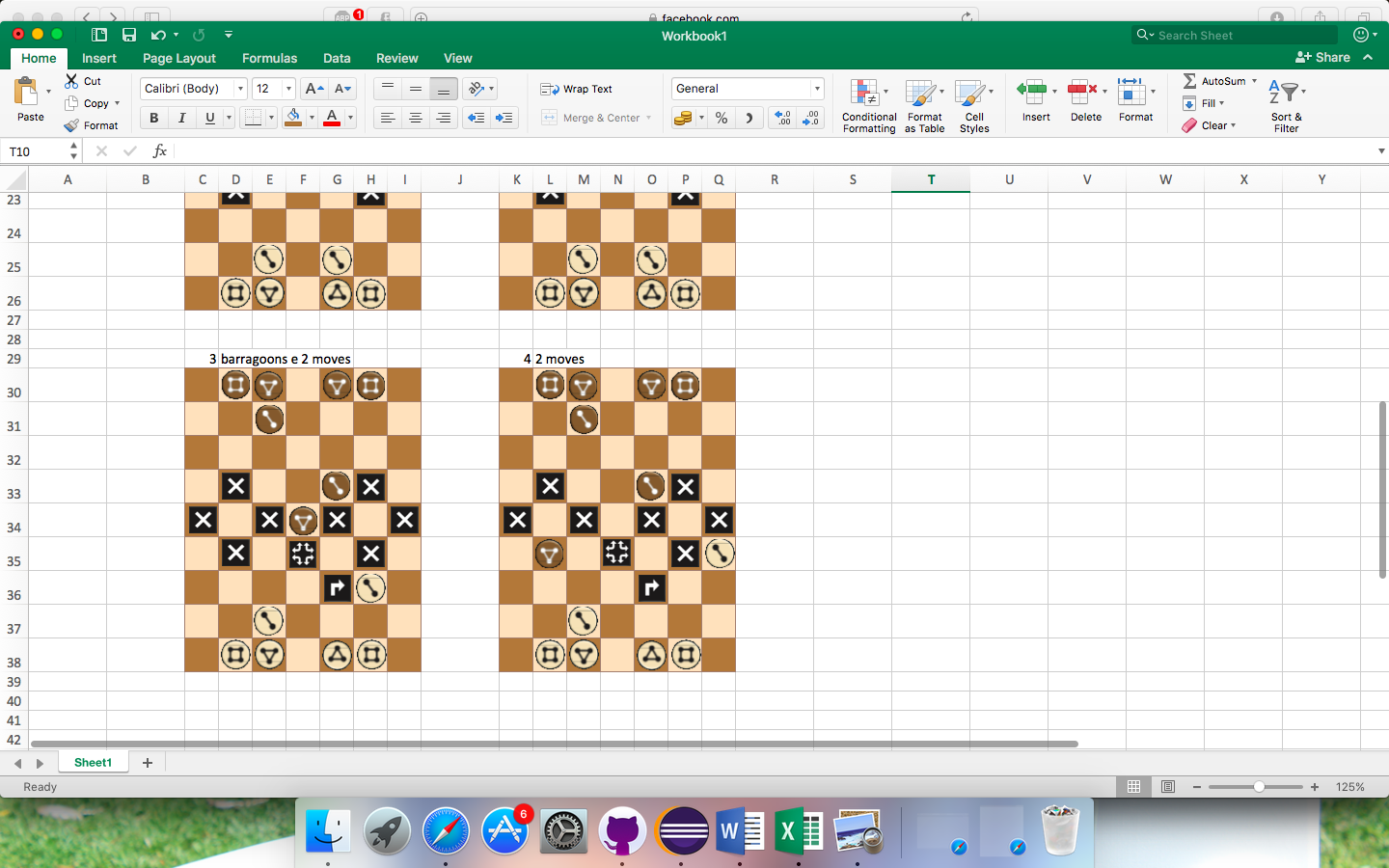
[empty, bg-'barraX', empty, empty, b-2, barraX, empty],

