Distrify

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Distrify_4:
João Guarda - 201303463
Ricardo Lopes - 201303933

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

11 de Outubro de 2015

1 O Jogo Distrify

[1] [2] [3] Distrify é um jogo de tabuleiro, combinatorial e de conexão. Foi concebido a 22 de Agosto de 2015 por David Stoner. É constituído por um tabuleiro quadrado de 9x9 a 19x19 quadrículas e por duas peças de cor diferente (branco e preto).

Regras

O jogador com as peças de cor preta começa a jogar posicionando uma peça numa qualquer quadrícula do tabuleiro. Seguidamente, existem duas possibilidades:

1. Colocar uma peça numa qualquer quadrícula desde que não forme um ***crosscut***.
2. Colocar duas peças em quadrículas vazias, desde que não sejam adjacentes diagonalmente entre elas e não resultem num ***triplet*** ou ***crosscut***.

De realçar que **haverá sempre uma jogada válida**, pelo que os jogadores **não podem passar a vez**.

Conceitos

- ***crosscut***: quando se verifica uma cruz constituída pelo mesmo número de peças de ambos os jogadores. Vi-de fig.1.
- ***triplet***: quando 3 peças estão seguidas na horizontal, na vertical ou diagonalmente.
- Duas peças da mesma cor são consideradas **ligadas** quando estão adjacentes horizontalmente, verticalmente ou diagonalmente.

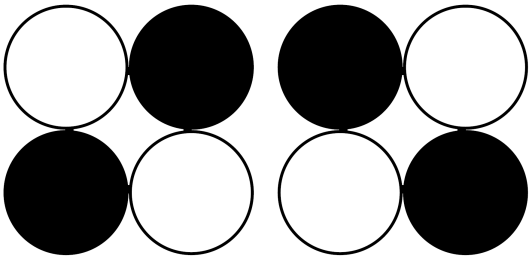


Figura 1: Padrão de *crosscut*

[1] `https://boardgamegeek.com/boardgame/182752/distrify`

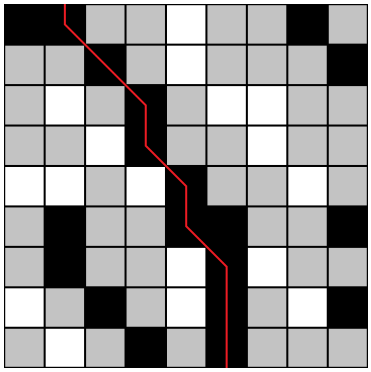
[2] `https://boardgamegeek.com/filepage/121580/distrify-sample-game-9x9` (Descarregar DistrifyGame.pdf)

[3] `https://boardgamegeek.com/filepage/121566/distrify-rules` (Descarregar Distrify.pdf)

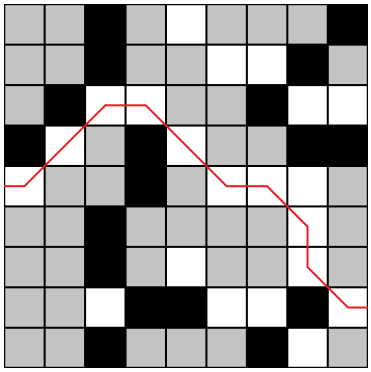
Objetivo

O jogador com peças pretas ganha se, a qualquer momento, se verificar um caminho contínuo de peças ligadas entre elas desde o topo do tabuleiro até à base (Fig.2 (a)).

O jogador com peças brancas ganha se a qualquer momento se verificar um caminho contínuo de peças ligadas entre elas desde o lado esquerdo até ao lado direito do tabuleiro (Fig.2 (b)).



(a) Caminho (a vermelho) de vitória das peças pretas.



(b) Caminho (a vermelho) de vitória das peças brancas.

Figura 2: Exemplos de vitórias.

2 Representação do Estado do Jogo

Neste jogo a implementação mais simples para representar o tabuleiro é uma lista de listas. Cada lista da lista corresponde a uma linha do tabuleiro. Com esta implementção é mais fácil aceder às diferentes linhas do tabuleiro, visto que em Prolog a visita em listas é feita recursivamente.

