

*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

Resumo sucinto do trabalho com 150 a 250 palavras (problema abordado,

objetivo, como foi o problema resolvido/abordado, principais resultados e conclusões).

Índice

[Introdução 3](#_Toc498277725)

[O Jogo: Fabrik 4](#_Toc498277726)

[Lógica do Jogo 7](#_Toc498277727)

[Representação do Estado do Jogo 7](#_Toc498277728)

[Visualização do Tabuleiro 9](#_Toc498277729)

[Lista de Jogadas Válidas 10](#_Toc498277730)

[Execução de Jogadas 11](#_Toc498277731)

[Avaliação do Tabuleiro 12](#_Toc498277732)

[Final do Jogo 13](#_Toc498277733)

[Jogada do Computador 14](#_Toc498277734)

[Interface com o Utilizador 15](#_Toc498277735)

[Conclusões 16](#_Toc498277736)

[Bibliografia 17](#_Toc498277737)

[Anexo I 18](#_Toc498277738)

[Anexo II 22](#_Toc498277739)

# *Introdução*

Texto.

# *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],

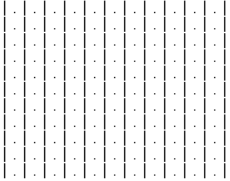
[empty,empty,**red**,**black**,empty,empty,**black**,empty,**white**,empty,empty],

[**white**,**black**,empty,empty,empty,empty,empty,empty,empty,**red**,empty],

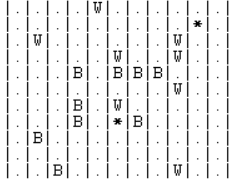
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],

[empty,empty,**black**,empty,empty,empty,empty,empty,**white**,empty,empty]

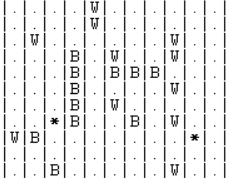
]).



**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:

*/\*pieces symbols\*/*

symbol(empty,S) :- S='.'.

symbol(black,S) :- S='B'.

symbol(white,S) :- S='W'.

symbol(red,S) :- S='\*'.

*/\*print board\*/*

printBoard([Head|Tail]) :-

write('|'),

printLine(Head),

nl,

printBoard(Tail).

printBoard([ ]).

*/\*print line \*/*

printLine([Head|Tail]) :-

symbol(Head,S),

write(S),

write('|'),

printLine(Tail).

printLine([ ]).

O output produzido está ilustrado nas imagens da página anterior.

## *Lista de Jogadas Válidas*

Texto.

## *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*

Texto.

## *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*

Texto.

# *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 7:** 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.

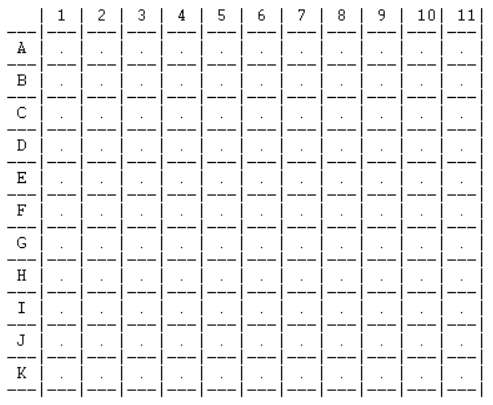
Evidentemente, existem aspetos que podiam ser melhorados no trabalho desenvolvido, como o uso de inteligência artificial, que não chegou a ser implementada como um nível de dificuldade no modo de jogo com o computador, devido a falta de tempo. Seria também desejável de melhorar a eficiência dos métodos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