[barraX, empty, barraX, empty, barraX, empty, barraX],

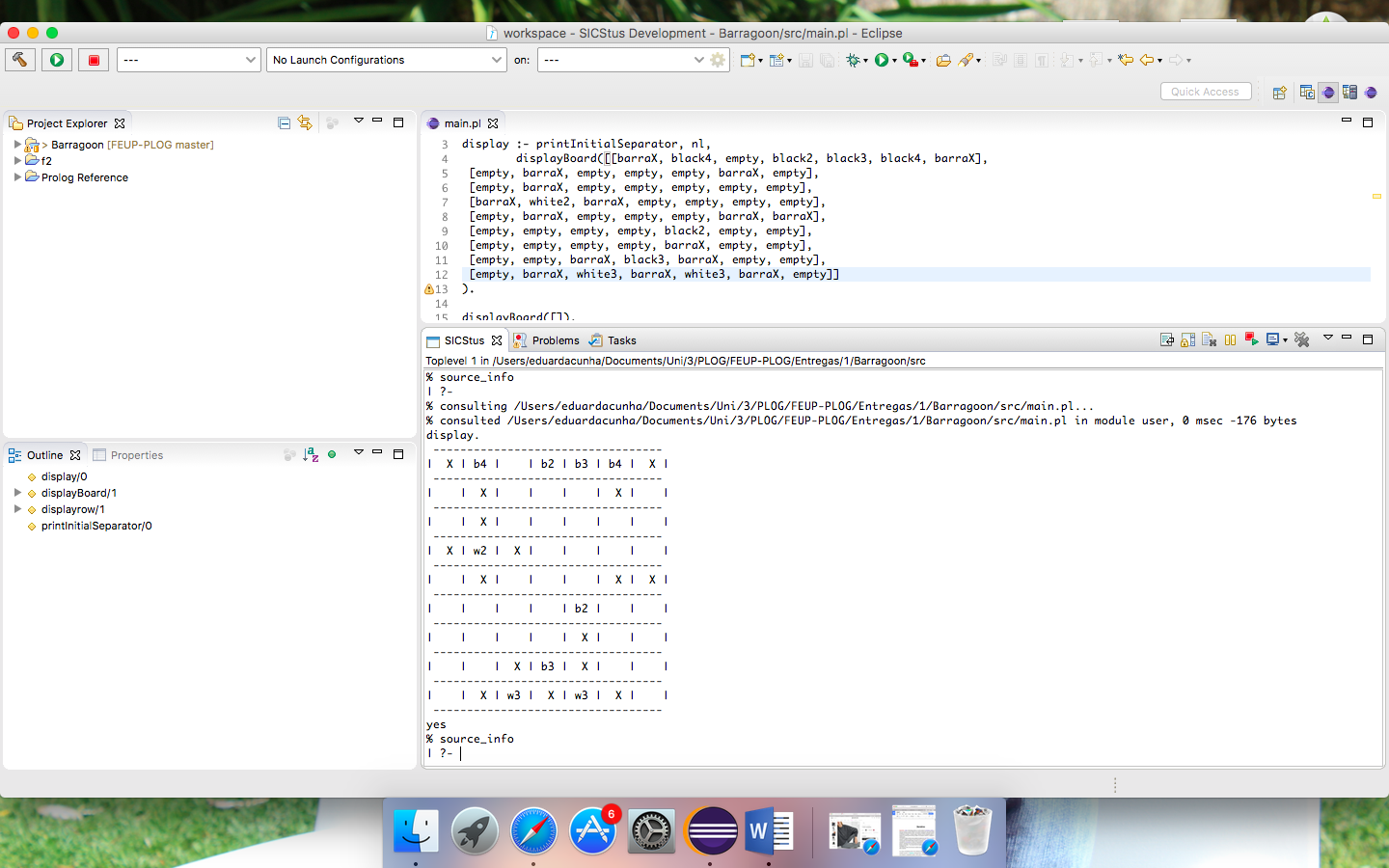
[empty, b-3, empty, allDir, empty, barraX, empty],

