Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Ni-Ju

Programação em lógica

Relatório Intercalar - TP1

Ana Cláudia Fonseca Santos - 200700742
Ana Margarida Oliveira Pinheiro da Silva - 201505505

14 de Outubro de 2017

Índice

1 Descrição do jogo **2**
- 1.1 História . . . 2
- 1.2 Peças . . . 2
- 1.3 Objetivo . . . 3
- 1.4 Instruções . . . 3
- 1.5 Fim do jogo . . . 5

2 Representação do estado do jogo **6**
- 2.1 Cada Peça . . . 6
- 2.2 Tabuleiro . . . 7

3 Visualização do tabuleiro **10**

4 Movimentos **12**

1 Descrição do jogo

1.1 História

O "Ni-ju", que significa "20" em japonês, é um jogo de tabuleiro concebido por Néstor Romeral Andrés e foi lançado em 2016. É um jogo para dois jogadores no qual estes colocam as peças de jogo em padrões específicos. Para além disso, situa-se na categoria de estratégia abstrata, isto é, não tem tema, cada jogada é direta/mecânica e não contêm nenhum elemento de sorte/ocorrência aleatória.

1.2 Peças

Cada jogador tem 20 peças (figura 1) e cada peça tem um padrão diferente, daí o nome do jogo (20). Existem 70 padrões diferentes se incluírmos as rotações. Existem também 40 discos vermelhos.

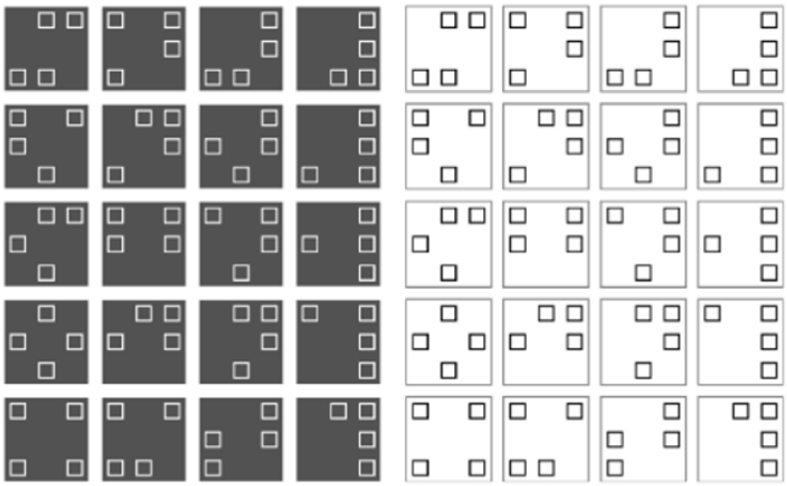


Figura 1: Peças do jogo

1.3 Objetivo

Um jogador ganha quando conseguir cercar uma das suas peças em jogo por pelo menos 4 da sua cor, em que o seu padrão aponta a direção dessas peças, como se verifica na figura 2.

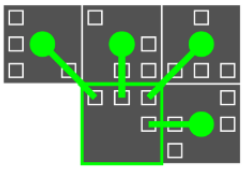
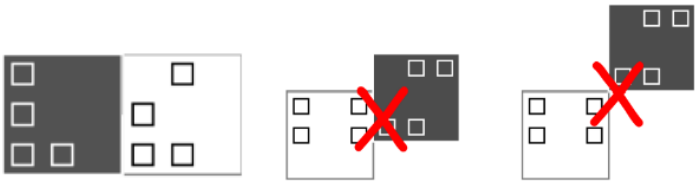


Figura 2: Como ganhar

1.4 Instruções

O "tabuleiro" começa vazio e cada jogador escolhe a sua cor (branco ou preto). A primeira jogada é realizada por aquele que tiver as peças brancas, sendo esta peça colocada numa posição aleatória sobre a mesa. As próximas jogadas tem como intuito alinhar as peças de forma a concretizar o objetivo já referido no ponto anterior, mas também deverão impedir o adversário de ganhar. As jogadas possíveis são sempre referentes às peças já colocadas (excepto a primeira), nas quais têm se ser obrigatoriamente alinhadas de forma horizontal e vertical (figura 3).



