Ni-ju

Relatório Intercalar

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo xx:
João Pedro Furriel de Moura Pinheiro - up201104913
Ventura de Sousa Pereira - up201404690

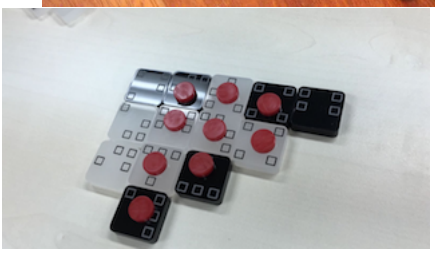
Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

12 de Outubro de 2017

1 O Jogo Ni-ju

Ni-ju (traduzido para 20, em japonês) é um jogo que desenvolvido pelo designer e artista Néstor Romeral Andrés e artista, em 2016, tendo sido publicado pela HenMar Games, nestorgames. O jogo é constituido por 40 peças, sendo estas divididas por cor - branca e preta, entre os jogadores, ficando, cada um, com 20 peças de padrões diferentes (daí o nome do jogo). Será possivel ter 70 padrões diferentes, se incluirmos rotações. Para comelar, cada jogador deverá colocar uma peça na zona de jogo, seguido pelo seu adversário, até as condições de vitória se reunirem. O objetivo do jogo é recriar o padrão descrito na peça, à volta da mesma. No final ganha quem conseguir recriar mais padrões, quando se esgotarem as peças.

Regra: Na sua vez o jogador apenas poderá colocar a sua peça adjacente a uma das já jogadas.

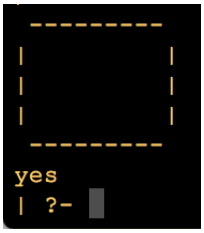


2 Representação do Estado do Jogo

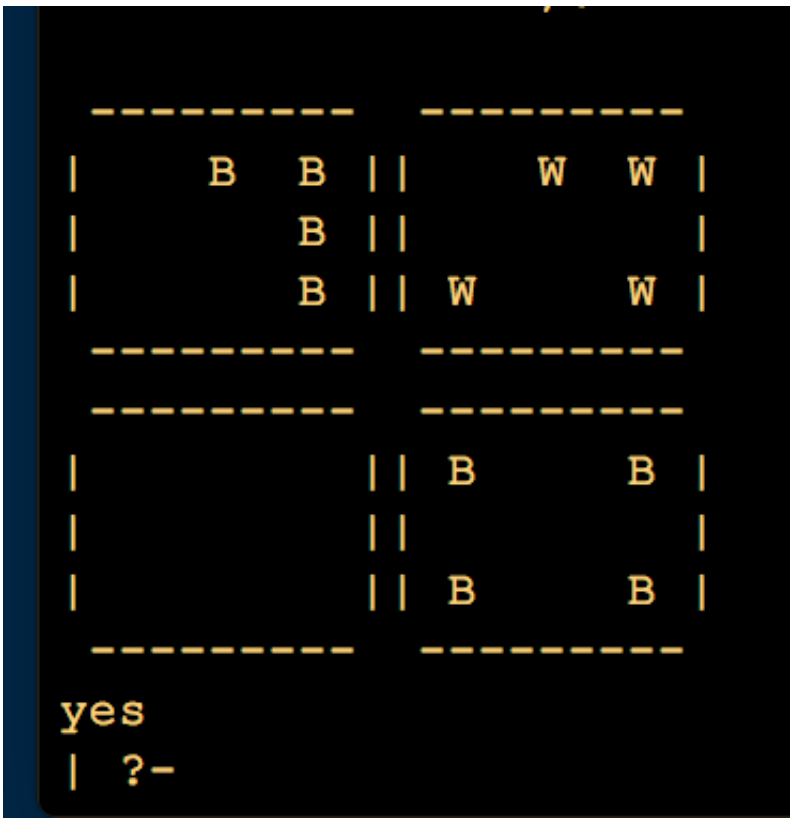
Inicialmente, o estado inicial do jogo será uma lista que represente um espaço vazio, visto que este é dinâmico. Modifica-se conforme a posição de cada peça jogada.

Estado Inicial:

[[[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1]]]]



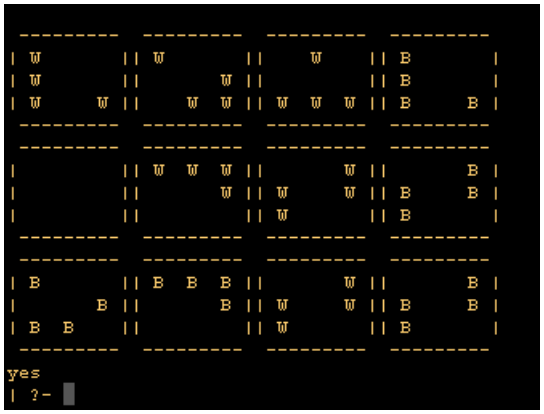
Num estado intermédio, o jogo terá algumas peças em jogo.
Possível representação:
[[
[[0, 1, 1], [0, 0, 1], [0, 0, 1]],
[[0, 2, 2], [0, 0, 0], [2, 0, 2]]
],
[
[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1]],
[[1, 0, 1], [0, 0, 0], [1, 0, 1]]
]
]



O estado final é quando todas as peças já foram colocadas. Num hipotético jogo apenas com 11 peças, este seria um possível estado final.