[empty, empty, empty, empty, right, empty, empty],

[empty, empty, w-2, w-3, empty, empty, empty],

 [empty, w-4, w-3, empty, w-3, w-4, empty]]

**Estado final:**

[[barraX, b-4, empty, b-2, b-3, b-4, barraX],

[empty, barraX, empty, empty, empty, barraX, empty],

[empty, barraX, empty, empty, empty, empty, empty],

[barraX, w-2, barraX, empty, empty, empty, empty],

[empty, barraX, empty, empty, empty, barraX, barraX],

[empty, empty, empty, empty, b-2, empty, empty],

[empty, empty, empty, empty, barraX, empty, empty],

[empty, empty, barraX, b-3, barraX, empty, empty],

 [empty, barraX, w-3, barraX, w-3, barraX, empty]]

## Visualização do Tabuleiro

1. initialBoard(
2. [[empty, b-*4*, b-*3*, empty, b-*3*, b-*4*, empty],
3. [empty, empty, b-*2*, b-*3*, b-*2*, empty, empty],
4. [empty, empty, empty, empty, empty, empty, empty],
5. [empty, bg-'barraX', empty, empty, empty, bg-'barraX', empty],
6. [bg-'barraX', empty, bg-'barraX', empty, bg-'barraX', empty, bg-'barraX'],
7. [empty, bg-'barraX', empty, empty, empty, bg-'barraX', empty],
8. [empty, empty, empty, empty, empty, empty, empty],
9. [empty, empty, w-*2*, w-*3*, w-*2*, empty, empty],
10. [empty, w-*4*, w-*3*, empty, w-*3*, w-*4*, empty]])**.**
11. displayGame(**Game**) **:-**
12. getBoard(Game, **Board**),
13. getCurrentPlayer(Game, **Player**),
14. clearScreen,
15. displayPlayerTurn(Player),
16. lettersAxis,nl,
17. horizontalBorder, nl,
18. numbersAxis(**RowNumbers**),
19. displayBoard(Board, RowNumbers), nl**.**
20. displayBoard([], [])**.**
21. displayBoard([**RowToDisplay**|**RemainingBoard**],
22. [**RowToDisplayNumber**|**RemainingRowNumbers**]) **:-**
23. translate([RowToDisplayNumber]),
24. translate(RowToDisplay),border, nl,
25. horizontalBorder, nl,
26. displayBoard(RemainingBoard, RemainingRowNumbers)**.**
27. *% -- Board Translation --*
28. translate([])**.**
29. translate(['empty'|**R**]) **:-** border, write(' '), !, translate(R)**.**
30. translate([w-*2*|**R**]) **:-** border, write(' w2 '), !, translate(R)**.**
31. translate([w-*3*|**R**]) **:-** border, write(' w3 '), !, translate(R)**.**
32. translate([w-*4*|**R**]) **:-** border, write(' w4 '), !, translate(R)**.**
33. translate([b-*2*|**R**]) **:-** border, write(' b2 '), !, translate(R)**.**
34. translate([b-*3*|**R**]) **:-** border, write(' b3 '), !, translate(R)**.**
35. translate([b-*4*|**R**]) **:-** border, write(' b4 '), !, translate(R)**.**
36. translate([bg-'barraX'|**R**]) **:-** border, write(' X '), !, translate(R)**.**
37. translate([bg-'allDir'|**R**]) **:-** border, write(' + '), !, translate(R)**.**
38. translate([bg-'oDirU'|**R**]) **:-** border, write(' V '), !, translate(R)**.**
39. translate([bg-'oDirD'|**R**]) **:-** border, write(' A '), !, translate(R)**.**
40. translate([bg-'oDirL'|**R**]) **:-** border, write(' <= '), !, translate(R)**.**
41. translate([bg-'oDirR'|**R**]) **:-** border, write(' => '), !, translate(R)**.**
42. translate([bg-'tDirH'|**R**]) **:-** border, write(' X '), !, translate(R)**.**
43. translate([bg-'tDirV'|**R**]) **:-** border, write(' X '), !, translate(R)**.**
44. translate([bg-'DtoR'|**R**]) **:-** border, write(' .> '), !, translate(R)**.**
45. translate([bg-'DtoL'|**R**]) **:-** border, write(' <. '), !, translate(R)**.**
46. translate([bg-'UtoR'|**R**]) **:-** border, write(' \'> '), !, translate(R)**.**
47. translate([bg-'UtoL'|**R**]) **:-** border, write(' <\' '), !, translate(R)
48. translate([bg-'LtoU'|**R**]) **:-** border, write(' -^ '), !, translate(R)**.**
49. translate([bg-'LtoD'|**R**]) **:-** border, write(' -v '), !, translate(R)**.**
50. translate([bg-'RtoU'|**R**]) **:-** border, write(' ^- '), !, translate(R)**.**
51. translate([bg-'RtoD'|**R**]) **:-** border, write(' v- '), !, translate(R)**.**
52. translate(['um'|**R**]) **:-** write('1'), !, translate(R)**.**
53. translate(['dois'|**R**]) **:-** write('2'), !, translate(R)**.**
54. translate(['tres'|**R**]) **:-** write('3'), !, translate(R)**.**
55. translate(['quatro'|**R**]) **:-** write('4'), !, translate(R)**.**
56. translate(['cinco'|**R**]) **:-** write('5'), !, translate(R)**.**
57. translate(['seis'|**R**]) **:-** write('6'), !, translate(R)**.**
58. translate(['sete'|**R**]) **:-** write('7'), !, translate(R)**.**
59. translate(['oito'|**R**]) **:-** write('8'), !, translate(R)**.**
60. translate(['nove'|**R**]) **:-** write('9'), !, translate(R)**.**
61. *% -- Board Axis –*
62. numbersAxis([um, dois, tres, quatro, cinco, seis, sete, oito, nove])**.**
63. lettersAxis **:-** write(' A B C D E F G')**.**
64. *% -- Board Borders –*
65. horizontalBorder **:-** write(' ----------------------------------')**.**
66. border **:-** write('|')**.**