Representação do estado inicial do tabuleiro:

```
1  emptyBoard([
2          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
3          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
4          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
5          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
6          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
7          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
8          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
9          [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell],
10         [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
               emptyCell, emptyCell, emptyCell]]).
```

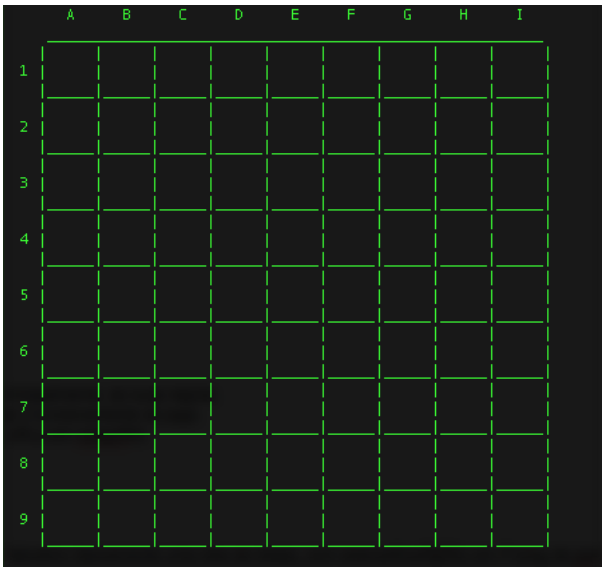


Figura 3: Estado Inicial do Tabuleiro na consola

Representação de um estado intermédio do tabuleiro:

```
intermediumBoard([
        [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
            emptyCell, white, emptyCell],
        [emptyCell, emptyCell, white, emptyCell, white, white, emptyCell,
            black, white],
        [emptyCell, emptyCell, emptyCell, black, emptyCell, black, white,
            emptyCell, black],
        [emptyCell, white, black, emptyCell, white, black, emptyCell, black,
             emptyCell],
        [emptyCell, emptyCell, black, white, black, emptyCell, emptyCell,
            emptyCell, emptyCell],
        [emptyCell, white, emptyCell, emptyCell, emptyCell, black,
            emptyCell, emptyCell, emptyCell],
        [emptyCell, emptyCell, black, emptyCell, white, black, emptyCell,
            emptyCell, emptyCell],
        [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, white,
            black, white, emptyCell],
        [emptyCell, emptyCell, emptyCell, emptyCell, emptyCell, emptyCell,
            emptyCell, emptyCell, emptyCell]]).
```

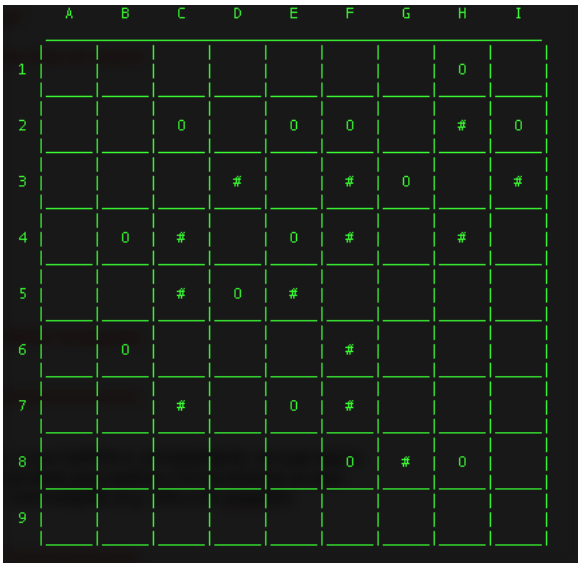


Figura 4: Um Estado Intermédio do tabuleiro na consola

Representação de um estado final do tabuleiro:

```
finalBoard([
        [emptyCell, emptyCell, emptyCell, black, emptyCell, emptyCell, black, white, emptyCell],
        [emptyCell, emptyCell, white, black, white, white, emptyCell, black, white],
        [emptyCell, white, emptyCell, black, white, black, white, emptyCell, black],
        [emptyCell, white, black, emptyCell, white, black, white, black, emptyCell],
        [emptyCell, emptyCell, black, white, black, emptyCell, emptyCell, white, black],
        [emptyCell, white, black, emptyCell, white, black, emptyCell, black, emptyCell],
        [emptyCell, emptyCell, black, emptyCell, white, black, emptyCell, black, emptyCell],
        [emptyCell, white, white, white, emptyCell, white, black, white, emptyCell],
        [emptyCell, emptyCell, black, emptyCell, emptyCell, emptyCell, black, emptyCell, emptyCell]]).
```

