FABRIK

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Fabrik3:

André Cruz - 201503776
Edgar Carneiro - 201503748

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

12 de Novembro de 2017

Resumo

Este trabalho foi desenvolvido no âmbito da unidade curricular de Programação em Lógica, integrada no 3º ano do Mestrado Integrado em Engenharia Informática e Computação, tendo como objetivo aprofundar os conhecimentos adquiridos nas aulas teóricas e práticas desta unidade curricular, bem como a abordagem de problemas mais práticos com recurso à linguagem *PROLOG*. Assim, propusemo-nos a implementar o jogo *Fabrik*, com uma representação na linha de comandos, bem como modos de jogo **jogador vs jogador**, **jogador vs computador** e **computador vs computador**. Recorrendo à excelente bibliografia indicada pelo professores, [3] e [1], foi possível resolver todos os problemas que encontramos na implementação deste jogo, bem como uma aprendizagem contínua dos conceitos relacionados com este paradigma de programação.

Resumo sucinto do trabalho com 150 a 250 palavras (problema abordado, objetivo, como foi o problema resolvido/abordado, principais resultados e conclusões).

Conteúdo

1 Introdução

Descrever os objetivos e motivação do trabalho. Descrever num parágrafo breve a estrutura do relatório.

2 O Jogo *Fabrik*

2.1 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.

2.2 Material

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

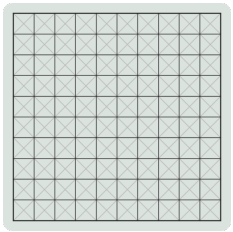

Figura 1: Tabuleiro vazio de 11 x 11 espaços

2.3 Regras

A implementação deste jogo foi baseada no manual de regras oficiais [2].

As pretas (jogador que joga com peças de cor preta) começam por colocar um dos trabalhadores num espaço à sua escolha. De seguida, as brancas (jogador que joga com peças de cor branca) colocam o outro trabalhador num espaço livre. De seguida, as pretas decidem quem começa por jogar.

O jogo procede por turnos, sendo que em cada turno um jogador pode, se assim optar, mover um dos trabalhadores para um espaço vazio. De seguida, o jogador deve jogar colocar uma das suas peças 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.

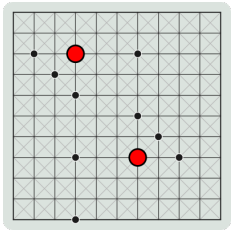

Figura 2: Pontos de interseção entre os dois trabalhadores

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.

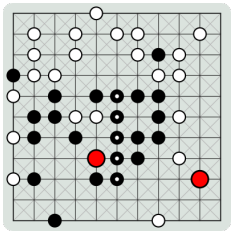

Figura 3: Final de uma partida de Fabrik, com vitórias das pretas

Referências:
https://spielstein.com/games/fabrik
https://spielstein.com/games/fabrik/rules

3 Lógica do Jogo

Descrever o projeto e implementação da lógica do jogo em Prolog, incluindo a forma de representação do estado do tabuleiro e sua visualização, execução de movimentos, verificação do cumprimento das regras do jogo, determinação do final do jogo e cálculo das jogadas a realizar pelo computador utilizando diversos níveis de jogo. Sugere-se a estruturação desta secção da seguinte forma:

3.1 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**:

```
[[none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none],
 [none, none, none, none, none, none, none, none, none, none, none]]
```

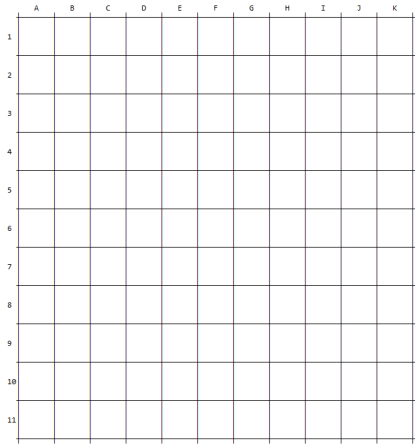

Figura 4: Representação do estado inicial na consola

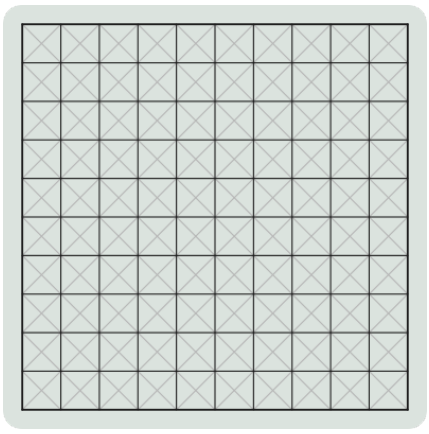

Figura 5: Tabuleiro original Vazio

Representação de um possível **estado intermédio**:

```
[[ none, none, none, none, none, none, none, none, none, none, none],
 [ none, none, none, none, none, white, worker, none, none, none, none],
 [ none, none, none, none, none, none, white, black, white, none, none],
 [ none, white, white, none, none, none, none, white, white, none, none],
 [white, none, none, none, black, black, none, black, none, none, none],
 [ none, none, black, white, none, none, none, none, white, none, none],
 [ none, black, none, none, none, black, black, black, none, none, none],
 [ none, none, none, none, none, none, black, none, white, none, none],
 [ none, none, none, none, none, black, none, none, none, none, none],
 [ none, none, none, none, worker, none, none, none, none, none, none],
 [ none, none, none, none, none, none, none, none, none, none, none]]
```