## Lista de Jogadas Válidas

O predicado **getMovesAvailable(+Game, +Row, +Column, -List)** retorna uma lista com as várias opções de percurso que uma peça pode percorrer, já validadas.

## Execução de Jogadas

O ciclo principal **playGame(+Game, -NewGame)** é caracterizado pela execução da jogada **playerTurn(+Game, -UpdatedGame)**, responsável por efetuar todos os pontos relativos à jogada de um jogador, **switchPlayer(+UpdatedGame, -NextPlayerGame)**, que trata de alternar os jogadores a cada jogada, e a chamada recursiva de **playGame**.

O predicado **playerTurn** é constituído por **playerMove(+Game, +RolSrc, +ColSrc, -Path)** ou **botMove(+Game, -Row, -Column, -Path)** dependendo do modo jogo que se trata, **validateMove(+Game, +RowSrc, +ColSrc, +Path, -PieceCaptured)**, **movePiece(+Game, +RowSrc, +ColSrc, +Path, -NewGame1)** e uma verificação de se alguma peça, telha ou barragoon, foi capturada.

Em **playerMove**, é pedido ao jogador que escolha a telha que pretende mover, através da inserção na consola da linha, representada por um número de 1 a 9, e da coluna, representada por uma letra de A a G (**chooseTile()**), é validado que essa telha existe e lhe pertence (**validateTile()**), pede-se o percurso que a peça vai efetuar, representado pelas teclas WASD (**choosePath()**) e, por fim, verifica-se se esse caminho não ultrapassa os limites do tabuleiro (**validatePath()**).

O predicado **botMove**, que existe em paralelo com **playerMove**, ao invés de receber do jogador as coordenadas e caminho, gera-os aleatoriamente a partir de listas de posições e percursos que já foram selecionados por serem válidos. Isto torna a função seguinte, **validateMove**, algo desnecessária para esta jogada. Com algum tempo, podia haver alguma reestruturação de código de forma a eliminar esta redundância.

