FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

PROGRAMAÇÃO EM LÓGICA – TP1 – NI-JU2

JOÃO PEDRO FURRIEL DE MOURA PINHEIRO – UP201104913
VENTURA DE SOUSA PEREIA – UP201404690

12 DE NOVEMBRO DE 2017

RESUMO

O objetivo do trabalho era implementar um jogo de tabuleiro utilizando a linguagem PROLOG. O objetivo final era apresentar um jogo que pode ser jogado na consola e que simula o jogo real de tabuleiro. O problema foi abordado dividindo o desenvolvimento em 4 partes fundamentais, a saber: - Utilidades, Impressões, Lógica de Jogo e Interface. Nas utilidades foram desenvolvidos predicados auxiliares a ser utilizados noutras partes, a Impressão trata de mostrar o estado do jogo, entre outras coisas na consola, a lógica do jogo controla as regras, a vez do jogo, ente outras coisas e a interface serve como meio de ligação entre o jogo e os jogadores. O jogo desenvolvido tem a vertente de humano contra humano, humano contra máquina em três níveis de dificuldade diferentes, e ainda é possível assistir a um jogo exclusivamente jogado por duas máquinas. Os resultados obtidos foram muito satisfatórios, tendo-se chegado à conclusão que a abordagem de programação em lógica para o desenvolvimento deste tipo de jogos é muito adequada.

Índice

1. INTRODUÇÃO

O objetivo deste trabalho prendia-se com a implementação do jogo NI-JU, um jogo de tabuleiro com o objetivo de formar padrões num tabuleiro dinâmico. Desta forma, este relatório pretende descrever de forma sucinta todo o processo de desenvolvimento do mesmo. Começa-se por descrever o jogo e as suas regras, depois a lógica que foi usada para implementar os mesmos e por fim descreve-se as conclusões do trabalho.

2. O JOGO NI-JU

O jogo Ni-ju (20, em japonês) é um jogo desenvolvido pelo designer e artista Nestor Romeral Andrés, em 2016, tendo sido publicado pela HenMar Games, nestorgames

O jogo é constituído por 40 peças, sendo estas divididas por cor branca e preta. Cada peça tem um padrão desenhado sempre por 4 quadrados desenhados.

Figura 1 – Peças pretas

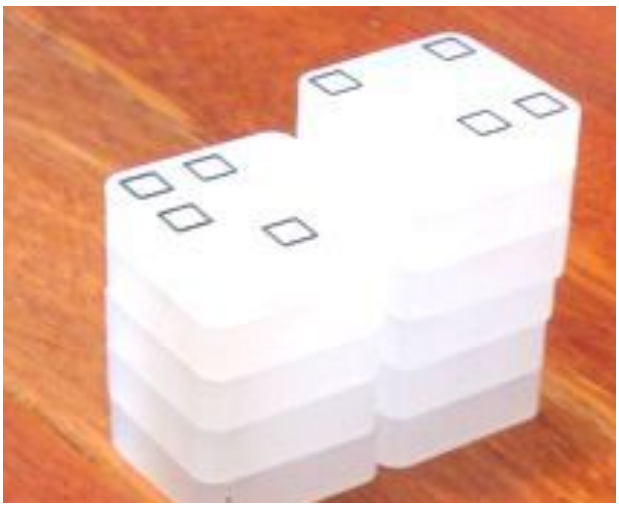


Figura 2 – Peças Brancas

De uma forma mais esquemática, as peças podem ser vistas como matrizes de 3 por 3 em que o elemento do centro (que nunca está desenhado porque não pertence ao padrão) representa a peça e os restantes elementos o padrão da peça:

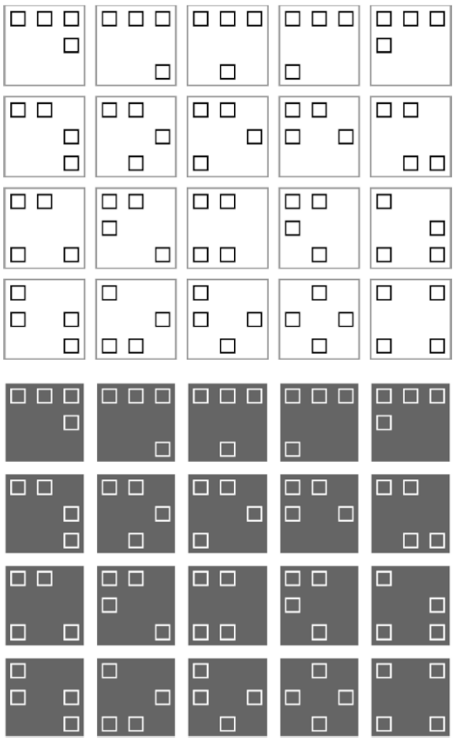


Figura 3 – Representação Esquemática das Peças

Cada peça representa um padrão desenhado com quadrados. Cada jogador deverá colocar uma peça na zona de jogo, seguido pelo seu adversário. O jogador pode colocar a peça com a rotação desejada, desde que virada para cima. O objetivo do jogo é formar padrões à volta das peças com peças do jogador. Como exemplo de um padrão conseguido, temos a seguinte situação:

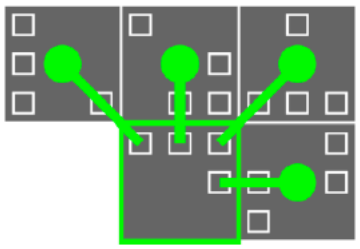


Figura 4 - Peça com padrão conseguido

Como forma de defesa, o jogador adversário, deverá tentar evitar esta situação. No caso que vemos na figura 4, se o jogador das peças brancas houvesse colocado uma das suas peças em qualquer uma das posições representados na figura com a pinta verde, o jogador das peças pretas não teria conseguido formar o padrão.

O jogo começa com a colocação de uma peça, e alternadamente, os jogadores vão colocando as peças no tabuleiro. Para jogar uma peça, esta terá de ser colocada numa posição adjacente a uma peça já existente no tabuleiro. Poderá ser colocado à direita, esquerda, em cima ou em baixo de uma peça já colocada. Desta forma, nunca poderá haver uma peça isolada no tabuleiro, ou apenas na diagonal de outras peças.

Na versão do jogo que se implementou, o jogo termina quando se esgotarem as peças dos dois jogadores. No final ganha o jogador que conseguir recriar mais padrões. Nesta versão que se desenvolveu, não foram usados os discos de ajuda. Existem outras variações do jogo que podem ser encontrada no manual, no entanto, escolheu-se implementar esta versão do jogo Ni-Ju.

3. LÓGICA DO JOGO

Nesta secção, vai ser descrita a forma como se abordou o problema na linguagem PROLOG e de que forma se conseguiu implementar as regras do jogo, o estado do mesmo e as condições de vitória e derrota. Decidiu-se que, não sendo uma questão revelante, o jogador 1 ficaria sempre associado às peças brancas e o jogador 2 às peças pretas. Também foi definido que a primeira jogada é sempre realizada pelo jogador 1. No início os jogadores introduzem os nomes, no caso do computador, os nomes estão já associados. Para iniciar o jogo, chamar o predicado start.

3.1. REPRESENTAÇÃO DO ESTADO DO JOGO

Em cada momento, o estado do jogo é composto por 3 componentes, a saber:

- Estado das peças do jogador 1
- Estado das peças do jogador 2
- Estado do tabuleiro (ou mesa)

Para representar qualquer um destes componentes, é necessário representar a peça. A peça é representada por uma matriz de 3 por 3 em que o elemento do meio (linha1, coluna 1) representa a cor da peça. Todas as peças foram representadas pelo predicado 'piece' da seguinte forma:

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

Neste caso, temos a peça numero 1, a peça que podemos ver na figura 4 (a que não tem a pinta verde).

