U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Fabrik

Relatório Final

Programação em Lógica

Grupo: Fabrik_2

(12 de novembro de 2017)

Bárbara Sofia Lopez de Carvalho Ferreira da Silva **up201505628**@fe.up.pt

Julieta Pintado Jorge Frade **up201506530**@fe.up.pt

Resumo

Este trabalho consiste na conceção de um jogo de tabuleiro utilizando uma linguagem de programação em lógica denominada *Prolog*.

Fabrik foi o jogo escolhido e é destinado a dois jogadores, baseando-se no famoso jogo *Cinco em Linha*. Este jogo torna-se mais complexo pois são adicionadas mais restrições de jogada ao jogo em que se baseou. Note-se que foram implementados três modos de utilização perfeitamente funcionais: Humano/Humano, Humano/Computador e Computador/Computador. Nestes três modos todas as regras do jogo foram implementadas com sucesso.

Este trabalho permitiu a consolidação dos conhecimentos adquiridos nas aulas tanto teóricas como práticas da cadeira de Programação Lógica e confirmar o quão eficiente é usar a linguagem de *Prolog* para resolver problemas de decisão.

Inicialmente, o principal obstáculo a ultrapassar foi o facto de nunca se ter tido contacto com esta linguagem, a adaptação à mesma demorou um pouco, mas à medida que o projeto foi sendo desenvolvido, as dúvidas foram esclarecidas.

Assim, foi concebido com sucesso um jogo simples, intuitivo e de fácil interação com o utilizador, com três modos à escolha.

Índice

Introdução

Este projeto foi desenvolvido no Sistema de Desenvolvimento *SICStus Prolog* no âmbito da unidade curricular de Programação Lógica de 3º ano do curso Mestrado Integrado em Engenharia Informática e de Computação e tem como tema o jogo de tabuleiro *Fabrik*. O objetivo deste trabalho foi implementar, em linguagem *Prolog*, um jogo de tabuleiro e de peças, pelas regras de movimentação das peças (jogadas possíveis) e pelas condições de terminação do jogo com derrota, vitória ou empate.

Este relatório tem a seguinte estrutura:

- **O Jogo Fabrik:** Descrição do jogo e das suas regras.
- **Lógica do Jogo:** Descrição da implementação da lógica do jogo, tendo a seguinte estrutura:
  - **Representação do Estado do Jogo:** Exemplificação de estados iniciais, intermédios e finais do jogo.
  - **Visualização do Tabuleiro:** Descrição do predicado de visualização.
  - **Lista de Jogadas Válidas**: Descrição dos predicados usadas para a validação das jogadas.
  - **Execução de Jogadas**: Explicitação do ciclo do jogo e de como é analisada cada jogada.
  - **Avaliação do Tabuleiro:** Descrição dos predicados que retornam o conteúdo do tabuleiro.
  - **Final do Jogo:** Descrição dos predicados que verificam o fim de jogo.
  - **Jogada do Computador:** Descrição dos predicados de geração de movimentos do computador.
- **Interface com o Utilizador:** Descrição do módulo de interface com o utilizador.

O Jogo Fabrik

Fabrik é um jogo de tabuleiro criado em agosto de 2017. Consiste no conceito de duas figuras neutras, denominadas por *worker* ou *arbeiter*, estas são acessíveis aos dois jogadores, que em colaboração determinam os espaços em que os mesmos podem jogar, ou seja, onde podem deixar a sua peça em cada ronda.

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. Na verdade, as regras de colocação restrita no *Fabrik* ajudam a superar a vantagem do primeiro jogador, que existem em muitos outros jogos, como *Gomoku* e, assim, *Fabrik* está de certa forma relacionado com *Renju*.

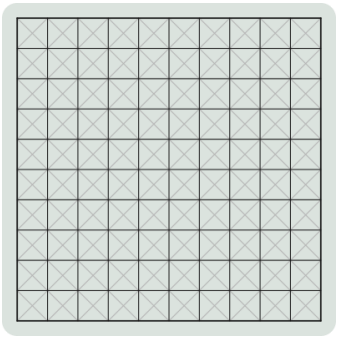


Figura 1: tabuleiro do jogo.

O material necessário para o jogo é um tabuleiro quadrado com 11x11 espaços, uma grande quantidade de peças brancas e pretas, e duas peças vermelhas, chamadas *workers*.

Preparação

Inicialmente, o tabuleiro está vazio. O jogador das peças pretas começa por colocar um dos *workers* em qualquer espaço. De seguida, o jogador das peças brancas coloca o outro *worker* num espaço ainda livre.

O jogador das peças pretas decide quem joga primeiro. Este deverá colocar uma peça da sua cor de acordo com as regras descritas mais abaixo. Após o jogo estar preparado, os jogadores deverão alternar entre si.

Objetivo

Os jogadores ganham assim que um deles conseguir obter uma linha de, pelo menos, 5 peças da sua cor seguidas, na ortogonal ou diagonal.

Desenvolvimento

Em cada ronda o jogador poderá mover um dos *workers* e colocá-lo num outro espaço vazio, este passo é opcional. Depois, deverá colocar uma das suas peças em qualquer linha de interseção de um dos *workers*, chamadas **linhas de vista**. Estas linhas radiam da posição do *worker* numa direção ortogonal e diagonal, enquanto existem espaços vazios. Assim que uma linha de vista alcançar uma peça, esta acaba nessa posição.

Em certos casos, é possível que os *workers* fiquem localizados na mesma linha ortogonal ou diagonal, assim, todos os espaços entre eles são considerados pontos de interseção, desde que estejam vazios.

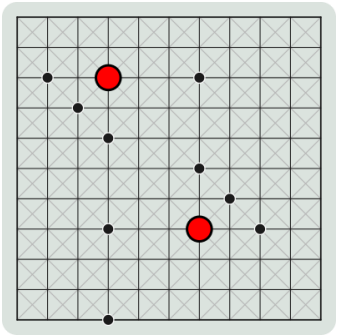


Figura 2: os pontos de interseção das linhas de visão dos *workers* determinam onde as peças podem ser colocadas.

Fim

O jogador pede o jogo assim que não consiga colocar nenhum dos dois *workers* numa posição em que seja possível inserir uma peça nova.

Assim, ganha o jogo aquele que conseguir colocar, pelo menos, 5 peças da sua cor seguidas numa direção ortogonal ou diagonal.

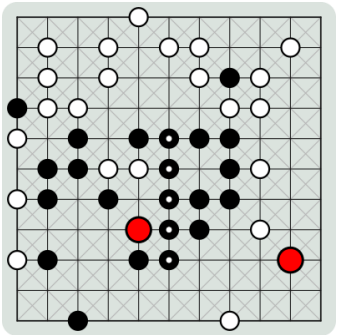


Figura 3: fim de jogo em que o jogador com as peças pretas ganha.

Lógica do Jogo

Representação do Estado do Jogo

Situação Inicial

```
initialBoard([
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]
]).
```

Situação Intermédia

```
midBoard([
[empty,empty,empty,empty,white,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,red,empty],
[empty,white,empty,empty,empty,empty,empty,empty,white,empty,empty],
[empty,empty,empty,empty,empty,white,empty,empty,white,empty,empty],
[empty,empty,empty,black,empty,black,black,black,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,white,empty,empty],
[empty,empty,empty,black,empty,white,empty,empty,empty,empty,empty],
[empty,empty,empty,black,empty,red,black,empty,empty,empty,empty],
[empty,black,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,black,empty,empty,empty,empty,empty,white,empty,empty]
]).
```

Situação Final

```
finalBoard([
[empty,empty,empty,empty,white,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,white,empty,empty,empty,empty,empty,empty],
[empty,white,empty,empty,empty,empty,empty,empty,white,empty,empty],
[empty,empty,empty,black,empty,white,empty,empty,white,empty,empty],
[empty,empty,empty,black,empty,black,black,black,empty,empty,empty],
[empty,empty,empty,black,empty,empty,empty,empty,white,empty,empty],
[empty,empty,empty,black,empty,white,empty,empty,empty,empty,empty],
[empty,empty,red,black,empty,empty,black,empty,white,empty,empty],
[white,black,empty,empty,empty,empty,empty,empty,empty,red,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,black,empty,empty,empty,empty,empty,white,empty,empty]
]).
```

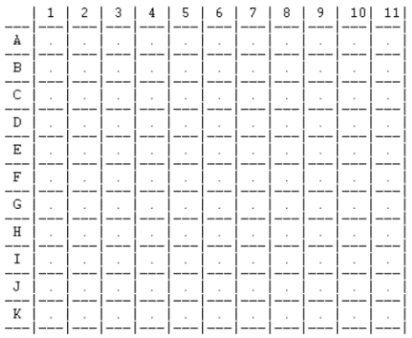


Figura 4: situação inicial vista na consola.