(a) Legal (b) Ilegal

Figura 3: Exemplo de jogadas

Se acordado entre os jogadores, existe a possibilidade de ser colocado um pequeno disco vermelho sobre as peças em jogo que já não permitem desencadear a jogada da vitória. Adicionado assim clareza ao estado atual de jogo. Na fig. 5 apresentamos um exemplo de quando colocar um disco vermelho, neste caso a peça branca contém um quadrado do padrão "bloqueado" por uma peça preta, não podendo ser ligado nem na horizontal nem na diagonal a outro quadrado de uma peça branca.

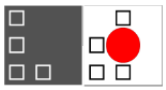


Figura 4: Exemplo de colocação do disco vermelho

Se as peças dos jogadores se esgotarem sem que nenhum tenha obtido sucesso estes podem continuar, jogando as peças que já se encontravam no tabuleiro, desde que estas tenha pelo menos um dos lados "livres", ou seja, sem peças. Se houver, é necessário remover os discos vermelhos.

1.5 Fim do jogo

Exemplo de uma jogada da figura 5: o jogador das peças pretas acaba de colocar a peça destacada a verde, conseguindo ligar cada um dos quadrados desta peça as outras pretas já no tabuleiro exceptuando a do canto superior. O jogador das peças brancas tenta bloquear esta jogada colocando uma peça no local do ponto verde.

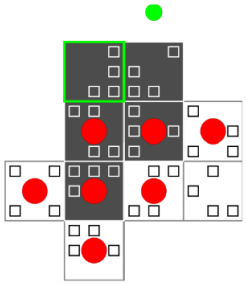


Figura 5: Exemplo de uma jogada (parte 1)

A peça preta em questão é bloqueada porém o jogador das peças pretas na sua vez de jogar consegue chegar à vitória através da peça com o asterisco verde.

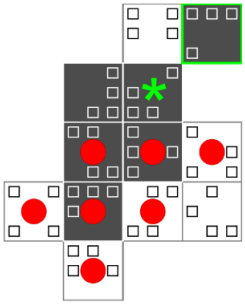


Figura 6: Exemplo de uma jogada (parte 2)

2 Representação do estado do jogo

2.1 Cada Peça

Informação referente a cada peça:

- Peça
  - Código
    - a
    - b
    - ...
    - t
  - Posição
    - 0 - 0º
    - 1 - 90º
    - 2 - 180º
    - 3 - 270º
  - Cor
    - 0 - Preta
    - 1 - Branca
  - Validade
    - 0 - Válida
    - 1 - Inválida

Figura 7: Peça preta com código T

Exemplo:
Representação da peça preta, na posição inicial e válida (figura 7):

```
piece([1  0  1],
      [0  0  0],
      [1  0  1]).
```

Informação a guardar:

```
pieceInfo([t,0,0,0]).
```

2.2 Tabuleiro

Estado inicial do tabuleiro:

```
[[empty]];
```

Estado do tabuleiro após colocação da primeira peça:

```
[[[empty], [empty],    [empty]],
 [[empty], [t,0,1,0], [empty]],
 [[empty], [empty],    [empty]]];
```

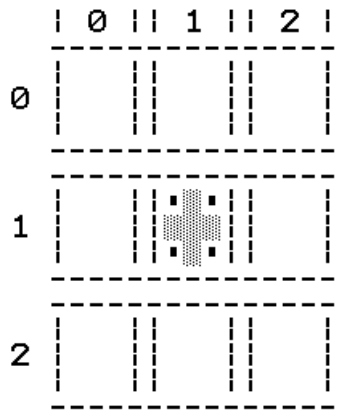


Figura 8

Possível estado intermédio do tabuleiro:

```
[[[empty], [empty],    [empty],    [empty]],
 [[empty], [t,0,1,0], [a,1,0,0], [empty]],
 [[empty], [k,2,1,0], [empty],    [empty]],
 [[empty], [empty],    [empty],    [empty]]];
```