As peças são construídas no início do jogo, substituindo o átomo x pelo átomo que representa o jogador 1 ou jogador 2. Ao construir as peças, se for uma peça do jogador 1 o átomo x é substituído por 1, se for o jogador 2 é substituído por 0. O inteiro que representa o número da peça é importante para o caso de se querer fazer uma partida com menos do que as 20 peças por jogador. Como exemplo, a peça número 1 do jogador seria representada pela seguinte lista: '[[1,1,1],[0,1,1],[0,0,0]]', ao passo que a mesma peça do jogador 2 seria representada pela lista '[[1,1,1],[0,0,1],[0,0,0]]'.

Assim sendo, o estado das peças de cada jogador será representado por uma lista das suas peças. Por exemplo:

'[[[1,1,1],[0,0,1],[0,0,0]], [[1,0,1],[0,0,0],[1,0,1]]]' (Peças do jogador 1, estado com apenas duas peças restantes).

Em relação ao componente do tabuleiro, em vez de uma lista, ele será representado por uma matriz em que cada linha da matriz é uma lista de peças. Por exemplo:

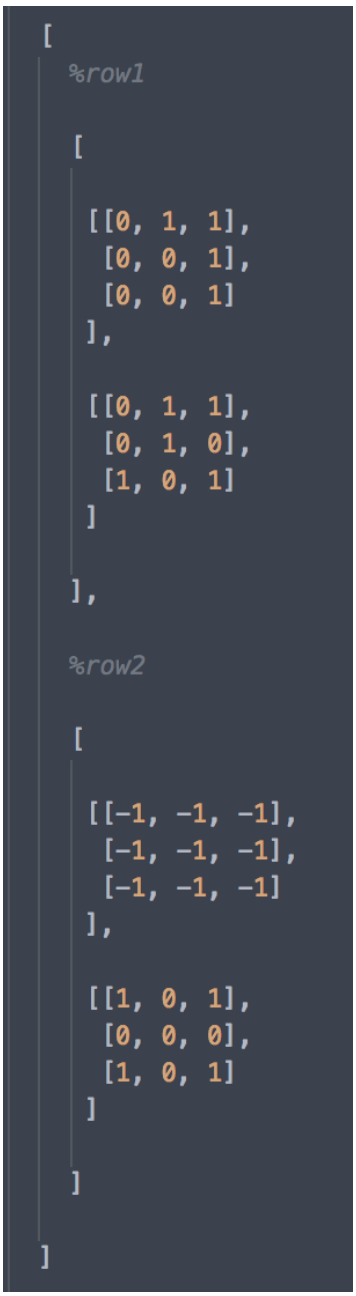


Figura 5 - Exemplo de Estado de Tabuleiro

Optou-se por manter sempre um tabuleiro regular, isto é, com todas as linhas do mesmo tamanho. Para isso, houve necessidade de criar uma representação dum estado vazio da seguinte forma: uma lista com a mesma estrutura da peça, mas com outros símbolos:

emptySpace([[-1, -1, -1],[-1, -1, -1],[-1, -1, -1]]). (predicado usado para construir um espaço vazio).

No exemplo da figura 5 temos um tabuleiro em que a posição (1,0) representa uma posição vazia.

