FABRIK

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Fabrik3:

André Cruz - 201503776
Edgar Carneiro - 201503778

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

14 de Outubro de 2017

Conteúdo

1 O Jogo*Fabrik* **3**
1.1 História 3
1.2 Material 3
1.3 Regras 3

2 Modelação do Jogo em Prolog **5**
2.1 Representação do Estado do Jogo 5
2.2 Visualização do Tabuleiro 8
2.3 Movimentos 10

1 O Jogo*Fabrik*

1.1 História

O jogo - *Fabrik* - foi recentemente desenvolvido por Dieter Stein, em Agosto de 2017, como parte de um estudo para o desenvolvimento de um novo jogo, Urbino.

1.2 Material

- Tabuleiro quadrangular
- Quantidade suficiente de peças pretas e brancas
- Duas peças vermelhas chamadas trabalhadores

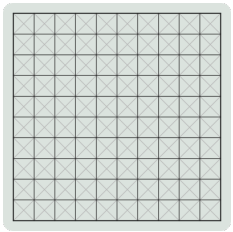

Figura 1: Tabuleiro vazio de 11 x 11 espaços

1.3 Regras

As pretas (jogador que joga com peças de cor preta) começam por colocar um dos trabalhadores num espaço á sua escolha. De seguida, as brancas (jogador que joga com peças de cor branca) coloca o outro trabalhador num espaço qualquer. As pretas decidem quem começa por jogar. O jogo funciona por turnos sendo que em cada um turno um jogador pode, se assim optar, mexer um dos trabalhadores para um espaço vazio. De seguida, o jogador deve jogar uma das suas peças e coloca-la num ponto de interseção entre as "linhas de visão dos dois trabalhadores". As linhas de visão dos trabalhadores são as linhas na diagonal, horizontal e vertical sobre as quais os trabalhadores se encontram posicionados.

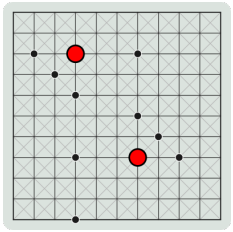

Figura 2: Pontos de interseção entre os dois trabalhadores

No caso especial em que os dois trabalhadores se encontram sobre uma mesma linha ortogonal ou diagonal, apenas os espaços entre eles são considerados pontos de interseção (se estiverem vazios), ao invés da totalidade dessa linha. Ganha o jogo o jogador que consiga criar uma linha de pelo menos 5 pedras da sua cor, ortogonalmente ou diagonalmente. Um jogador ganha também o jogo se o seu adversário não conseguir posicionar nenhum dos trabalhadores de forma a poder colocar uma pedra sua no tabuleiro.

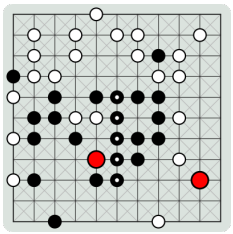

Figura 3: Final de uma partida de Fabrik, com vitórias das pretas

Devem ser incluidas as fontes de informação (e.g. URLs em rodapé).

2 Modelação do Jogo em Prolog

2.1 Representação do Estado do Jogo

A representação dos estados de jogo é feita com recurso a uma lista de listas, de forma a simular o uso de uma Matriz. Seguem de seguida, a representaçao de diferentes estados de jogo:

Representação do estado inicial:

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

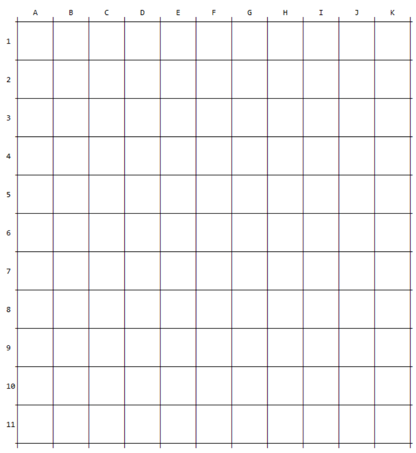

Figura 4: Representação do estado inicial na consola

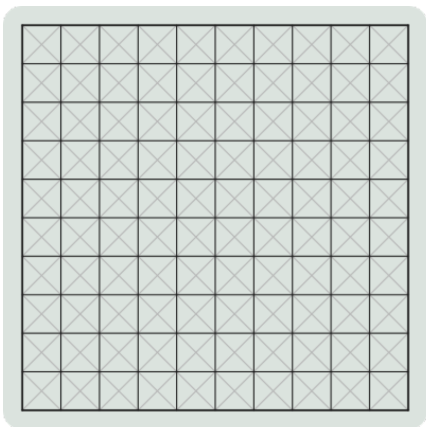

Figura 5: Tabuleiro original Vazio

Representação de um possível estado intermédio:

```
[[empty, empty, empty, empty, empty, empty, empty, empty, empty, empty,
    empty],
 [empty, empty, empty, empty, empty, white, worker, empty, empty, empty,
     empty],
 [empty, empty, empty, empty, empty, empty, white, black, white, empty,
     empty],
 [empty, white, white, empty, empty, empty, empty, white, white, empty,
     empty],
 [white, empty, empty, empty, black, black, empty, black, empty, empty,
     empty],
 [empty, empty, black, white, empty, empty, empty, empty, white, empty,
     empty],
 [empty, black, empty, empty, empty, black, black, black, empty, empty,
     empty],
 [empty, empty, empty, empty, empty, empty, black, empty, white, empty,
     empty],
 [empty, empty, empty, empty, empty, black, empty, empty, empty, empty,
     empty],
 [empty, empty, empty, empty, worker, empty, empty, empty, empty, empty,
     empty],
 [empty, empty, empty, empty, empty, empty, empty, empty, empty, empty,
     empty]]
```

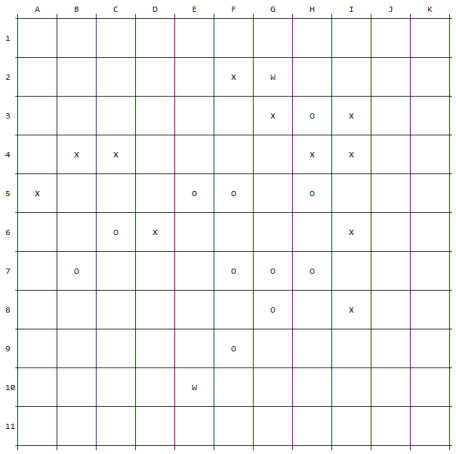

Figura 6: Representação na consola, de um possível estado intermédio

Representação de um possível estado final:

```
[[empty, empty, empty, empty, white, empty, empty, empty, empty, empty,
    empty],
 [empty, white, empty, white, empty, white, white, empty, empty, white,
     empty],
 [empty, white, empty, white, empty, empty, white, black, white, empty,
     empty],
 [black, white, white, empty, empty, empty, empty, white, white, empty,
     empty],
 [white, empty, black, empty, black, black, black, black, empty, empty,
     empty],
 [empty, black, black, white, white, black, empty, black, white, empty,
     empty],
 [white, black, empty, black, empty, black, black, black, empty, empty,
     empty],
 [empty, empty, empty, empty, worker, black, black, empty, white, empty,
     empty],
 [white, black, empty, empty, black, black, empty, empty, empty, worker,
     empty],
 [empty, empty, empty, empty, empty, empty, empty, empty, empty, empty,
     empty],
 [empty, empty, black, empty, empty, empty, empty, white, empty, empty,
     empty]]
```