Em **validateMove**, trata-se de todas as questões relacionadas com o movimento da peça respeitar as regras do jogo. Verificando através de **getCell(+Board, +RowSrc, +ColSrc, -Piece)** qual é a telha que está a ser movida, com recurso a **isShortMove(+Piece, +Path)** e **isFullMove(+Piece, +Path)** determinamos se o percurso inserido pelo jogador é válido. Ainda, em **validateCrossMovements(+Game, +RowSrc, +ColSrc, +Path, +IsLongMove, +Piece, -PieceCaptured)** calcula-se se o movimento em questão obedece às regras de passagem em cima de barragoons e se não passa por cima de telhas se não for com o objetivo de as comer.

Em **movePiece**, é calculada a célula final em que a telha deve acabar através de **getDestCellFromPath(+RowSrc, +ColSrc, +Path, -RowDest, -ColDest)**, elimina-se o conteúdo dessa posição e procede-se à colocação da telha no novo destino com **moveFromSrcToDest(+Game, +RowSrc, +ColSrc, +RowDest, +ColDest, -NewGame)**.

## Avaliação do Tabuleiro

Avaliação do estado do jogo, que permitirá comparar a aplicação das diversas jogadas disponíveis.

## Final do Jogo

O jogo termina quando um dos jogadores já não tem telhas ou as que possui não se conseguem deslocar.

De forma a lidar com o primeiro ponto, utilizamos o predicado **countPlayerPieces(+Board, +CurrentPlayer, -CountPieces)** que verifica no tabuleiro o número de telhas do jogador cuja vez for de jogar. Se **CountPieces** retornar a 0, o jogo acaba e esse jogador perde.

Para verificar se, existindo peças, estas possuem movimentos possíveis, recorremos a **countMovesAvailable(+Game, +Row, +Column, -Count)** para contar os movimentos de uma peça. Se **Count** retornar a 0 para todas as peças em tabuleiro do jogador, o jogo acaba e esse jogador perde. Esta solução é um pouco *hardcoded*, pois recorremos a todos os percursos possíveis para cada telha para efetuar esta verificação.

## Jogada do Computador

Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade.

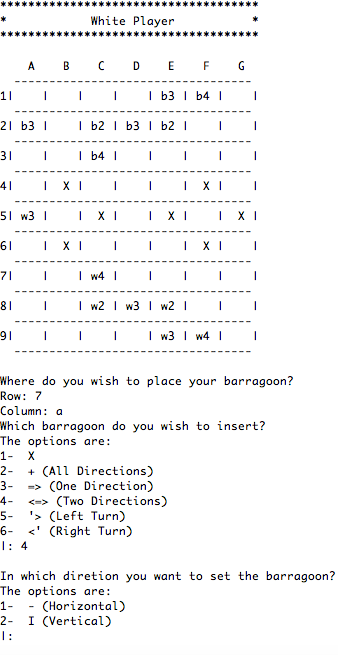
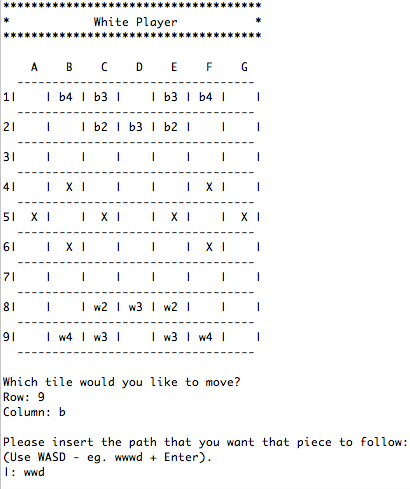
# Interface com o Utilizador

Quando iniciado com o comando *start*, é apresentado o menu principal com as opções de jogar em modo humano contra humano, humano contra computador e computador contra computador, apresentar as regras ou sair do jogo.

Se for escolhida a opção 4 - Regras, é apresentado o seguinte menu que enuncia as regras relevantes para o jogador.