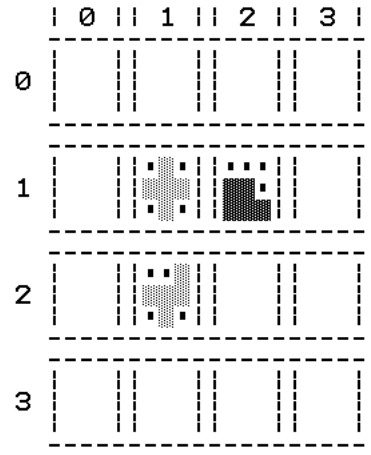


Figura 9

Possível estado final do tabuleiro:

```
[[[empty],[empty],  [empty],     [empty],     [empty],     [empty],     [empty]],
 [[empty],[empty],  [empty],     [empty],     [s, 0, 0, 0],[empty],     [empty]],
 [[empty],[j,0,1,0],[empty],     [empty],     [j, 0, 0, 0],[empty],     [empty]],
 [[empty],[i,0,1,0],[p, 0, 1, 0],[b, 0, 1, 1],[t, 0, 0, 1],[p, 0, 0, 0],[empty]],
 [[empty],[empty],  [empty],     [empty],     [o, 0, 1, 0],[empty],     [empty]],
 [[empty],[empty],  [empty],     [empty],     [empty],     [empty],     [empty]]];
```

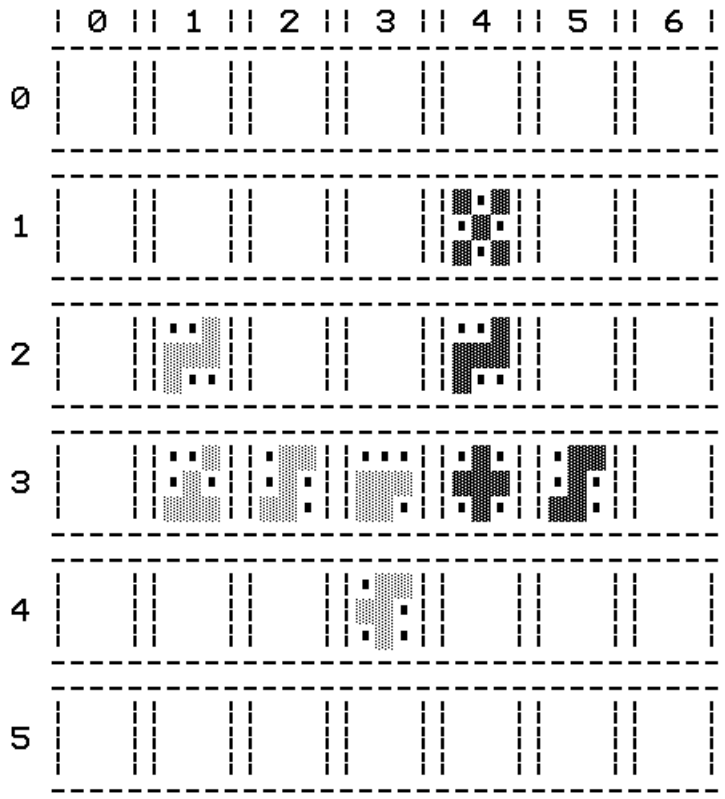


Figura 10: Representação de um fim de jogo onde a peça branca colocada na linha 3 e na coluna 2 ganhou.