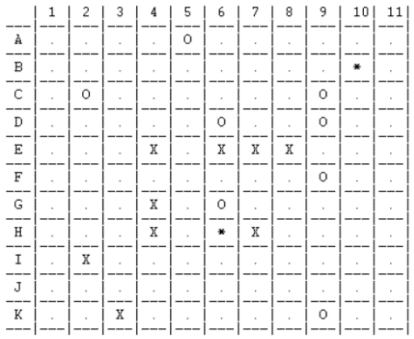


Figura 5: situação intermédia vista na consola.

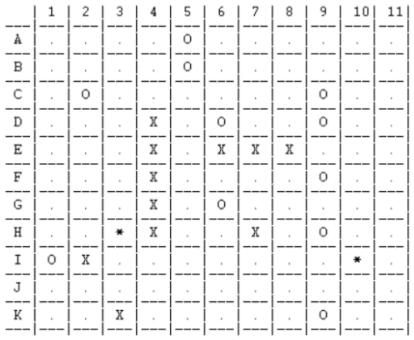


Figura 6: situação final vista na consola.

Visualização do Tabuleiro

Segue-se o código que será utilizado para mostrar o tabuleiro na consola:

```
printBoard(X) :-
    nl,
    write('   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11|\n'),
    write('---|---|---|---|---|---|---|---|---|---|---|---|\n'),
    printMatrix(X, 1).

printMatrix([], 12).

printMatrix([Head|Tail], N) :-
    letter(N, L),
    write(' '),
    write(L),
    N1 is N + 1,
    write(' | '),
    printLine(Head),
    write('\n---|---|---|---|---|---|---|---|---|---|---|---|\n'),
    printMatrix(Tail, N1).

printLine([]).

printLine([Head|Tail]) :-
    symbol(Head, S),
    write(S),
    write(' | '),
    printLine(Tail).
```

O output produzido está ilustrado nas imagens da página anterior.

Lista de Jogadas Válidas

Uma posição é considerada válida para os jogadores (X/O) se a célula estiver vazia e se estiver na linha de visão de pelo menos um *worker*, isto é, a célula tem que estar num dos 8 sentidos (**N, S, E, O, NE, NO, SE, SO**) em relação ao *worker* e entre essa célula e o *worker* não existir nenhuma peça. Na imagem à direita podemos ver um exemplo de tabuleiro onde as linhas azuis são as diferentes linhas provenientes do *worker* e a rosa as linhas de visão destes. Uma posição é considerada válida para o movimento dos *workers* se a célula se encontrar vazia.

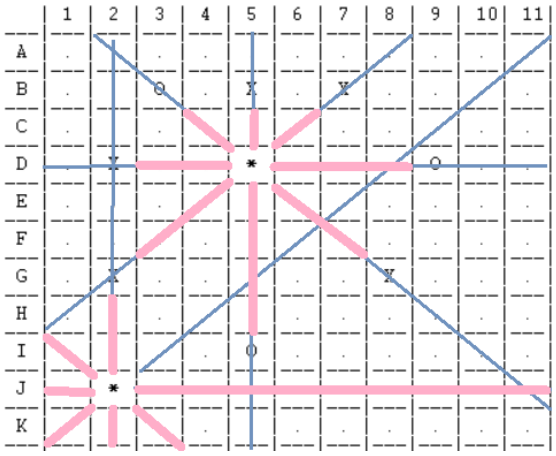


Figura 7: linhas de visão dos *workers*.

Para a validação das jogadas foram usados os seguintes predicados que se encontram no ficheiro *logic.pl*:

```
checkMove(Board, Player, NewBoard, Expected, ColumnIndex, RowIndex)
isEmptyCell(Board, Row, Column, Res)
isValidPosLines(Board, Row, Column, Res)
isWorkerLines(Board, WorkerRow, WorkerColumn, Row, Column, Res)
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResN, 'N' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResNE, 'NE' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResE, 'E' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResSE, 'SE' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResS, 'S' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResSO, 'SO' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResO, 'O' )
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, 1, ResNO, 'NO' )
```

checkMove – É chamado para verificar todas as jogadas (tanto do jogador, como do worker) recorrendo ao predicado **isValidPosLines** e getValueFromMatrix.

Este ultimo serve para verificar se naquela célula do tabuleiro está o conteúdo pretendido (*Expected*). Por exemplo, se for para adicionar uma peça, o conteúdo pretendido é *'empty'*, mas caso seja para escolher o *worker* a mover, o conteúdo pretendido vai ser '*red'*. Se não for possível fazer o movimento, este predicado chama o *write* que informa o jogador qual a razão da falha, pedindo umas novas coordenadas.

isEmptyCell – Verifica se a célula (*Row, Column)* está vazia recorrendo a chamadas ao predicado **getValueFromMatrix**.

isValidPosLines – Vai buscar as posições dos *workers* com a ajuda do predicado **getWorkersPos** e verifica se a célula (*Row, Column*) está na linha de visão de pelo menos um dos *workers*.

isWorkerLines – Verifica se a célula (*Row, Column*) está nalguma das linhas de visão do *worker* (*WorkerRow, WorkerColumn*) com a ajuda do predicado *verifyLine*.

verifyLine – Verifica se a célula está na linha de visão do *worker*. Foi feito o *overload* deste predicado para cada um dos sentidos com o intuito do código se tornar mais percetível.

Execução de Jogadas

O jogo tem um ciclo principal, denominado **gameLoop**, que está encarregue de executar as jogadas de cada jogador e a verificação do estado do jogo.

```
gameLoop(Board, Player1, Player2) :-
      blackPlayerTurn(Board, NewBoard, Player1),
      ((checkGameState('black', NewBoard), write('\nThanks for playing!\n'));
       (whitePlayerTurn(NewBoard, FinalBoard, Player2),
       ((checkGameState('white', FinalBoard), write('\nThanks for playing!\n'));
        (gameLoop(FinalBoard, Player1, Player2))))).
```

Relativamente a cada jogada, o jogador decide primeiro se pretende ou não mover o *worker* para uma outra posição, caso o deseje fazer, irá passar por dois processos de validação. Inicialmente, são verificadas se as coordenadas inseridas pelo utilizador ou pelo computador correspondem a um *worker*, assim que esta condição for satisfeita, serão analisadas as novas coordenadas inseridas, ou seja, a nova posição do *worker*. Assim que este processo tiver completo, o jogador poderá então escolher onde vai querer colocar a sua peça.

```
blackPlayerTurn(Board, NewBoard, 'P') :-
      write('\n------------------ PLAYER X ------------------\n\n'),
      write('1. Do you want to move a worker? [0(No)/1(Yes)]'),
      manageMoveWorkerBool(MoveWorkerBoolX),
      moveWorker(Board, MoveWorkerBoolX, Board1),
      askCoords(Board1, black, NewBoard, empty),
      printBoard(NewBoard).
```

O predicado responsável pela execução e validação da jogada de uma peça é o **askCoords**, este começa por receber uma linha e uma coluna, e assim que recebe, analisa se é válida, dentro das proporções do tabuleiro. Se tal se comprovar, verifica então se a jogada é válida, tendo em conta as regras do jogo, com o predicado **checkMove**. Este último tema está explicado detalhadamente no tópico **Lista de Jogadas Válidas**.

```
askCoords(Board, Player, NewBoard, Expected) :-
      manageRow(NewRow),
      manageColumn(NewColumn),
      write('\n'),
      ColumnIndex is NewColumn - 1,
      RowIndex is NewRow - 1,
      checkMove(Board, Player, NewBoard, Expected, ColumnIndex, RowIndex).
```

Avaliação do Tabuleiro

Os predicados mais relevantes de avaliação do tabuleiro estão no ficheiro *utilities.pl* e são eles:

```
getWorkersPos(Board, WorkerRow1, WorkerColumn1, WorkerRow2, WorkerColumn2)
getValueFromMatrix([_H|T], Row, Column, Value)
checkFullBoard(Board)
```

getWorkerPos – Percorre o tabuleiro e com a ajuda do predicado **getValueFromMatrix**, devolve em *WorkerRow1*, *WorkerColumn1*, *WorkerRow2* e *WorkerColumn2* as posições dos workers na matriz.

getValueFromMatrix – Analisa o que está na célula (*Row, Column*) da matriz. Retorna em *Value* o conteúdo daquela célula, ou, caso *Value* já esteja atribuído, a função falha.

checkFullBoard – Verifica se o tabuleiro está cheio, confirmando se não há nenhuma célula '*empty*' no tabuleiro.