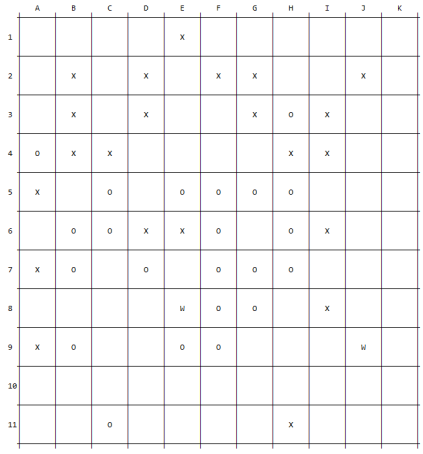

Figura 7: Representação do estado final na consola

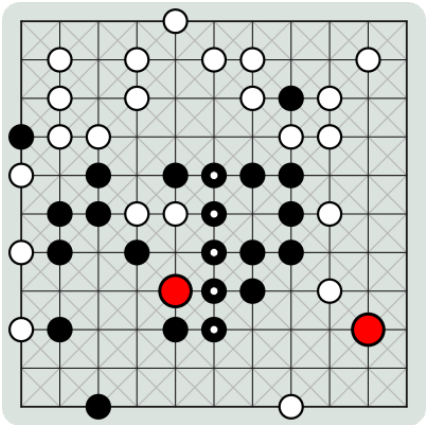

Figura 8: Representção do mesmo estado final, no tabuleiro original

2.2 Visualização do Tabuleiro

Para a representação do tabuleiro em modo de texto, foi criado o seguinte código em prolog:

```
%Board Printing - Character Translation
translate(empty, Symbol) :- Symbol = ' '.
translate(black, Symbol) :- Symbol = 'O'. %Dark Pieces
translate(white, Symbol) :- Symbol = 'X'. %White Pieces
translate(worker, Symbol) :- Symbol = 'W'. %Red Workers

%Board Printing - arguments: Board and Board size
printFabrik(Board, N):-
        nl,
        write(' '), printHorizontalLabel(N, N),
        printBoard(Board, N, 1).

printBoard([], N, _):-
        printRowDivider(N), nl.

printBoard([Line | Board], N, CurrentL) :-
        printRowDivider(N),
        printDesignRow(N),
        printVerticalLabel(CurrentL),
        put_code(9474),
        printLine(Line),
        printDesignRow(N),
        NewL is (CurrentL+1),
        printBoard(Board, N, NewL).

printLine([]) :- nl.
printLine([Head | Tail]) :-
        translate(Head, Symbol),
        write('  '),
        write(Symbol),
        write('  '), put_code(9474),
        printLine(Tail).

%             AESTHETICS

printRowDivider(N):-
        write(' '),
        put_code(9532),
        printRowDividerRec(N).

printRowDividerRec(0) :- nl.
printRowDividerRec(N) :-
        put_code(9472), put_code(9472), put_code(9472), put_code(9472),
        put_code(9472), put_code(9472), put_code(9472), put_code(9532),
        N1 is (N-1),
        printRowDividerRec(N1).

printDesignRow(N):-
        write(' '),
        put_code(9474),
        printDesignRowRec(N).
```

```
printDesignRowRec(0) :- nl.
printDesignRowRec(N) :-
        write('      '), put_code(9474),
        N1 is (N-1),
        printDesignRowRec(N1).

%Dictionary for Labels
getLabel(0, L):- L = 'A'.
getLabel(1, L):- L = 'B'.
getLabel(2, L):- L = 'C'.
getLabel(3, L):- L = 'D'.
getLabel(4, L):- L = 'E'.
getLabel(5, L):- L = 'F'.
getLabel(6, L):- L = 'G'.
getLabel(7, L):- L = 'H'.
getLabel(8, L):- L = 'I'.
getLabel(9, L):- L = 'J'.
getLabel(10, L):- L = 'K'.
getLabel(11, L):- L = 'L'.

printHorizontalLabel(0, _):- nl.
printHorizontalLabel(N, Total):-
        Pos is (Total-N),
        getLabel(Pos, L),
        write('   '), write(L), write(' '),
        N1 is (N-1),
        printHorizontalLabel(N1, Total).

printVerticalLabel(CurrentL):-
        CurrentL < 10,
        write(CurrentL),
        write(' ').

printVerticalLabel(CurrentL):-
        write(CurrentL).
```

2.3 Movimentos

Elencar os movimentos (tipos de jogadas) possíveis e definir os cabeçalhos dos predicados que serão utilizados (ainda não precisam de estar implementados).

Colocação de uma peça no tabuleiro: putPiece(Row, Column, Board, pieceType)

Movimentação de um worker: moveWorker(Row, Col, DestRow, DestCol, Board)

Só estes? xD