Iniciado o jogo em modo humano contra humano, as jogadas apresentam-se com um cabeçalho que indica a vez do jogador – White Player ou Black Player, o tabuleiro de jogo com eixos de números para as linhas e letras para as colunas e, por fim, perguntas relativas ao *input* do utilizador, como coordenadas da peça a mover e caminho que a peça deve seguir. Caso alguma destas informações seja inserida de forma errada, é sempre apresentada uma mensagem de erro e é pedido de novo ao jogador que insira a informação em questão até que a insira corretamente.

Se for o caso de inserir um barragoon no tabuleiro, é pedido ao respetivo jogador que insira a informação relativa a essa colocação (coordenadas de destino do barragoon e a face que se pretende que fique voltada para cima).



# Conclusões

Concluído o projeto, achamos importante ressaltar aquilo que poderia ter sido melhorado e não foi apenas pela ausência de tempo. Destaca-se o absentismo dos 2 níveis mínimos esperados para os modos de jogo humano contra computador e computador contra computador e a existência de soluções ditas *hardcoded* (por exemplo, o cálculo do fim do jogo através da verificação de todos os paths possíveis).

Ainda, pensamos que a nossa implementação é bastante intuitiva e de fácil compreensão, mas a sua complexidade poderia ter sido sensivelmente reduzida com alguma reutilização de código e *layering*.

Concluindo, consideramos que realizamos com sucesso aquilo a que nos propusemos e com o tempo que nos foi dado. Os conceitos das aulas foram, sem dúvida, consolidados e temos agora uma nova compreensão da utilidade desta linguagem de programação.

# Bibliografia

1. https://www.youtube.com/watch?v=qG1i0\_sn\_FI
2. https://boardgamegeek.com/boardgame/157779/barragooN
3. http://www.barragoon.de/bsp/BARRAGOON\_en.pdf
4. https://stackoverflow.com
5. http://www.swi-prolog.org

# Anexos

main.pl

1. **:-** include('Utilities.pl')**.**
2. **:-** include('Interface.pl')**.**
3. **:-** include('Logic.pl')**.**
4. **:-** use\_module(library(system))**.**
5. *%--------------------------------%*
6. *%-----------Barragoon -----------%*
7. *%--------------------------------%*
8. *%--------- escreva start --------%*
9. *%---- na consola para correr ----%*
10. *%--------------------------------%*
11. *%--------------------------------%*
12. *% --- START ---*
13. start **:-**
14. clearScreen,
15. mainMenu**.**