Final do Jogo

Após cada jogada é fundamental verificar o estado do jogo, pois, a qualquer momento, um dos jogadores pode ganhar ou ocorrer um empate, isto é, caso não exista mais nenhum espaço válido para colocar uma nova peça ou caso não exista um espaço livre no tabuleiro. De forma a poder verificar todos estes casos, foi implementado o predicado **checkGameState**, que recebe o tipo de peça do jogador que acabou de jogar e o tabuleiro atual. Este predicado vai chamar 6 outros predicados, e caso algum deles se verifique, o jogo acaba.

```
checkGameState(Player, Board) :-
      ((checkVictory(Player, 'Row', Board), write('You won!'));
       (checkVictory(Player, 'Column', Board), write('You won!'));
       (checkVictory(Player, 'DiagonalDown', Board), write('You won!'));
       (checkVictory(Player, 'DiagonalUp', Board), write('You won!'));
       (checkFullBoard(Board), write('Woops, no more space left! It is a draw!'));
       (checkValidSpots(Board, 0, 0, Result), Result =:= 0, write('Woops, no more
space left! It is a draw!'))).
```

Como o nome indica, o predicado **checkVictory** verifica se o jogador ganhou o jogo. Em particular, existem 4 condições vencedoras: o jogador ter 5 peças seguidas na mesma linha, coluna ou diagonal. Deste modo, cada predicado verifica se existe esse padrão no tabuleiro.

Quanto ao predicado **checkFullBoard**, este verifica se não existem mais espaços livres, *empty*, em todo o tabuleiro. Já o predicado **checkValidSpots** verifica se não existe nenhum espaço válido, ou seja, dentro das linhas de visão de cada *worker*, para o próximo jogador colocar a sua peça. Portanto, se algum destes dois predicados se satisfizer, o jogo acaba com empate.

Jogada do Computador

Os predicados **startGame** e **gameLoop** recebem dois átomos *Player1* e *Player2*, que vão ser passados aos predicados **blackPlayerTurn** e **whitePlayerTurn.** Estes átomos podem ser *'P'* (*player*) ou *'C'* (*computer*). Assim é possível saber no ficheiro logic.pl, se se deve chamar os predicados relacionados com o computador.

Para a geração de jogadas do computador foram usados os seguintes predicados que se encontram no ficheiro *bot.pl*:

```
generatePlayerMove(Board, Row, Column)
moveWorker(Board, WorkerRow, WorkerColumn, WorkerNewRow, WorkerNewColumn, Res)
chooseWorker(Board, WorkerRow, WorkerColumn)
generateWorkerMove(Board,WorkerNewRow, WorkerNewColumn)
```

generatePlayerMove – Gera um linha e coluna aleatória, verifica se é uma jogada válida, ou seja, se é uma célula que está nas linhas de visão de algum *worker* e se a célula está atualmente vazia. Caso seja válida, ele devolve essa linha e coluna, caso contrário este predicado chama-se a si própria para tentar gerar uma nova posição.

moveWorkerPos – Escolhe aleatoriamente se vai mover ou não o *worker*. Se escolher mover o *worker*, chama o predicado **chooseWorker** para escolher aleatoriamente o *worker* a mover e seguidamente chama a função **generateWorkerMove** para saber a posição para qual mover.

chooseWorker – Escolhe aleatoriamente o *worker* a mover.

generateWorkerMove – Tal como o **generatePlayerMove,** gera uma linha e uma coluna e verifica se é valida. Neste caso é só verificar se a célula está vazia. Caso seja válida, ele devolve essa linha e coluna, caso contrário este predicado chama-se a si própria para tentar gerar uma nova posição.

Interface com o Utilizador

Relativamente ao módulo da interface com o utilizador, a aplicação inicia com o menu principal, e sempre que o jogo acaba, volta sempre para o mesmo.

Figura 8: menu principal.

Neste menu o jogador tem **quatro opções**, ao escolher uma delas, deverá inserir o número da mesma seguido de um ponto final e *enter*. O programa tem um mecanismo de validação do input, pelo que se a opção inserida for inválida, será mostrada uma mensagem e pedido um novo input.

Quanto à primeira opção, é iniciado o jogo que suporta dois jogadores. Na segunda, será o jogador contra o computador, e por último, a terceira opção podemos visualizar o jogo do computador contra computador. Em relação à interface, esta é igual em qualquer modo de jogo.

O jogo começa por mostrar o tabuleiro vazio. Em primeiro lugar, pede ao jogador das peças pretas, denominado ***player x***, onde quer posicionar o primeiro *worker*. Em segundo lugar, pede ao jogador das peças brancas, denominado **player o**, onde quer posicionar o segundo *worker*. Seguidamente, será sempre o mesmo ciclo, em que é pedido a cada jogador, alternadamente, o seguinte:

- Se quer mover ou não um *worker*. Se sim, insere a posição atual do *worker* escolhido e depois a sua futura posição; se não, continua.
- Posição onde quer colocar a nova peça.

Adicionalmente, estão presentes no **Anexo I** algumas capturas de ecrã, para que seja mais fácil compreender a interface com o utilizador.

Conclusões

O projeto teve como principal objetivo aplicar o conhecimento adquirido nas aulas teóricas e práticas, assim como foi realizado no âmbito da unidade curricular de Programação em Lógica.

Ao longo do desenvolvimento deste projeto, foram encontradas algumas dificuldades, nomeadamente o pensamento recursivo e que melhor caminho tomar em cada predicado. Todas estas foram eventualmente superadas.

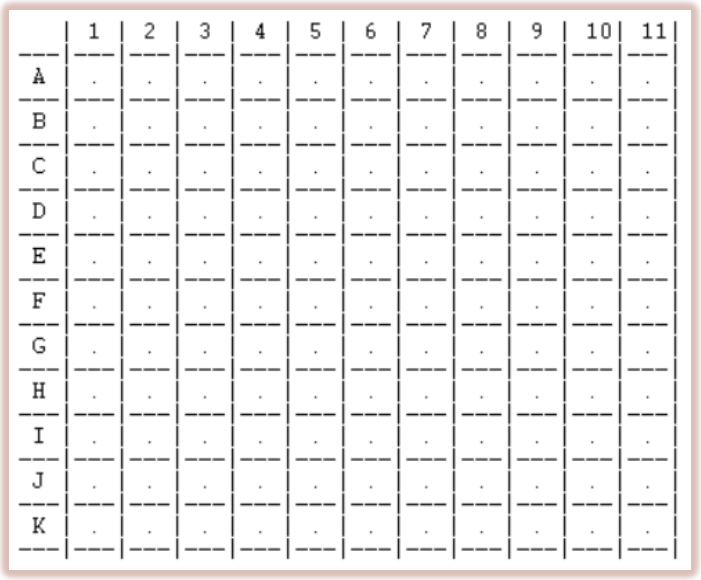
Evidentemente, existem aspetos que podiam ser melhorados no trabalho desenvolvido, como o uso de inteligência artificial, que não chegou a ser implementada como um nível de dificuldade no modo de jogo com o computador, devido a falta de tempo. Seria também desejável de melhorar a eficiência dos métodos concebidos.

Em suma, o trabalho foi concluído com sucesso, e o seu desenvolvimento contribuiu positivamente para uma melhor compreensão da linguagem *Prolog*, que se demonstrou ser bastante complexa.

Bibliografia

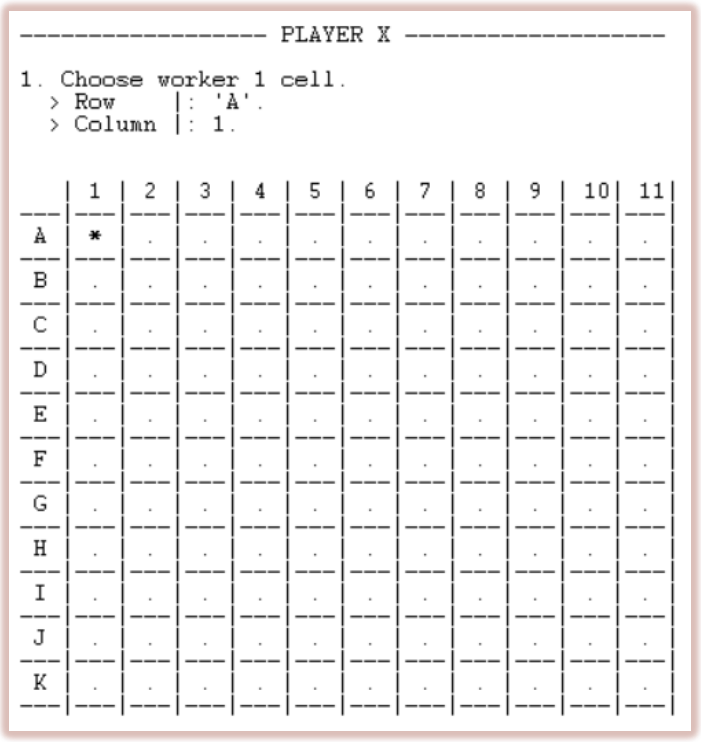
- *https://spielstein.com/games/fabrik*

Anexo I



	1	2	3	4	5	6	7	8	9	10	11
A	.	.	.	.	.	.	.	.	.	.	.
B	.	.	.	.	.	.	.	.	.	.	.
C	.	.	.	.	.	.	.	.	.	.	.
D	.	.	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	.	.	.	.	.
G	.	.	.	.	.	.	.	.	.	.	.
H	.	.	.	.	.	.	.	.	.	.	.
I	.	.	.	.	.	.	.	.	.	.	.
J	.	.	.	.	.	.	.	.	.	.	.
K	.	.	.	.	.	.	.	.	.	.	.

