

*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.

O jogo escolhido chama-se Fabrik, é destinado a dois jogadores e baseia-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.

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 a eficiência que é usar a linguagem de Prolog para resolver problemas de decisão.

A principal dificuldade deste projeto foi a difícil adaptação à linguagem pois nunca tínhamos tido qualquer contacto com esta linguagem antes.

O resultado final é um jogo simples, intuitivo e de fácil interação com o utilizador.

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*

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 XXX:** 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**:
  + **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],

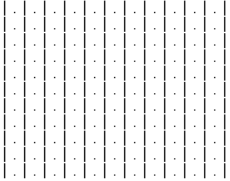
[empty,empty,red,black,empty,empty,black,empty,white,empty,empty],

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

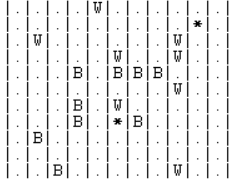
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],

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

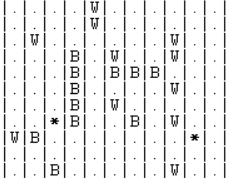
]).



**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),

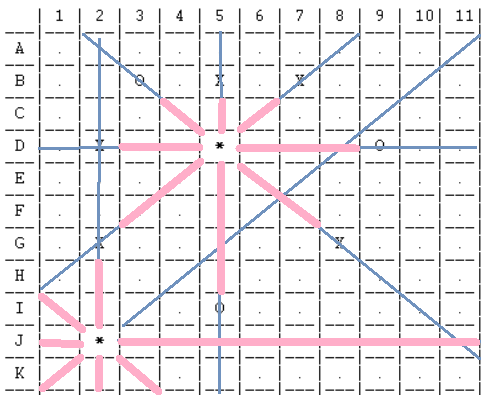
write(S),

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 das 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 à 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*

Texto.

## *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.

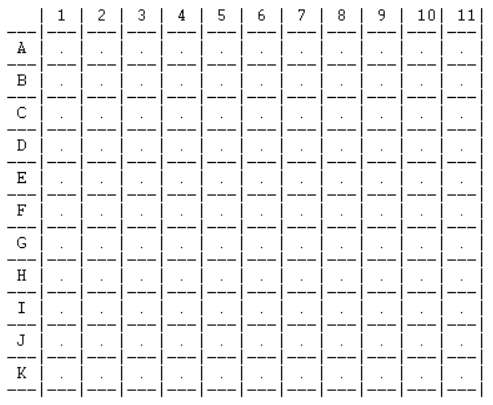
# *Conclusões*

Texto.