Representação final:

```
[
[
[[2, 0, 0], [2, 0, 0], [2, 0, 2] ],
[[2, 0, 0], [0, 0, 2], [0, 2, 2] ],
[[0, 2, 0], [0, 0, 0], [2, 2, 2] ],
[[1, 0, 0], [1, 0, 0], [1, 0, 1] ]
],
[
[[-1, -1, -1], [-1, -1, -1], [-1, -1, -1] ],
[[2, 2, 2], [0, 0, 2], [0, 0, 0] ],
[[0, 0, 2], [2, 0, 2], [2, 0, 0] ],
[[0, 0, 1], [1, 0, 1], [1, 0, 0] ]
],
[
[[1, 0, 0], [0, 0, 1], [1, 1, 0] ],
[[1, 1, 1], [0, 0, 1], [0, 0, 0] ],
[[0, 0, 2], [2, 0, 2], [2, 0, 0] ],
[[0, 0, 1], [1, 0, 1], [1, 0, 0] ]
]
]
```



3 Visualização do Tabuleiro

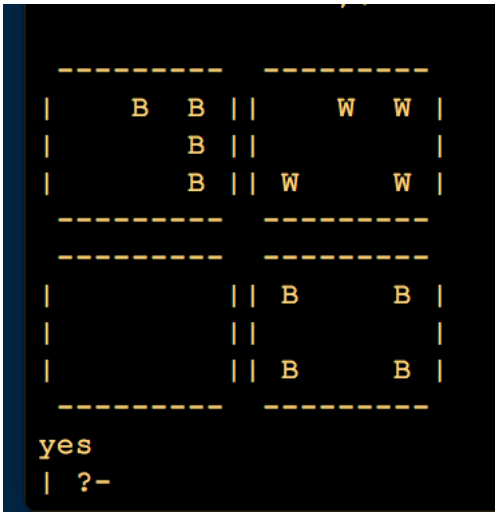
O padrão de cada peça é representado por B's ou W's, dependendo se a peça é preta ou branca, respetivamente. Os espaços vazios no tabuleiro são representados por uma matriz, de 3x3, de espaços. Além disso, os espaços vazios da peça são, também, representados por espaços.

Predicados usados para a impressão do tabuleiro:

```
printElement(X) :- X =:= 1, write(' B ').
printElement(X) :- X =:= 2, write(' W ').
printElement(X) :- X =:= -1, write('   ').
```

```
printElement(X) :- X =:= 0, write('     ').
printPieceRowSeparation(BoardRow) :-

        BoardRow = [].

printPieceRowSeparation(BoardRow) :-

        BoardRow \= [],
        [\_ | Rest] = BoardRow,
        write(' ————————— '),
        printPieceRowSeparation(Rest).

printPieceRow([X1,X2,X3]) :-

        write('|'),
        printElement(X1),
        printElement(X2),
        printElement(X3),
        write('|').


printBoard(Board) :-

        Board = [].

printBoard(Board) :-

        [Row | Rest] = Board,
        printPieceRowSeparation(Row),
        nl,
        printPiecesRow1(Row),
        nl,
        printPiecesRow2(Row),
        nl,
        printPiecesRow3(Row),
        nl,
        printPieceRowSeparation(Row),
        nl,
        printBoard(Rest).

printPiecesRow1(BoardRow) :-

        BoardRow = [].

printPiecesRow1(BoardRow) :-

        [Piece | Rest] = BoardRow,
        [PieceRow1 | \_] = Piece,
        printPieceRow(PieceRow1),
        printPiecesRow1(Rest).
```

```
printPiecesRow2(BoardRow) :-
        BoardRow = [].

printPiecesRow2(BoardRow) :-
        [Piece | Rest] = BoardRow,
        [\_,PieceRow2,\_] = Piece,
        printPieceRow(PieceRow2),
        printPiecesRow2(Rest).


printPiecesRow3(BoardRow) :-

        BoardRow = [].

printPiecesRow3(BoardRow) :-

        [Piece | Rest] = BoardRow,
        [\_,\_,PieceRow3] = Piece,
        printPieceRow(PieceRow3),
        printPiecesRow3(Rest).
```



4 Movimentos

Antes de cada jogada, os jogadores podem rodar as suas peças, para esse efeito os predicados a usar serão:

```
RotatePieceLeft(OldPiece, NewPiece)
RotatePieceright(OldPiece, NewPiece)
```

Depois da rotação os jogadores deverão posicionar as peças no tabuleiro numa posição adjacente a uma das peças já existentes no tabuleiro. Para tal, os predicados serão:

```
playPieceRight(CurrentBoard, Piece, Row, Column, ResultingBoard).
playPieceDown(CurrentBoard, Piece, Row, Column, ResultingBoard).
```

```
playPieceLeft(CurrentBoard, Piece, Row, Column, ResultingBoard).
playPieceUp(CurrentBoard, Piece, Row, Column, ResultingBoard).
```