Figura 9: tabuleiro inicial.



```
------------------- PLAYER X --------------------

1. Choose worker 1 cell.
   > Row    |: 'A'.
   > Column |: 1.
```

	1	2	3	4	5	6	7	8	9	10	11
A	*	.	.	.	.	.	.	.	.	.	.
B	.	.	.	.	.	.	.	.	.	.	.
C	.	.	.	.	.	.	.	.	.	.	.
D	.	.	.	.	.	.	.	.	.	.	.
E	.	.	.	.	.	.	.	.	.	.	.
F	.	.	.	.	.	.	.	.	.	.	.
G	.	.	.	.	.	.	.	.	.	.	.
H	.	.	.	.	.	.	.	.	.	.	.
I	.	.	.	.	.	.	.	.	.	.	.
J	.	.	.	.	.	.	.	.	.	.	.
K	.	.	.	.	.	.	.	.	.	.	.

Figura 10: *player x* escolhe a posição do primeiro *worker*.

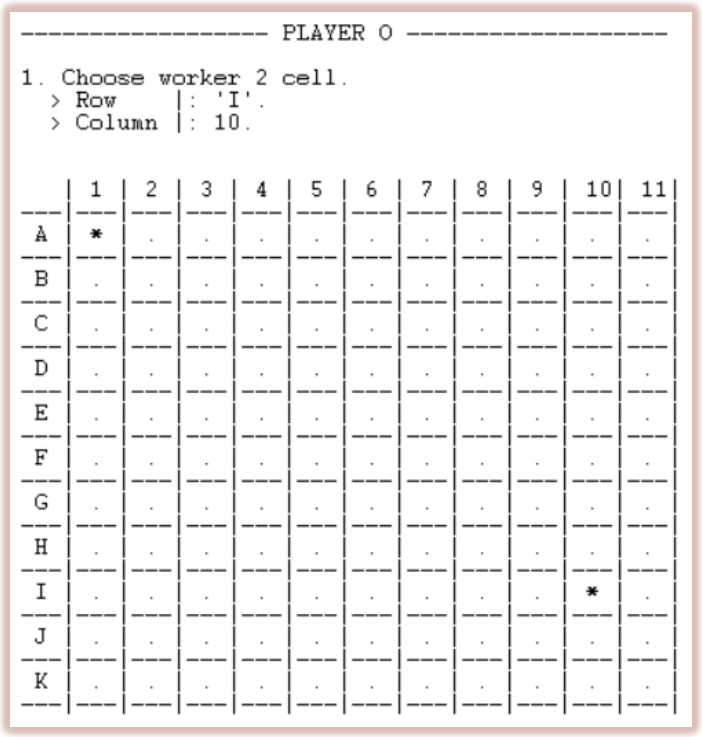


Figura 11: *player o* escolhe a posição do segundo *worker*.

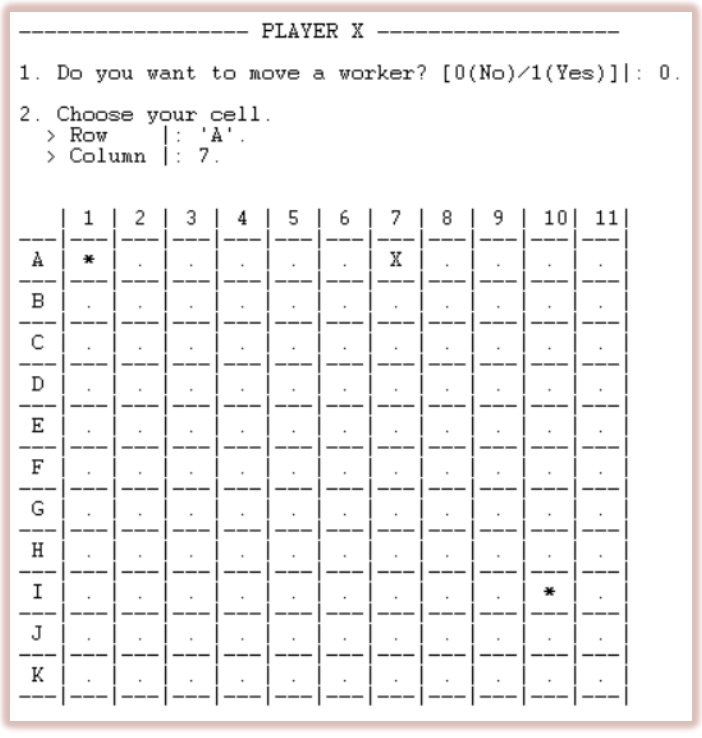


Figura 12: jogada do *player x*.

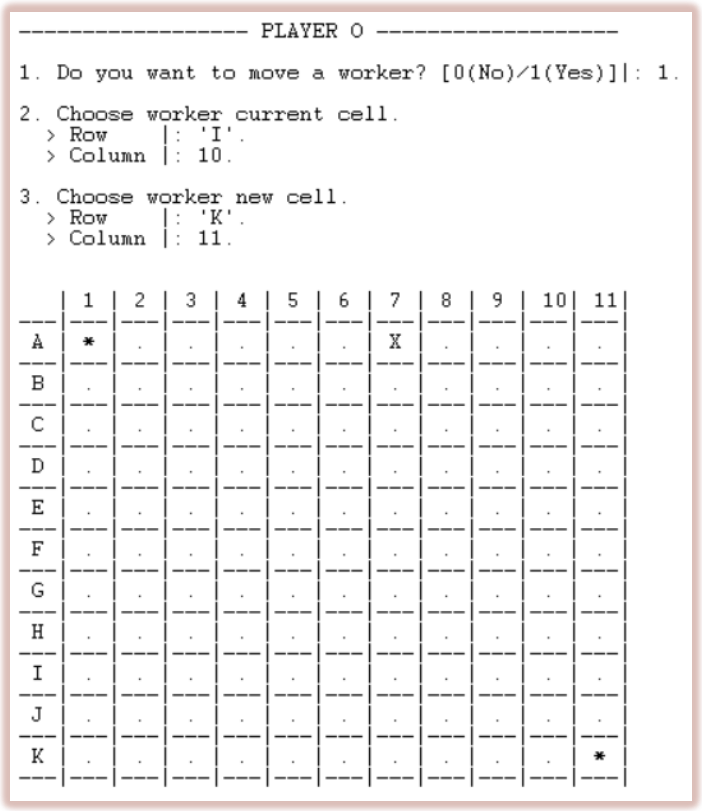


Figura 13: jogada do *player o*, em que move *worker*.