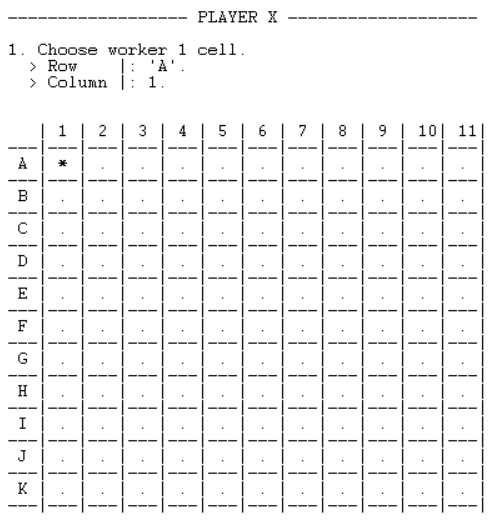
# *Bibliografia*

* <https://spielstein.com/games/fabrik>

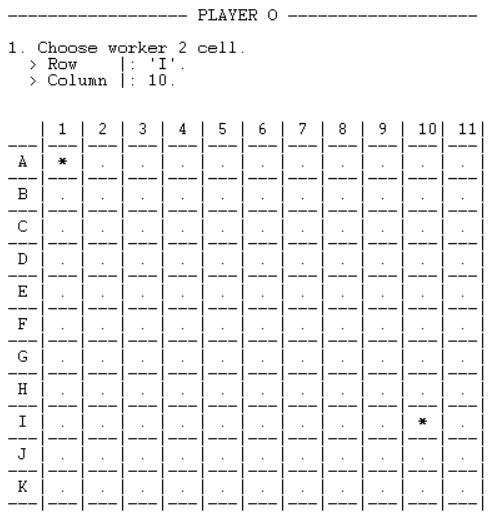
# *Anexo I*



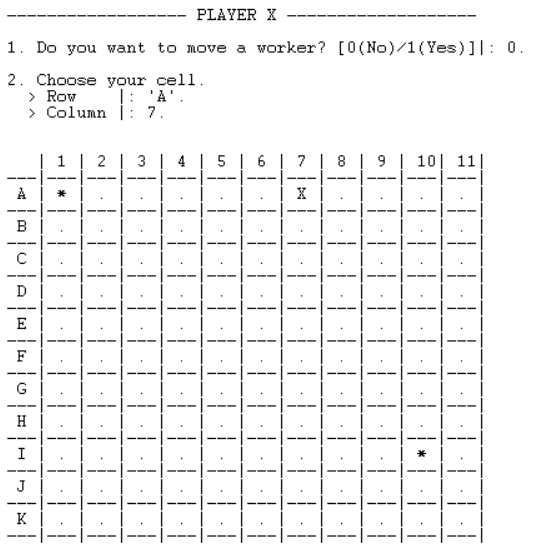
**Figura 8:** tabuleiro inicial.



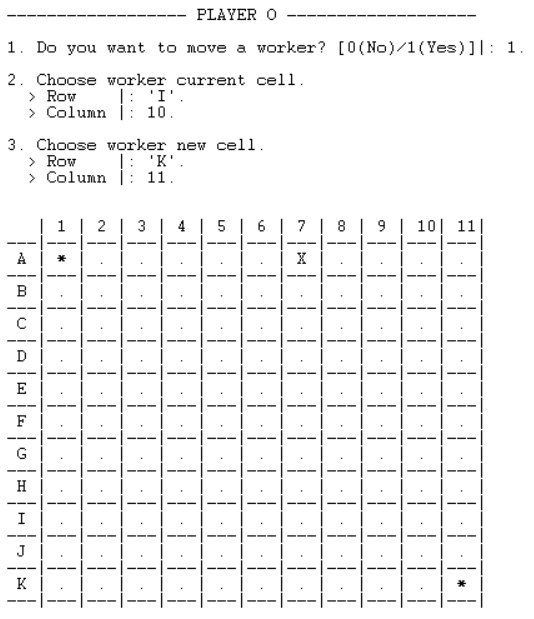
**Figura 9:** player x escolhe a posição do primeiro worker.



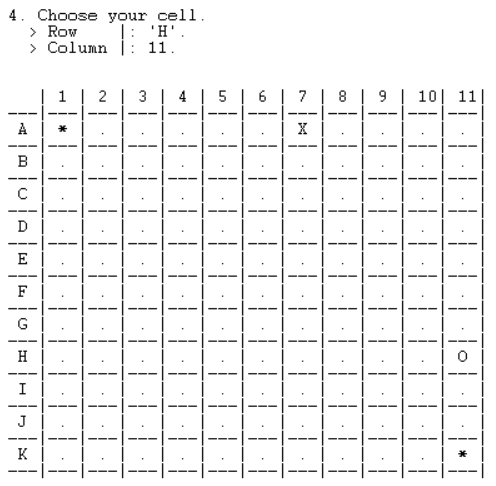
**Figura 10:** player o escolhe a posição do segundo worker.



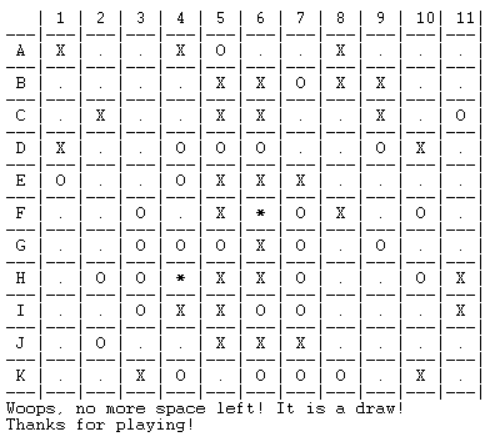
**Figura 11:** jogada do player x.



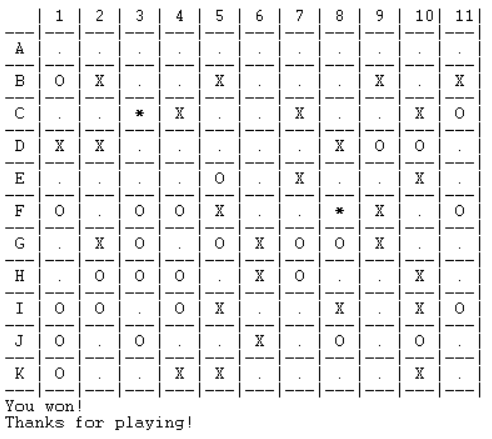
**Figura 12:** jogada do player o, em que move worker.



**Figura 13:** jogada do player o.



**Figura 14:** fim de jogo no caso de empate.



**Figura 15:** fim de jogo no caso de jogador vencedor.

# *Anexo II*