Interface.pl

1. *%-------------------------------%*
2. *%----------Interface------------%*
3. *%-------------------------------%*
4. *% -------------------------------------------------------------------------*
5. *% ----------------------------------- MENUS -------------------------------*
6. *% -------------------------------------------------------------------------*
7. *% -- Menus --*
8. mainMenu **:-**
9. displayMainMenu,
10. getCharThenEnter(**Option**),
11. (
12. Option = '1' -> **startGamePvP**;
13. Option = '2';
14. Option = '3';
15. Option = '4' -> displayRules;
16. Option = '5';
17. clearScreen,
18. write('ERROR : invalid input...'), spacing(*1*),
19. mainMenu
20. )**.**
21. displayMainMenu **:-**
22. upperFrame,
23. titleFrame,
24. write('| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |'),nl,
25. write('| \* Main Menu \* |'),nl,
26. write('| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |'),nl,
27. write('| Play: |'),nl,
28. write('| 1 - Player vs Player |'),nl,
29. write('| 2 - Player vs Computer |'),nl,
30. write('| 3 - Computer vs Computer |'),nl,
31. write('| |'),nl,
32. write('| 4 - Rules |'),nl,
33. write('| 5 - Quit |'),nl,
34. lowerFrame**.**
35. displayRules **:-**
36. clearScreen,
37. upperFrame,
38. titleFrame,
39. write('| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |'),nl,
40. write('| \* Game Rules \* |'),nl,
41. write('| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |'),nl,
42. write('| 1. A piece is captured if the oponent\'s piece finishes |'),nl,
43. write('| in the same cell. |'),nl,
44. write('| 2. Captures are only allowed during a full move. |'),nl,
45. write('| 3. Tiles with 2 circles can\'t capture barragoons with |'),nl,
46. write('| the "all directions" symbol turned up. |'),nl,
47. write('| 4. If a barragoon is captured, it must be placed back |'),nl,
48. write('| in the board, in a free place, with whichever symbol |'),nl,
49. write('| the player preffers. |'),nl,
50. write('| 5. When a tile is captured, 2 barragoons are added to |'),nl,
51. write('| the board, 1 per player and the first to place it is |'),nl,
52. write('| the one whose tile was captured. |'),nl,
53. write('| 6. Once a barragoon is placed, it can\'t be moved. |'),nl,
54. write('| 7. During a move, there can only be one change of |'),nl,
55. write('| direction. |'),nl,
56. write('| 8. Movements can only be vertical or horizontal, never |'),nl,
57. write('| diagonal. |'),nl,
58. write('| 9. During a move, there can only be one 90º turn. |'),nl,
59. lowerFrame,
60. spacing(*2*),
61. next, mainMenu, !**.**
62. displayPlayerTurn(**Player**) **:-**
63. ifelse(
64. Player = w ,
65. (
66. write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,
67. write('\* White Player \*'),nl,
68. write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,nl
69. ),
70. (
71. write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,
72. write('\* Black Player \*'),nl,
73. write('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'),nl,nl
74. ))**.**
75. *% -- Logo --*
76. titleFrame **:-**
77. write('| \_\_\_\_ |'), nl,
78. write('| | \_ \\ |'), nl,
79. write('| | |\_) | \_\_ \_ \_ \_\_ \_ \_\_ \_\_ \_ \_\_ \_ \_\_\_ \_\_\_ \_ \_\_ |'), nl,
80. write('| | \_ < / \_` | \'\_\_| \'\_\_/ \_` |/ \_` |/ \_ \\ / \_ \\| \'\_ \\ |'), nl,
81. write('| | |\_) | (\_| | | | | | (\_| | (\_| | (\_) | (\_) | | | | |'), nl,
82. write('| |\_\_\_\_/ \\\_\_,\_|\_| |\_| \\\_\_,\_|\\\_\_, |\\\_\_\_/ \\\_\_\_/|\_| |\_| |'), nl,
83. write('| \_\_/ | |'), nl,
84. write('| |\_\_\_/ |'), nl,
85. write('| |'),nl**.**
86. *% -- Frames --*
87. upperFrame **:-**
88. write(' \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'),nl,
89. write('| |'),nl**.**
90. lowerFrame **:-**
91. write('| |'),nl,
92. write('|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|'),nl**.**
93. *% -------------------------------------------------------------------------*
94. *% ----------------------------------- BOARD -------------------------------*
95. *% -------------------------------------------------------------------------*