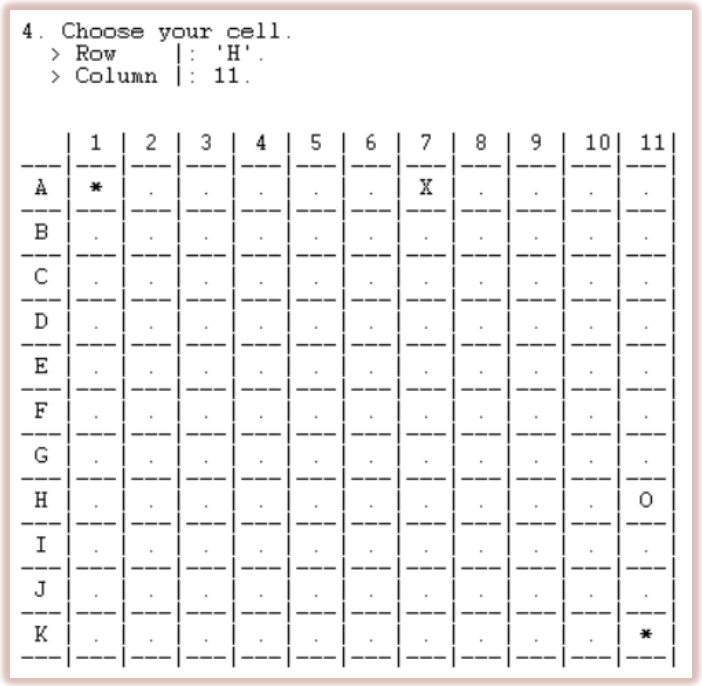
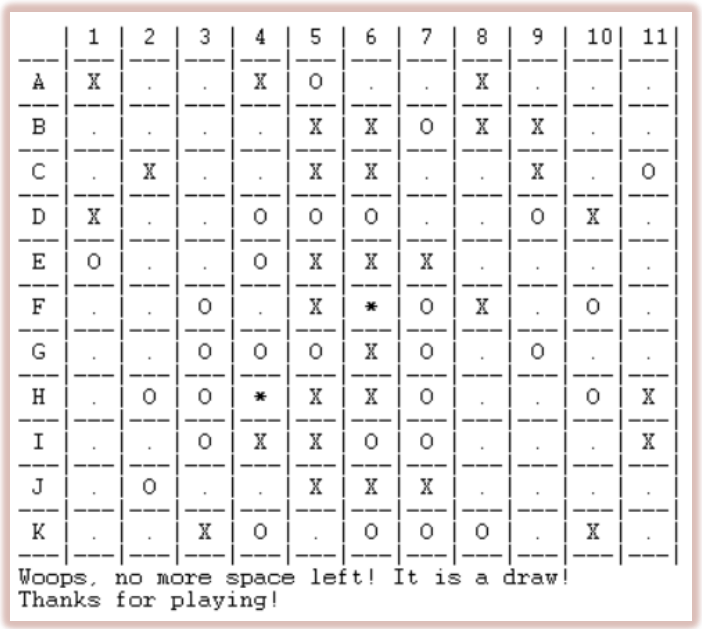
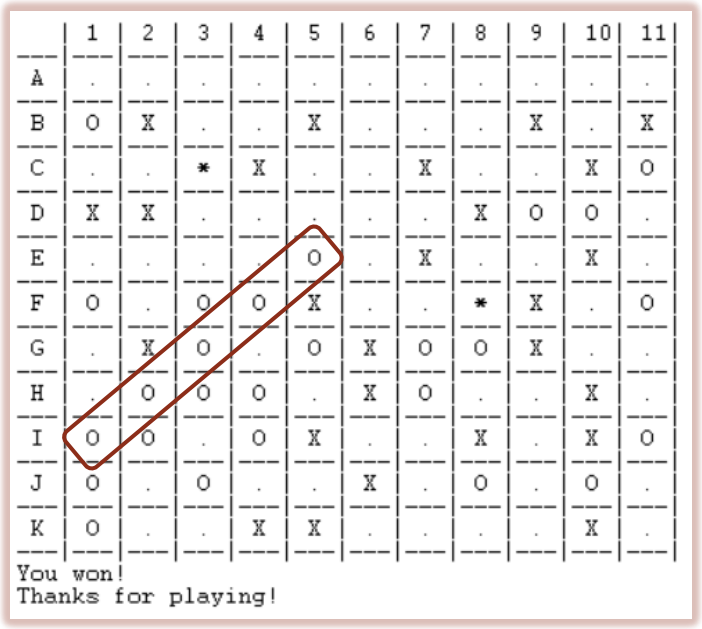


Figura 14: jogada do *player o*.



```
   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11|
---|---|---|---|---|---|---|---|---|---|---|---|
 A | X | . | . | X | O | . | . | X | . | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 B | . | . | . | . | X | X | O | X | X | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 C | . | X | . | . | X | X | . | . | X | . | O |
---|---|---|---|---|---|---|---|---|---|---|---|
 D | X | . | . | O | O | O | . | . | O | X | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 E | O | . | . | O | X | X | X | . | . | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 F | . | . | O | . | X | * | O | X | . | O | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 G | . | . | O | O | O | X | O | . | O | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 H | . | O | O | * | X | X | O | . | . | O | X |
---|---|---|---|---|---|---|---|---|---|---|---|
 I | . | . | O | X | X | O | O | . | . | . | X |
---|---|---|---|---|---|---|---|---|---|---|---|
 J | . | O | . | . | X | X | X | . | . | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 K | . | . | X | O | . | O | O | O | . | X | . |
---|---|---|---|---|---|---|---|---|---|---|---|
Woops, no more space left! It is a draw!
Thanks for playing!
```

Figura 15: fim de jogo no caso de empate.



```
   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10| 11|
---|---|---|---|---|---|---|---|---|---|---|---|
 A | . | . | . | . | . | . | . | . | . | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 B | O | X | . | . | X | . | . | . | X | . | X |
---|---|---|---|---|---|---|---|---|---|---|---|
 C | . | . | * | X | . | . | X | . | . | X | O |
---|---|---|---|---|---|---|---|---|---|---|---|
 D | X | X | . | . | . | . | . | X | O | O | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 E | . | . | . | . | O | . | X | . | . | X | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 F | O | . | O | O | X | . | . | * | X | . | O |
---|---|---|---|---|---|---|---|---|---|---|---|
 G | . | X | O | . | O | X | O | O | X | . | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 H | . | O | O | O | . | X | O | . | . | X | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 I | O | O | . | O | X | . | . | X | . | X | O |
---|---|---|---|---|---|---|---|---|---|---|---|
 J | O | . | O | . | . | X | . | O | . | O | . |
---|---|---|---|---|---|---|---|---|---|---|---|
 K | O | . | . | X | X | . | . | . | . | X | . |
---|---|---|---|---|---|---|---|---|---|---|---|
You won!
Thanks for playing!
```

Figura 16: fim de jogo no caso de jogador vencedor.

Anexo II

fabrik.pl

```
:- consult('menus.pl').
:- consult('display.pl').
:- consult('logic.pl').
:- consult('utilities.pl').
:- consult('input.pl').
:- consult('bot.pl').
:- use_module(library(random)).
:- use_module(library(system)).

fabrik :-
      mainMenu.
```

menus.pl

```
mainMenu :-
    printMainMenu,
    askMenuOption,
    read(Input),
    manageInput(Input).

manageInput(1) :-
    startGame('P','P'),
    mainMenu.

manageInput(2) :-
    startGame('P','C'),
    mainMenu.

manageInput(3) :-
    startGame('C','C'),
    mainMenu.

manageInput(4) :-
    write('valid option!\n\n').

manageInput(0) :-
    write('\nExiting...\n\n').

manageInput(_Other) :-
    write('\nERROR: that option does not exist.\n\n'),
    askMenuOption,
    read(Input),
    manageInput(Input).
```

```
printMainMenu :-
    nl,nl,
    write(' _____________________________________________________________________
'),nl,
    write('|
|'),nl,
    write('|
|'),nl,
    write('|                          ___     _       _ _
|'),nl,
    write('|                         |  _|___| |_ ___|_| |_
|'),nl,
    write('|                         |  _| ."| . |  _| | "_|
|'),nl,
    write('|                         |_| |__,|___|_| |_|_,_|
|'),nl,
    write('|
|'),nl,
    write('|
|'),nl,
    write('|                            Barbara Sofia Silva
|'),nl,
    write('|                               Julieta Frade
|'),nl,
    write('|                ----------------------------------------
|'),nl,
    write('|
|'),nl,
    write('|
|'),nl,
    write('|                            1. Player vs Player
|'),nl,
    write('|
|'),nl,
    write('|                            2. Player vs Computer
|'),nl,
    write('|
|'),nl,
    write('|                            3. Computer vs Computer
|'),nl,
    write('|
|'),nl,
    write('|                            0. Exit
|'),nl,
    write('|
|'),nl,
    write('|
|'),nl,
```

```
    write(' ____________________________________________________________________
'),nl,nl,nl.

askMenuOption :-
    write('> Insert your option ').
```