Assim, de cada vez que um jogador joga uma peça nos limites do tabuleiro, este expande-se. As situações onde ocorre essa expansão são as seguintes:

- Jogador joga na linha 0 (é acrescentada uma linha vazia à matriz ficando esta na posição 0)
- Jogador joga na última linha (é acrescentada uma linha vazia ao final do tabuleiro)
- Jogador joga na primeira coluna (é acrescentada uma coluna vazia à esquerda do tabuleiro)
- Jogador joga na última coluna (é acrescentada uma coluna vazia à direita do tabuleiro)

3.2. VISUALIZAÇÃO DO TABULEIRO

Para o efeito de visualização, funcionalidade incluída no ficheiro 'print.pl' foram feitas várias abordagens. Por um lado, a visualização do tabuleiro em si, por outro, a visualização da lista de peças em cada jogada para que o jogador saiba quais são as suas peças ainda disponíveis. Como o tabuleiro é representado como uma lista de listas em que cada uma das listas é uma linha do tabuleiro que por sua vez é uma lista de peças, essa representação facilita a visualização. Foi desenvolvido um predicado recursivo que mostra cada linha de peças e outro também recursivo que percorre todas as linhas da matriz do tabuleiro e as desenha na consola. Caso haja espaços vazios, esse são representados por espaços vazios na matriz. Para o caso da lista de peças dos jogadores, estas são desenhadas numa lista horizontal e cada uma contém um índice para que o jogador a possa escolher para jogar.

Para cada peça, foi também desenvolvido um predicado para a desenhar, que imprime o elemento que representa cada elemento da matriz exceto elemento do centro, (linha 1, coluna 1) onde é imprimido um 'W' se for uma peça do jogador 1 ou um B se for uma peça do jogador 2.

A cada jogada é atualizado tabuleiro e as peças dos jogadores sendo estes elementos impressos de cada vez que há uma nova jogada.

Para facilitar a jogada, é também impresso no tabuleiro o número da linha e da coluna.