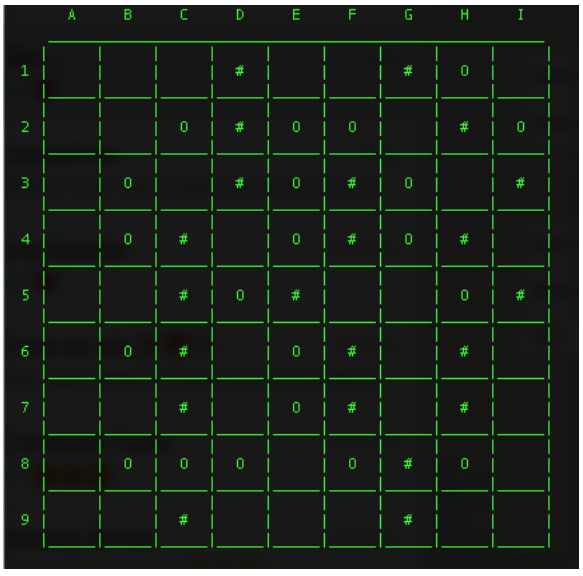


Figura 5: Um Estado Final do Tabuleiro na consola

3 Visualização do Tabuleiro

Para obter a visualização de um tabuleiro basta utilizar o predicado Prolog **printBoard(Board)**. Por sua vez, este utiliza os seguintes predicados:

```
1  rowIdentifiers ([' 1 ',' 2 ',' 3 ',' 4 ',' 5 ',' 6 ',' 7 ',' 8 ',' 9 ']).
2
3  getSymbol(emptyCell, ' ').
4  getSymbol(white, 'O').
5  getSymbol(black, '#').
6
7
8  printColumnId :- write('    A   B   C   D   E   F   G   H   I'),nl.
9  printInitialSeparator :- write('
                 ---------------------------------------------------       '),nl.
10 printMiddleSeparator :- write('   |       |       |       |       |       |       |       |       |'),nl.
11 printFinalSeparator :- write('   | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |'),nl.
12
13 printCell(Char) :- write('| '), write(Char), write(' ').
14 printBoardLine([]) :- write('|'), nl, printFinalSeparator.
15 printBoardLine([Head|Tail]) :- getSymbol(Head,Char), printCell(Char),
       printBoardLine(Tail).
16
17 printBoardAux([],[]) :- nl.
18 printBoardAux([Head|Tail], [RowId|RowTail]) :- printMiddleSeparator, write
       (RowId), printBoardLine(Head), printBoardAux(Tail, RowTail).
19
20 printBoard(Board) :- printColumnId, printInitialSeparator, rowIdentifiers(
       RowId), printBoardAux(Board, RowId).
```

A chamada deste predicado produz então o seguinte *output*:

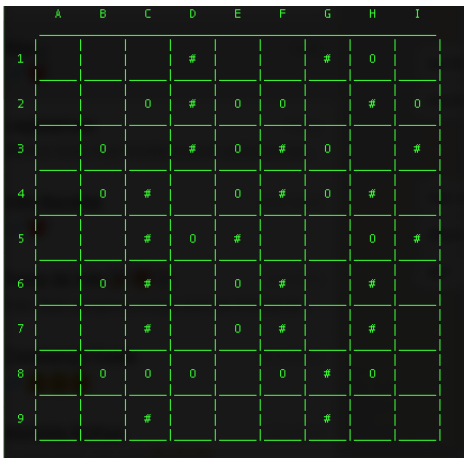


Figura 6: Output do predicado de visualização

4 Movimentos

Em cada movimento o jogador pode colocar duas peças no tabuleiro. Para isso basta utilizar o seguinte predicado Prolog:

placePiece(Board, Player, Row, Column, NewBoard).

A variável Board é uma lista de listas com o actual estado do jogo. A variável Player é uma variável que nos indica que jogador está a colocar a sua peça no tabuleiro. Row e Column indicam em que posição no tabuleiro irá ser colocada a peça. Já a NewBoard é a variável que irá conter o novo estado do jogo.