logic.pl

```
/*Verifica o estato atual do jogo após cada jogada. Caso alguma dos predicados se
satisfaca, acaba o jogo.*/
checkGameState(Player, Board) :-
      (
            (checkVictory(Player, 'Row', Board), write('You won!'));
            (checkVictory(Player, 'Column', Board), write('You won!'));
            (checkVictory(Player, 'DiagonalDown', Board), write('You won!'));
            (checkVictory(Player, 'DiagonalUp', Board), write('You won!'));
            (checkFullBoard(Board), write('Woops, no more space left! It is a
draw!'));
            (checkValidSpots(Board, 0, 0, Result), Result =:= 0, write('Woops, no
more space left! It is a draw!'))
      ).

/*Verifica se existe o padrão de 5 X's seguidos numa linha do tabuleiro.*/
checkVictory(X, 'Row', Board) :-
      append(_, [R|_], Board),
      append(_, [X,X,X,X,X|_], R).

/*Verifica se existe o padrão de 5 X's seguidos numa coluna do tabuleiro.*/
checkVictory(X, 'Column', Board) :-
      append(_, [R1,R2,R3,R4,R5|_], Board),
      append(C1, [X|_], R1), append(C2, [X|_], R2),
      append(C3, [X|_], R3), append(C4, [X|_], R4), append(C5, [X|_], R5),
      length(C1, M), length(C2, M), length(C3, M), length(C4, M), length(C5, M).

/*Verifica se existe o padrão de 5 X's seguidos numa "diagonal a descer" do
tabuleiro.*/
checkVictory(X, 'DiagonalDown', Board) :-
      append(_, [R1,R2,R3,R4,R5|_], Board),
      append(C1, [X|_], R1), append(C2, [X|_], R2),
      append(C3, [X|_], R3), append(C4, [X|_], R4), append(C5, [X|_], R5),
      length(C1, M1), length(C2, M2), length(C3, M3), length(C4, M4), length(C5, M5),
      M2 is M1+1, M3 is M2+1, M4 is M3+1, M5 is M4+1.

/*Verifica se existe o padrão de 5 X's seguidos numa "diagonal a subir" do
tabuleiro.*/
checkVictory(X, 'DiagonalUp', Board) :-
      append(_, [R1,R2,R3,R4,R5|_], Board),
      append(C1, [X|_], R1), append(C2, [X|_], R2),
```

```
      append(C3, [X|_], R3), append(C4, [X|_], R4), append(C5, [X|_], R5),
      length(C1, M1), length(C2, M2), length(C3, M3), length(C4, M4), length(C5, M5),
      M2 is M1-1, M3 is M2-1, M4 is M3-1, M5 is M4-1.

/*Verifica se existe alguma posição válida tendo em conta as linhas de visão do
worker para o próximo jogador colocar a peça. Atribui 0 a Result
se não existir nenhuma, e atribui 1 se existir pelo menos uma.*/
checkValidSpots(Board, Row, Column, Result) :-
      (
            (Column =:= 11, Row1 is Row + 1, checkValidSpots(Board, Row1, 0,
Result));
            (Row =:= 11, Result is 0);
            ((isValidPosLines(Board, Row, Column, Res)),
                  ((Res =:= 0, Column1 is Column + 1, checkValidSpots(Board, Row,
Column1, Result));
                  (Res =:=1 , Result is 1)))
      ), !.

/*Verifica se a célula está na linha de visão do worker. Foi feito o overload
deste predicado para cada um dos sentidos com o intuito do código se tornar mais
percetível.*/
%Default
verifyLine(_Board, _WorkerRow, _WorkerColumn, _Row, _Column, 12, Res, _Ray) :-
      Res is 0.
%O
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'O' ):-
      (Column =:= WorkerColumn - Index, Row =:= WorkerRow, Res is 1);
      ((ColumnTemp is WorkerColumn - Index, getValueFromMatrix(Board, WorkerRow,
ColumnTemp, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'O')).
%NO
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'NO' ):-
      (Row =:= WorkerRow - Index, Column =:= WorkerColumn - Index, Res is 1); %NO
      ((RowTemp is WorkerRow - Index, ColumnTemp is WorkerColumn - Index,
getValueFromMatrix(Board, RowTemp, ColumnTemp, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'NO')).
%N
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'N' ):-
      (Row =:= WorkerRow - Index, Column =:= WorkerColumn, Res is 1);
      ((RowTemp is WorkerRow - Index, getValueFromMatrix(Board, RowTemp,
WorkerColumn, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'N')).
%NE
```

```
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'NE' ):-
      (Row =:= WorkerRow - Index, Column =:= WorkerColumn + Index, Res is 1);
      ((RowTemp is WorkerRow -Index, ColumnTemp is WorkerColumn + Index,
getValueFromMatrix(Board, RowTemp, ColumnTemp, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'NE')).
%E
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'E' ):-
      (Column =:= WorkerColumn + Index, Row =:= WorkerRow, Res is 1);
      ((ColumnTemp is WorkerColumn + Index, getValueFromMatrix(Board, WorkerRow,
ColumnTemp, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'E')).
%SE
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'SE' ):-
      (Row =:= WorkerRow + Index, Column =:= WorkerColumn + Index, Res is 1);
      ((RowTemp is WorkerRow + Index, ColumnTemp is WorkerColumn + Index,
getValueFromMatrix(Board, RowTemp, ColumnTemp, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'SE')).
%S
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'S'):-
      (Row =:= WorkerRow + Index, Column =:= WorkerColumn, Res is 1);
      ((RowTemp is WorkerRow + Index, getValueFromMatrix(Board, RowTemp,
WorkerColumn, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'S')).
%SO
verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index, Res, 'SO' ):-
      (Row =:= WorkerRow + Index, Column =:= WorkerColumn - Index, Res is 1);
      ((RowTemp is WorkerRow + Index, ColumnTemp is WorkerColumn - Index,
getValueFromMatrix(Board, RowTemp, ColumnTemp, Value), Value \= empty, Res is 0),!);
      (Index < 12,
      Index1 is Index + 1,
      verifyLine(Board, WorkerRow, WorkerColumn, Row, Column, Index1, Res, 'SO')).

/*Verifica se a célula (Row, Column) está nalguma das linhas de visão do worker
(WorkerRow, WorkerColumn) com a ajuda do predicado verifyLine.*/
%Res = 1 if that cell is in the worker lines, Res = 0 if it's not.
isWorkerLines(Board, WorkerRow, WorkerColumn, Row, Column, Res) :-
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResN, 'N' ), ResN
=:= 1, Res is 1);
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResNE, 'NE' ), ResNE
=:= 1, Res is 1);
```

```
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResE, 'E' ), ResE
=:= 1, Res is 1);
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResSE, 'SE' ), ResSE
=:= 1, Res is 1);
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResS, 'S' ), ResS
=:= 1, Res is 1);
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResSO, 'SO' ), ResSO
=:= 1, Res is 1);
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResO, 'O' ), ResO
=:= 1, Res is 1);
      (verifyLine(Board, WorkerRow, WorkerColumn, Row, Column,1, ResNO, 'NO' ), ResNO
=:= 1, Res is 1);
      Res is 0.

/*Verifica se a célula (Row, Column) está vazia recorrendo à chamadas ao predicado
getValueFromMatrix.*/
isEmptyCell(Board, Row, Column, Res) :-
    ((getValueFromMatrix(Board, Row, Column, Value), Value == empty, !,
    Res is 1);
    Res is 0).

/*Vai buscar as posições dos workers com a ajuda do predicado getWorkersPos e
verifica se a célula (Row, Column) está na linha de visão de pelo menos um dos
workers.*/
%Res = 1 that cell is valid, Res = 0 if not.
isValidPosLines(Board, Row, Column, Res) :-
      (
            (isEmptyCell(Board, Row, Column, Res), Res =:= 1,
                  (getWorkersPos(Board, Worker1Row, Worker1Column, Worker2Row,
Worker2Column),
                  ((isWorkerLines(Board, Worker1Row, Worker1Column, Row, Column,
ResIsWorkerLines1), ResIsWorkerLines1 =:= 1, Res is 1);
                  (isWorkerLines(Board, Worker2Row, Worker2Column, Row, Column,
ResIsWorkerLines2), ResIsWorkerLines2 =:= 1, Res is 1);
                  Res is 0)));
            (Res is 0)
      ).

/*É chamado para verificar todas as jogadas (tanto do jogador, como do worker)
recorrendo ao predicado isValidPosLines e  getValueFromMatrix. Este ultimo serve
para verificar se naquela célula do tabuleiro está o conteúdo pretendido
(Expected). Por exemplo, se for para  adicionar uma peça, o conteúdo pretendido
é 'empty', mas caso seja para escolher o worker a mover, o conteúdo pretendido
vai ser 'red'. Se não for possível fazer o movimento, este predicado chama o
write que informa o jogador qual a razão da falha, pedindo umas novas
coordenadas.*/
checkMove(Board, Player, NewBoard, Expected, ColumnIndex, RowIndex):-
      (((Player == empty, Expected == red),
            ((getValueFromMatrix(Board, RowIndex, ColumnIndex, Expected),
```

```
                  replaceInMatrix(Board, RowIndex, ColumnIndex, Player, NewBoard));
                  (write('INVALID MOVE: There is no worker in that cell, please try
again!\n\n'),
                  askCoords(Board, Player, NewBoard, Expected))));
      ((Player == red, Expected == empty),
            ((getValueFromMatrix(Board, RowIndex, ColumnIndex, Expected),
                  replaceInMatrix(Board, RowIndex, ColumnIndex, Player, NewBoard));
                  (write('INVALID MOVE: That cell is not empty, please try
again!\n\n'),
                  askCoords(Board, Player, NewBoard, Expected))));
      ((Player == empty),
            ((getValueFromMatrix(Board, RowIndex, ColumnIndex, Expected),
                  replaceInMatrix(Board, RowIndex, ColumnIndex, Player, NewBoard));
                  (write('INVALID MOVE: That cell is not empty, please try
again!\n\n'),
                  askCoords(Board, Player, NewBoard, Expected))));
      ((Player == white; Player == black),
            ((getValueFromMatrix(Board, RowIndex, ColumnIndex, Expected),
                   ((isValidPosLines(Board, RowIndex, ColumnIndex,
ResIsValidPosLines), ResIsValidPosLines =:= 1),
                        replaceInMatrix(Board, RowIndex, ColumnIndex, Player,
NewBoard);
                        (write('INVALID MOVE: That cell is not within the workers
lines of sight, please try again!\n\n'),
                        askCoords(Board, Player, NewBoard, Expected))));
            (write('INVALID MOVE: That cell is not empty, please try again!\n\n'),
            askCoords(Board, Player, NewBoard, Expected))))).


/*Predicado que pede e analisa cada jogada.*/
askCoords(Board, Player, NewBoard, Expected) :-
      manageRow(NewRow),
      manageColumn(NewColumn),
      write('\n'),
      ColumnIndex is NewColumn - 1,
      RowIndex is NewRow - 1,
      checkMove(Board, Player, NewBoard, Expected, ColumnIndex, RowIndex).


printComputerWorkerMove(WorkerRowIndex, WorkerColumnIndex, WorkerNewRowIndex,
WorkerNewColumnIndex):-
      WorkerRow is WorkerRowIndex + 1,
      WorkerColumn is WorkerColumnIndex + 1,
      WorkerNewRow is WorkerNewRowIndex + 1,
      WorkerNewColumn is WorkerNewColumnIndex + 1,
      letter(WorkerRow, WorkerRowLetter),
      letter(WorkerNewRow, WorkerNewRowLetter),
      write(' > Computer moved the worker in the cell [row: '),
write(WorkerRowLetter), write(' column: '), write(WorkerColumn), write('] to the cell
```

```
[row: '),  write(WorkerNewRowLetter), write(' column: '), write(WorkerNewColumn),
write('].\n').

printComputerMove(NewRowIndex, NewColumnIndex):-
      NewRow is NewRowIndex + 1,
      NewColumn is NewColumnIndex + 1,
      letter(NewRow, RowLetter),
      write(' > Computer added a piece in the cell [row: '), write(RowLetter),
write(' column: '), write(NewColumn), write(']\n').

printComputerAddWorker(WorkerRowIndex, WorkerColumnIndex):-
      WorkerRow is WorkerRowIndex + 1,
      WorkerColumn is WorkerColumnIndex + 1,
      letter(WorkerRow, WorkerRowLetter),
      write(' > Computer added a worker in the cell [row: '), write(WorkerRowLetter),
write(' column: '), write(WorkerColumn), write(']\n').

computerMoveWorkers(Board1, NewBoard):-
      ((moveWorker(Board1, WorkerRowIndex, WorkerColumnIndex, WorkerNewRowIndex,
WorkerNewColumnIndex, ResMoveWorker), ResMoveWorker =:= 1,
      sleep(1),
      replaceInMatrix(Board1,  WorkerRowIndex, WorkerColumnIndex, empty, Board2),
      replaceInMatrix(Board2,  WorkerNewRowIndex, WorkerNewColumnIndex, red,
NewBoard),
      printComputerWorkerMove(WorkerRowIndex, WorkerColumnIndex, WorkerNewRowIndex,
WorkerNewColumnIndex));
      (NewBoard = Board1, sleep(1), write(' > Computer did not move any
worker.\n'))).

moveWorker(Board, 1, NewBoard) :-
        write('\n2. Choose worker current cell.\n'),
        askCoords(Board, empty, NoWorkerBoard, red),
        write('3. Choose worker new cell.\n'),
        askCoords(NoWorkerBoard, red, NewBoard, empty),
        printBoard(NewBoard),
        write('\n4. Choose your cell.\n').

moveWorker(Board, 0, NewBoard) :-
        NewBoard = Board,
        write('\n2. Choose your cell.\n').


addWorkers(InitialBoard, WorkersBoard, 'P', 'P') :-
      printBoard(InitialBoard),
      write('\n------------------ PLAYER X ------------------\n\n'),
      write('1. Choose worker 1 cell.\n'),
      askCoords(InitialBoard, red, Worker1Board, empty),
      printBoard(Worker1Board),
      write('\n------------------ PLAYER O ------------------\n\n'),
```

```
      write('1. Choose worker 2 cell.\n'),
      askCoords(Worker1Board, red, WorkersBoard, empty),
      printBoard(WorkersBoard).

addWorkers(InitialBoard, WorkersBoard, 'P', 'C') :-
      printBoard(InitialBoard),
      write('\n------------------ PLAYER X ------------------\n\n'),
      write('1. Choose worker 1 cell.\n'),
      askCoords(InitialBoard, red, Worker1Board, empty),
      printBoard(Worker1Board),
      write('\n----------------- COMPUTER O -----------------\n\n'),
      generateWorkerMove(Worker1Board, WorkerRowIndex, WorkerColumnIndex),
      sleep(1),
      replaceInMatrix(Worker1Board,  WorkerRowIndex, WorkerColumnIndex, red,
WorkersBoard),
      printComputerAddWorker(WorkerRowIndex, WorkerColumnIndex),
      printBoard(WorkersBoard).

addWorkers(InitialBoard, WorkersBoard, 'C', 'C') :-
      printBoard(InitialBoard),
      write('\n----------------- COMPUTER X -----------------\n\n'),
      generateWorkerMove(InitialBoard, WorkerRowIndex1, WorkerColumnIndex1),
      sleep(1),
      replaceInMatrix(InitialBoard,  WorkerRowIndex1, WorkerColumnIndex1, red,
Worker1Board),
      printComputerAddWorker(WorkerRowIndex1, WorkerColumnIndex1),
      printBoard(Worker1Board),
      write('\n----------------- COMPUTER O -----------------\n\n'),
      generateWorkerMove(Worker1Board, WorkerRowIndex2, WorkerColumnIndex2),
      sleep(1),
      replaceInMatrix(Worker1Board,  WorkerRowIndex2, WorkerColumnIndex2, red,
WorkersBoard),
      printComputerAddWorker(WorkerRowIndex2, WorkerColumnIndex2),
      printBoard(WorkersBoard).


blackPlayerTurn(Board, NewBoard, 'P') :-
      write('\n------------------ PLAYER X ------------------\n\n'),
      write('1. Do you want to move a worker? [0(No)/1(Yes)]'),
      manageMoveWorkerBool(MoveWorkerBoolX),
      moveWorker(Board, MoveWorkerBoolX, Board1),
      askCoords(Board1, black, NewBoard, empty),
      printBoard(NewBoard).

blackPlayerTurn(Board, NewBoard, 'C') :-
      write('\n----------------- COMPUTER X -----------------\n\n'),
      computerMoveWorkers(Board, Board1),
      generatePlayerMove(Board1, NewRowIndex, NewColumnIndex),
      replaceInMatrix(Board1,  NewRowIndex, NewColumnIndex, black, NewBoard),
```

```
      printComputerMove(NewRowIndex, NewColumnIndex),
      printBoard(NewBoard).

whitePlayerTurn(NewBoard, FinalBoard, 'P') :-
      write('\n------------------ PLAYER O ------------------\n\n'),
      write('1. Do you want to move a worker? [0(No)/1(Yes)]'),
      manageMoveWorkerBool(MoveWorkerBoolO),
      moveWorker(NewBoard, MoveWorkerBoolO, Board1),
      askCoords(Board1, white, FinalBoard, empty),
      printBoard(FinalBoard).

whitePlayerTurn(Board, FinalBoard, 'C') :-
      write('\n----------------- COMPUTER O -----------------\n\n'),
      computerMoveWorkers(Board, Board1),
      generatePlayerMove(Board1, NewRowIndex, NewColumnIndex),
      replaceInMatrix(Board1,  NewRowIndex, NewColumnIndex, white, FinalBoard),
      printComputerMove(NewRowIndex, NewColumnIndex),
      printBoard(FinalBoard).

/*Loop do jogo, em que recebe a jogada de cada jogador e verifica o estado do jogo a
seguir.*/
gameLoop(Board, Player1, Player2) :-
      blackPlayerTurn(Board, NewBoard, Player1),
      (
            (checkGameState('black', NewBoard), write('\nThanks for playing!\n'));
            (whitePlayerTurn(NewBoard, FinalBoard, Player2),
                  (
                        (checkGameState('white', FinalBoard), write('\nThanks for
playing!\n'));
                        (gameLoop(FinalBoard, Player1, Player2))
                  )
            )
      ).

startGame(Player1, Player2) :-
      initialBoard(InitialBoard),
      addWorkers(InitialBoard, WorkersBoard, Player1, Player2),
      gameLoop(WorkersBoard, Player1, Player2).
```