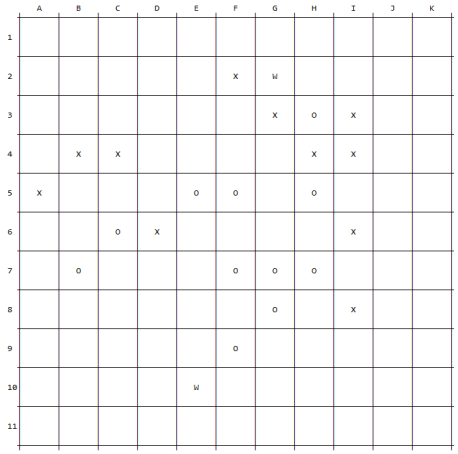

Figura 6: Representação na consola, de um possível estado intermédio

Representação de um possível **estado final**:

```
[[ none, none, none, none, white, none, none, none, none, none, none],
 [ none, white, none, white, none, white, white, none, none,white, none],
 [ none, white, none, white, none, none,white, black, white, none, none],
 [black, white, white, none, none, none, none, white, white, none, none],
 [white, none, black, none,black, black, black, black, none, none, none],
 [none, black, black, white,white, black, none, black,white, none, none],
 [white, black, none,black, none, black, black, black, none, none, none],
 [ none, none, none, none, worker, black, black, none,white, none, none],
 [white, black, none, none,black, black, none, none, none, worker, none],
 [ none, none, none, none, none, none, none, none, none, none, none],
 [ none, none, black, none, none, none, none, white, none, none, none]]
```

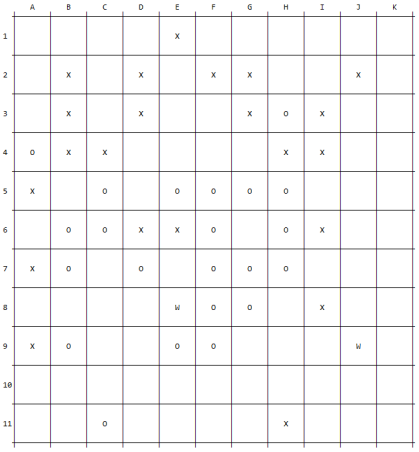

Figura 7: Representação do estado final na consola

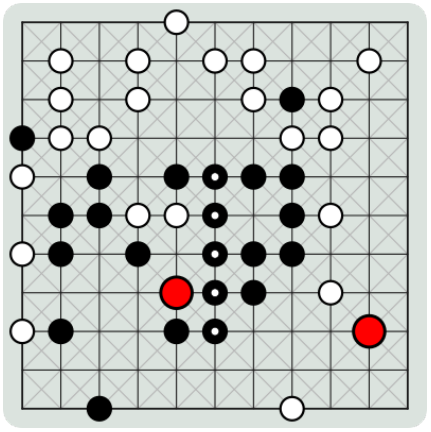

Figura 8: Representção do mesmo estado final, no tabuleiro original

3.2 Visualização do Tabuleiro

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

```
% Dictionary for Board Elements
translate(none, 32). %Empty Cell
translate(black, 79). %Dark Pieces
translate(white, 88). %White Pieces
translate(worker, 9608). %Red Workers

(...)

% General PrintBoard
printBoard(Board):-
        boardSize(N),
        printBoard(Board, N), !.

% Board Printing - arguments: Board and Board size
printBoard(Board, N):-
        clearConsole,
        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, Code),
        write('  '),
        put_code(Code),
        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, 'A').
getLabel( 1, 'B').
getLabel( 2, 'C').
getLabel( 3, 'D').
getLabel( 4, 'E').
getLabel( 5, 'F').
getLabel( 6, 'G').
getLabel( 7, 'H').
getLabel( 8, 'I').
getLabel( 9, 'J').
getLabel(10, 'K').
getLabel(11, 'L').
getLabel(_,_):-
        write('Error: Unrecognized Label.'), nl,
        fail.

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).
```

Representação de um tabuleiro, usando o código mencionado:

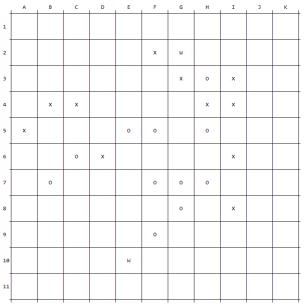

Figura 9: Representação de um tabuleiro na consola

3.3 Lista de Jogadas Válidas

Obtenção de uma lista de jogadas possíveis. Exemplo: *valid_moves(+Board, -ListOfMoves).*

3.4 Execução de Jogadas

Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo. Exemplo: *move(+Move, +Board, -NewBoard)*.

3.5 Avaliação do Tabuleiro

Avaliação do estado do jogo, que permitirá comparar a aplicação das diversas jogadas disponíveis. Exemplo: *value(+Board, +Player, -Value)*.

3.6 Final do Jogo

Verificação do fim do jogo, com identificação do vencedor. Exemplo: *game_over(+Board, -Winner).*

3.7 Jogada do Computador

Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade. Por exemplo: *choose_move(+Level, +Board, -Move)*.

4 Interface com o Utilizador

Descrever o módulo de interface com o utilizador em modo de texto.

5 Conclusões

Que conclui deste projecto? Como poderia melhorar o trabalho desenvolvido?

Bibliografia

[1] D. Stein, "Fabrik a 'worker placement' abstract."

[2] L. Sterling, E. Y. Shapiro, and D. H. D. Warren, *The Art of Prolog: Advanced Programming Techniques*. MIT Press, 2010.

[3] M. Carlsson and T. Frühwirth, *SICStus Prolog User's Manual*. SICS Swedish ICT AB, 4.3.5 ed., 2016.

A Nome do Anexo

Código Prolog implementado devidamente comentado e outros elementos úteis que não sejam essenciais ao relatório.