3 Visualização do tabuleiro

Para a visualização dos tabuleiros representados nas figuras 8, 9 e 10 foi usado o seguinte código, utilizando para isso o predicado *printBoard(Board).*

```
% Get Symbol
getSymbol(0, 0, 178).
getSymbol(1, 0, 254).
getSymbol(0, 1, 176).
getSymbol(1, 1, 254).

validSymbol(0, 255).
validSymbol(1, 157).
validSymbol([Head | Tail], Valid):- validSymbol(Head, Valid).

% Get color
getPieceInfo([Head | Tail], Color, Valid):- Color = Head,
        validSymbol(Tail, Valid).

% Get pattern
getPiecePattern(PieceNum, Letter, Pattern):-
        patternLetter(Letter, TempPattern),
        nth0(PieceNum, TempPattern, Pattern).

% Prints the symbols
printEachSymbol([],_,_).
printEachSymbol([Head | Tail], Color, Valid):-getSymbol(Head, Color, Char),
        put_code(Char), printEachSymbol(Tail, Color, Valid).
printPieceSymbols(PieceNum, Letter, Color, Valid):-
        getPiecePattern(PieceNum, Letter, Pattern),
        printEachSymbol(Pattern, Color, Valid).

% Prints each piece
printPiece([], PieceNum).
printPiece(nil, PieceNum) :- write('|___|').
printPiece([Letter, Rotation | Tail], PieceNum):-
        getPieceInfo(Tail, Color, Valid), write('|'),
        printPieceSymbols(PieceNum, Letter, Color, Valid), write('|').

% Prints row with pieces
printRowPieces([],Num,PieceNum):- nl.
printRowPieces([Head | Tail], Num, PieceNum):-
        printPiece(Head, PieceNum), printRowPieces(Tail, Num, PieceNum).
```

```
% Prints row's
printRow(0,_,_,_).
printRow(2,PieceNum,Row,RowNumber):-
        NewPieceNum is PieceNum + 1, write('␣'), write(RowNumber), write('␣'),
        printRowPieces(Row, 1, NewPieceNum), printRow(1, NewPieceNum,Row, RowNumber).
printRow(Num,PieceNum,Row, RowNumber):- NewNum is Num - 1,
        NewPieceNum is PieceNum + 1, write('␣␣␣'),
        printRowPieces(Row, NewNum, NewPieceNum),
        printRow(NewNum, NewPieceNum,Row,RowNumber).

% Board numbers for coordinates and separators
printTopNumbers(_, 0).
printTopNumbers(Count, ColumnsNum):- write('|␣'), write(Count),
        write('␣|'), NewCount is Count + 1, NewColumnsNum is ColumnsNum - 1,
        printTopNumbers(NewCount, NewColumnsNum).

% Prints separators
printSeparator(0).
printSeparator(ColumnsNum):- write('————'), NewColumnsNum is ColumnsNum - 1,
        printSeparator(NewColumnsNum).

prepareBoard([Head | Tail]):- nl,length(Head, ColumnsNum), write('␣␣␣'),
        printTopNumbers(0,ColumnsNum), nl, printBoard([Head | Tail], 0).

% Main function for print game board
printBoard([], _).
printBoard([Head | Tail], RowCount) :- length(Head, ColumnsNum), write('␣␣␣'),
        printSeparator(ColumnsNum), nl, printRow(3,-1,Head, RowCount),
        NewRowCount is RowCount + 1, write('␣␣␣'),printSeparator(ColumnsNum), nl,
        printBoard(Tail, NewRowCount).
```

4 Movimentos

```
addPiece(Board, Row, Column, PieceCode, Color, Rotation, NewBoard,
        NewColor).
```

Na primeira fase do jogo, quando existem peças disponiveis será usado este predicado. Destina-se a adicionar peças ao tabuleiro em uma jogada, em que o 'Board' é a estrutura que contém o estado atual de jogo, 'Row' e 'Column' são as posições escolhidas para colocar a peça. Já para esta é necessário saber o seu código ('PieceCode'), a sua 'Color' e a sua 'Rotation'. Por sua vez, as variaveis 'NewBoard' e 'NewColor' destinam-se a guardar os dados de saída necessários para a próxima jogada: o novo tabuleiro gerado e o proximo jogador a jogar.

```
movePiece(Board, SourceRow, SourceColumn, PieceCode, Color,
        Rotation, DestRow, DestColumn, NewBoard, NewColor).
```

Porém, se deixar de haver peças disponiveis será necessário um novo predicado. Este tem como objetivo retirar uma peça já colocada no tabuleiro e move-la para uma nova posição. As variaveis para os dados de entrada serão: 'Board', 'SourceRow', 'SourceColumn', 'PieceCode', 'Color', 'Rotation', 'DestRow' e 'DestColumn'. Mantendo-se as variaveis para dados de saída tal como no predicado anterior.