utilities.pl

```
replaceInList([_H|T], 0, Value, [Value|T]).
replaceInList([H|T], Index, Value, [H|TNew]) :-
        Index > 0,
        Index1 is Index - 1,
        replaceInList(T, Index1, Value, TNew).

replaceInMatrix([H|T], 0, Column,Value, [HNew|T]) :-
```

```
        replaceInList(H, Column, Value, HNew).

replaceInMatrix([H|T], Row, Column, Value, [H|TNew]) :-
        Row > 0,
        Row1 is Row - 1,
        replaceInMatrix(T, Row1, Column, Value, TNew).

getValueFromList([H|_T], 0, Value) :-
        Value = H.

getValueFromList([_H|T], Index, Value) :-
        Index > 0,
        Index1 is Index - 1,
        getValueFromList(T, Index1, Value).

getValueFromMatrix([H|_T], 0, Column, Value) :-
        getValueFromList(H, Column, Value).

getValueFromMatrix([_H|T], Row, Column, Value) :-
        Row > 0,
        Row1 is Row - 1,
        getValueFromMatrix(T, Row1, Column, Value).

/*Analisa o que está na célula (Row, Column) da matriz. Retorna em Value o
conteúdo daquela célula, ou, caso Value já esteja atribuído, a função falha.*/
getWorkersPosColumn(Board, Value, Row, Column, WorkerRow, WorkerColumn) :-
        (getValueFromMatrix(Board, Row, Column, Value),
        WorkerRow = Row, WorkerColumn = Column);
        (Column < 11,
        Column1 is Column + 1,
        getWorkersPosColumn(Board, Value, Row, Column1, WorkerRow, WorkerColumn)).

getWorkersPosRow(Board, Value, Row, Column, WorkerRow, WorkerColumn) :-
        getWorkersPosColumn(Board, Value, Row, Column, WorkerRow, WorkerColumn);
        (Row < 11,
        Row1 is Row + 1,
        getWorkersPosRow(Board, Value, Row1, Column, WorkerRow, WorkerColumn)).

/*Percorre o tabuleiro e com a ajuda do predicado getValueFromMatrix, devolve
em WorkerRow1, WorkerColumn1, WorkerRow2 e WorkerColumn2 as posições dos workers
na matriz. */
getWorkersPos(Board, WorkerRow1, WorkerColumn1, WorkerRow2, WorkerColumn2) :-
        Value = red,
        getWorkersPosRow(Board,Value, 0,0, WorkerRow1, WorkerColumn1),
        replaceInMatrix(Board, WorkerRow1, WorkerColumn1, 'RED', NewBoard),
%substituir worker1 por RED para nao ser considerado quando  procurar worker2.
        getWorkersPosRow(NewBoard,Value, 0,0, WorkerRow2, WorkerColumn2).

/*Verifica se o tabuleiro está cheio, confirmando se não há nenhuma célula
```

```
'empty' no tabuleiro.*/
checkFullBoard(Board) :-
      \+ (append(_, [R|_], Board),
        append(_, ['empty'|_], R)).
```