96. *% -- BOARD --*
97. initialBoard( [[empty, b-*4*, b-*3*, empty, b-*3*, b-*4*, empty],
98. [empty, empty, b-*2*, b-*3*, b-*2*, empty, empty],
99. [empty, empty, empty, empty, empty, empty, empty],
100. [empty, bg-barraX, empty, empty, empty, bg-barraX, empty],
101. [bg-barraX, empty, bg-barraX, empty, bg-barraX, empty, bg-barraX],
102. [empty, bg-barraX, empty, empty, empty, bg-barraX, empty],
103. [empty, empty, empty, empty, empty, empty, empty],
104. [empty, empty, w-*2*, w-*3*, w-*2*, empty, empty],
105. [empty, w-*4*, w-*3*, empty, w-*3*, w-*4*, empty]])**.**
106. displayGame(**Game**) **:-**
107. getBoard(Game, **Board**),
108. getCurrentPlayer(Game, **Player**),
109. clearScreen,
110. displayPlayerTurn(Player),
111. lettersAxis,nl,
112. horizontalBorder, nl,
113. numbersAxis(**RowNumbers**),
114. displayBoard(Board, RowNumbers), nl**.**
115. displayBoard([], [])**.**
116. displayBoard([**RowToDisplay**|**RemainingBoard**],
117. [**RowToDisplayNumber**|**RemainingRowNumbers**]) **:-**
118. translate([RowToDisplayNumber]),
119. translate(RowToDisplay),border, nl,
120. horizontalBorder, nl,
121. displayBoard(RemainingBoard, RemainingRowNumbers)**.**
122. *% -- Board Translation --*
123. translate([])**.**
124. translate(['empty'|**R**]) **:-** border, write(' '), !, translate(R)**.**
125. translate([w-*2*|**R**]) **:-** border, write(' w2 '), !, translate(R)**.**
126. translate([w-*3*|**R**]) **:-** border, write(' w3 '), !, translate(R)**.**
127. translate([w-*4*|**R**]) **:-** border, write(' w4 '), !, translate(R)**.**
128. translate([b-*2*|**R**]) **:-** border, write(' b2 '), !, translate(R)**.**
129. translate([b-*3*|**R**]) **:-** border, write(' b3 '), !, translate(R)**.**
130. translate([b-*4*|**R**]) **:-** border, write(' b4 '), !, translate(R)**.**
131. translate([bg-'barraX'|**R**]) **:-** border, write(' X '), !, translate(R)**.**
132. translate([bg-'allDir'|**R**]) **:-** border, write(' + '), !, translate(R)**.**
133. translate([bg-'oDirU'|**R**]) **:-** border, write(' A '), !, translate(R)**.**
134. translate([bg-'oDirD'|**R**]) **:-** border, write(' V '), !, translate(R)**.**
135. translate([bg-'oDirL'|**R**]) **:-** border, write(' <= '), !, translate(R)**.**
136. translate([bg-'oDirR'|**R**]) **:-** border, write(' => '), !, translate(R)**.**
137. translate([bg-'tDirH'|**R**]) **:-** border, write(' - '), !, translate(R)**.**
138. translate([bg-'tDirV'|**R**]) **:-** border, write(' I '), !, translate(R)**.**
139. translate([bg-'DtoR'|**R**]) **:-** border, write(' .> '), !, translate(R)**.**
140. translate([bg-'DtoL'|**R**]) **:-** border, write(' <. '), !, translate(R)**.**
141. translate([bg-'UtoR'|**R**]) **:-** border, write(' \'> '), !, translate(R)**.**
142. translate([bg-'UtoL'|**R**]) **:-** border, write(' <\' '), !, translate(R)**.**
143. translate([bg-'LtoU'|**R**]) **:-** border, write(' -^ '), !, translate(R)**.**
144. translate([bg-'LtoD'|**R**]) **:-** border, write(' -v '), !, translate(R)**.**
145. translate([bg-'RtoU'|**R**]) **:-** border, write(' ^- '), !, translate(R)**.**
146. translate([bg-'RtoD'|**R**]) **:-** border, write(' v- '), !, translate(R)**.**
147. translate(['1'|**R**]) **:-** write('1'), !, translate(R)**.**
148. translate(['2'|**R**]) **:-** write('2'), !, translate(R)**.**
149. translate(['3'|**R**]) **:-** write('3'), !, translate(R)**.**
150. translate(['4'|**R**]) **:-** write('4'), !, translate(R)**.**
151. translate(['5'|**R**]) **:-** write('5'), !, translate(R)**.**
152. translate(['6'|**R**]) **:-** write('6'), !, translate(R)**.**
153. translate(['7'|**R**]) **:-** write('7'), !, translate(R)**.**
154. translate(['8'|**R**]) **:-** write('8'), !, translate(R)**.**
155. translate(['9'|**R**]) **:-** write('9'), !, translate(R)**.**
156. *% -- Board Axis --*
157. numbersAxis(['1', '2', '3', '4', '5', '6', '7', '8', '9'])**.**
158. lettersAxis **:-** write(' A B C D E F G')**.**
159. *% -- Board Borders --*
160. horizontalBorder **:-** write(' ----------------------------------')**.**
161. border **:-** write('|')**.**

Logic.pl

Utilities.pl