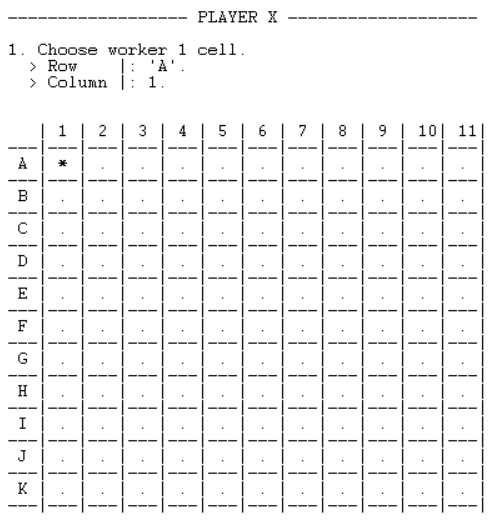
# *Bibliografia*

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

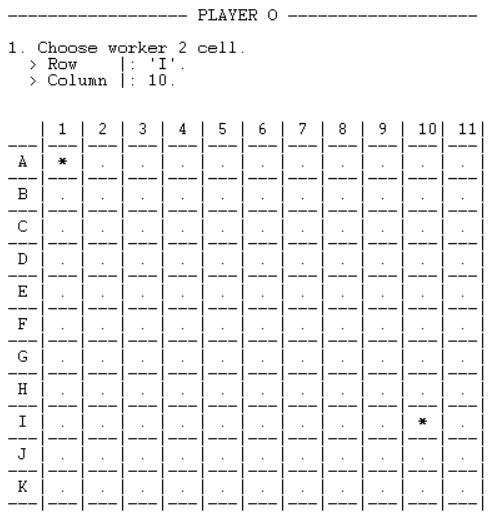
# *Anexo I*



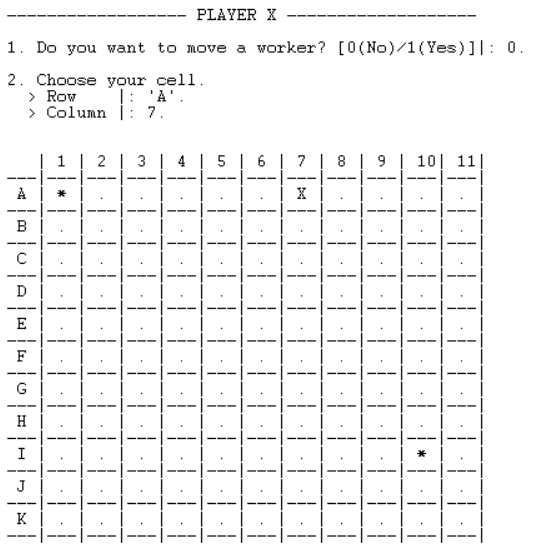
**Figura 9:** tabuleiro inicial.



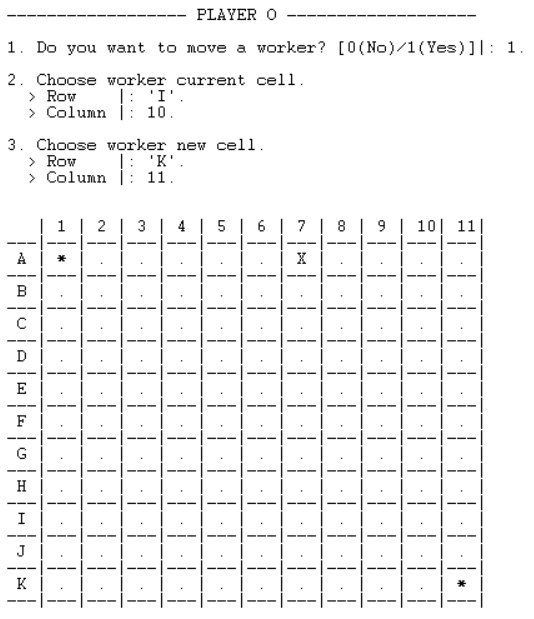
**Figura 10:** player x escolhe a posição do primeiro worker.



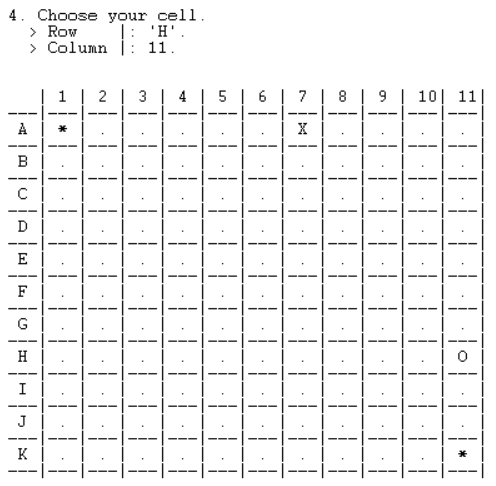
**Figura 11:** player o escolhe a posição do segundo worker.



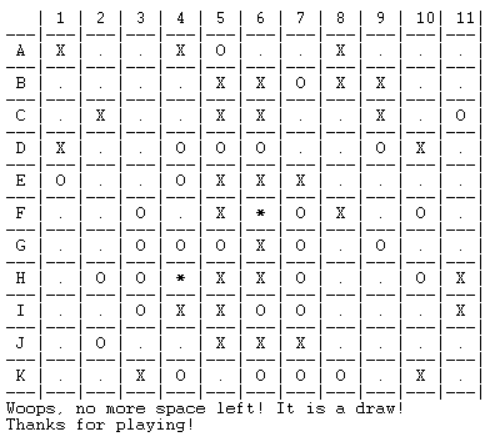
**Figura 12:** jogada do player x.



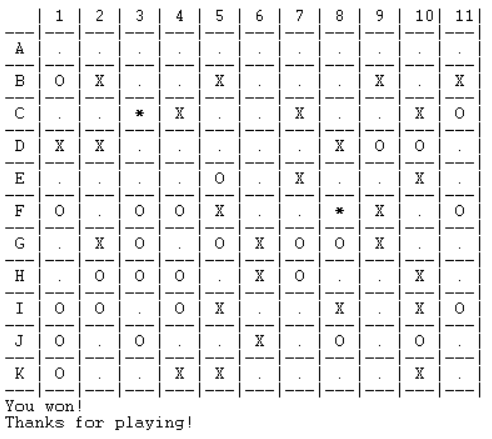
**Figura 13:** jogada do player o, em que move worker.



**Figura 14:** jogada do player o.



**Figura 15:** fim de jogo no caso de empate.



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

# *Anexo II*