bot.pl

```
verifyRandomMove(Board, Row, Column, Res):-
    ((isValidPosLines(Board, Row, Column, ResIsValidPosLines), ResIsValidPosLines =:=
1, !,
    Res is 1);
    Res is 0).

/*Gera um linha e coluna aleatória, verifica se é uma jogada válida, ou seja,
se é uma célula que está nas linhas de visão de algum worker e se a célula está
atualmente vazia. Caso seja válida, ele devolve essa linha e coluna, caso
contrário este predicado chama-se a si própria para tentar gerar uma nova posição.*/
generatePlayerMove(Board, Row, Column):-
    random(0,11,RandomRow),
    random(0,11,RandomColumn),
    ((verifyRandomMove(Board, RandomRow, RandomColumn,
ResVerifyRandomMove),ResVerifyRandomMove=:=1,
    isEmptyCell(Board, RandomRow, RandomColumn, ResIsEmptyCell), ResIsEmptyCell=:=1,
        Row is RandomRow, Column is RandomColumn);
        generatePlayerMove(Board, Row, Column)).

/*Escolhe aleatoriamente se vai mover ou não o worker. Se escolher mover o worker,
 chama o predicado chooseWorker para escolher aleatoriamente o worker a mover e
 seguidamente chama a função generateWorkerMove para saber a posição para qual
 mover.*/
moveWorker(Board, WorkerRow, WorkerColumn, WorkerNewRow, WorkerNewColumn, Res) :-
    random(0,2, Bool),
    ((Bool =:= 1,
    chooseWorker(Board, WorkerRow, WorkerColumn), generateWorkerMove(Board,
WorkerNewRow, WorkerNewColumn), Res = 1);
    Res is 0).

%Escolhe aleatoriamente o worker a mover.
chooseWorker(Board, WorkerRow, WorkerColumn) :-
    getWorkersPos(Board, WorkerRow1, WorkerColumn1, WorkerRow2, WorkerColumn2),
    random(1,3, Bool),
    ((Bool =:= 1, WorkerRow is WorkerRow1, WorkerColumn is WorkerColumn1);
    (Bool =:= 2, WorkerRow is WorkerRow2, WorkerColumn is WorkerColumn2)).


/*Gera um linha e coluna aleatória, verifica se é uma jogada válida, ou seja,
se é uma célula que está nas linhas de visão de algum worker e se a célula
```

```
está atualmente vazia. Caso seja válida, ele devolve essa linha e coluna,
caso contrário este predicado chama-se a si própria para tentar gerar uma
nova posição.*/
generateWorkerMove(Board,WorkerNewRow, WorkerNewColumn) :-
    random(0,11,RandomWorkerNewRow),
    random(0,11,RandomWorkerNewColumn),
    ((isEmptyCell(Board, RandomWorkerNewRow, RandomWorkerNewColumn, ResIsEmptyCell),
ResIsEmptyCell=:=1,
        WorkerNewRow is RandomWorkerNewRow, WorkerNewColumn is
RandomWorkerNewColumn);
        generateWorkerMove(Board, WorkerNewRow, WorkerNewColumn)).
```

input.pl

```
/*Todos os predicados deste ficheiro analisam o input, relativamente às linhas e
colunas inseridas pelo utilizador,
e se as mesmas estão dentro dos limites do tabuleiro. Caso não se satisfaca, pede
novamente a informação.*/

manageRow(NewRow) :-
    readRow(Row),
    validateRow(Row, NewRow).

manageColumn(NewColumn) :-
    readColumn(Column),
    validateColumn(Column, NewColumn).

readRow(Row) :-
    write('  > Row    '),
    read(Row).

readColumn(Column) :-
    write('  > Column '),
    read(Column).

validateRow('A', NewRow) :-
    NewRow = 1.

validateRow('B', NewRow) :-
    NewRow = 2.

validateRow('C', NewRow) :-
    NewRow = 3.

validateRow('D', NewRow) :-
    NewRow = 4.

validateRow('E', NewRow) :-
```

```
    NewRow = 5.

validateRow('F', NewRow) :-
    NewRow = 6.

validateRow('G', NewRow) :-
    NewRow = 7.

validateRow('H', NewRow) :-
    NewRow = 8.

validateRow('I', NewRow) :-
    NewRow = 9.

validateRow('J', NewRow) :-
    NewRow = 10.

validateRow('K', NewRow) :-
    NewRow = 11.

validateRow(_Row, NewRow) :-
    write('ERROR: That row is not valid!\n\n'),
    readRow(Input),
    validateRow(Input, NewRow).

validateColumn(1, NewColumn) :-
    NewColumn = 1.

validateColumn(2, NewColumn) :-
    NewColumn = 2.

validateColumn(3, NewColumn) :-
    NewColumn = 3.

validateColumn(4, NewColumn) :-
    NewColumn = 4.

validateColumn(5, NewColumn) :-
    NewColumn = 5.

validateColumn(6, NewColumn) :-
    NewColumn = 6.

validateColumn(7, NewColumn) :-
    NewColumn = 7.

validateColumn(8, NewColumn) :-
    NewColumn = 8.
```

```
validateColumn(9, NewColumn) :-
    NewColumn = 9.

validateColumn(10, NewColumn) :-
    NewColumn = 10.

validateColumn(11, NewColumn) :-
    NewColumn = 11.

validateColumn(_Column, NewColumn) :-
    write('ERROR: That column is not valid!\n\n'),
    readColumn(Input),
    validateColumn(Input, NewColumn).

manageMoveWorkerBool(NewMoveWorkerBool):-
      read(MoveWorkerBool),
      validateMoveWorkerBool(MoveWorkerBool, NewMoveWorkerBool).

validateMoveWorkerBool(1, NewMoveWorkerBool) :-
    NewMoveWorkerBool = 1.

validateMoveWorkerBool(0, NewMoveWorkerBool) :-
    NewMoveWorkerBool = 0.

validateMoveWorkerBool(_Bool, NewMoveWorkerBool) :-
    write('\nERROR: That answer is not valid, please try again![0(No)/1(Yes)]'),
    read(Input),
    validateMoveWorkerBool(Input, NewMoveWorkerBool).
```