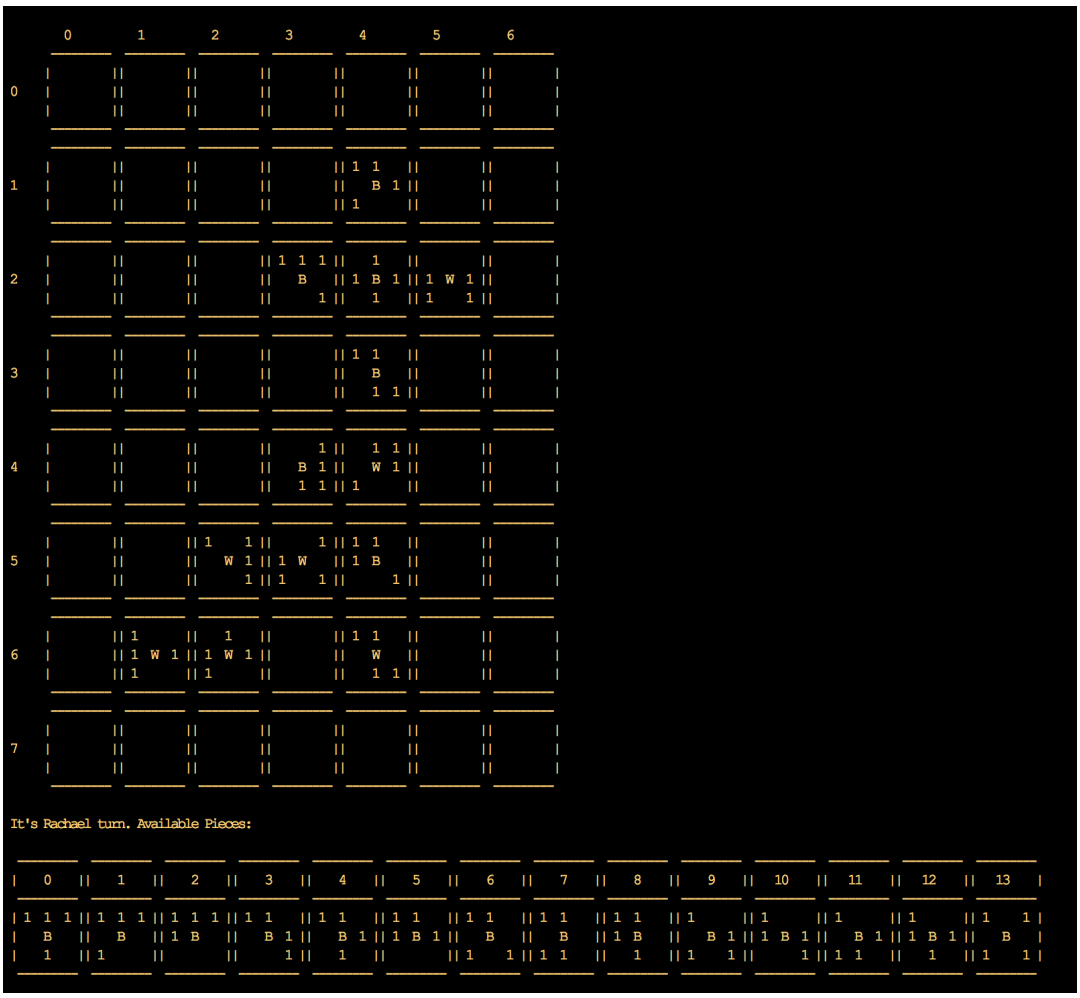


Figura 6 - Exemplo Impressões

3.3. VALIDAÇÃO DE JOGADAS

Em cada jogada são pedidos 5 parâmetros ao jogador:

- Peça a jogar
- Direção de Rotação
- Número de vezes a rodar
- Linha a jogar (Exceto na primeira jogada)
- Coluna a jogar (Exceto na primeira jogada)

Cada um destes parâmetros é avaliado antes de realizar a jogada. Em relação à peça a jogar é usado o predicado built-in nth0 que apenas sucede se o índice da peça escolhido for um índice valido da lista de peças ainda disponível, em relação à rotação, o jogador apenas poderá escolher a letra 'l' ou 'r' para rodar à esquerda ou à direita, caso falhe, o jogo volta a pedir ao jogador. Já no que diz

respeito ao número de rotações, o jogador apenas poderá escolher entre 0 e 2 voltando a escolher caso o valor não esteja compreendido entre 0 e 2. De seguida, é pedido ao jogador o índice da linha e da coluna onde quer jogar a peça. (De salientar que o programa valida inputs errados ie.: estar à espera de um carácter e ler um inteiro).

Caso todas estas validações de input sucedam, o programa tenta validar a jogada em si, aplicando a regra de adjacência:

validPlay(+Board,+ Row,+Column).

Este predicado sucede se para o tabuleiro instanciado, a linha e coluna instanciadas passadas nos parâmetros correspondem a uma jogada válida, ou seja, se o espaço no tabuleiro está vazio e se tem pelo menos uma peça à direita ou esquerda, em cima ou em baixo.

Para o modo de jogo humano vs computador, é possível também obter listas de possíveis jogadas tendo em conta vários parâmetros, como é explicado no ponto 3.7.

3.4. EXECUÇÃO DE JOGADAS

Como explicitado no ponto anterior, o programa pede 5 parâmetros ao jogador, que depois usa para preparar a jogada e para realizar a jogada. Mais concretamente, o parâmetro peça, rotação e vezes de rotação serve para preparar a peça, isto é feito com o seguinte predicado:

rotatePiece(+Piece,-PieceRotated,+Direction,+Times), este predicado recebe a peça original e roda-a as vezes e na direção especificadas, sendo depois substituída na lista de peças do jogador. Depois disso o predicado responsável para a realização da jogada é:

playPiece(+Board, -NewBoard, +Row, +Column, +PieceNumber, +Pieces, -NewPieces), este predicado devolve um novo tabuleiro e lista de peças caso a linha e coluna representem uma posição válida naquele tabuleiro, caso contrário devolve o mesmo tabuleiro e as mesmas peças:

playPiece(+Board,-Board, +Row, +Column, _, +Pieces, -Pieces).

Existe uma variação sem a passagem da linha e coluna nos parâmetros para a primeira jogada, já que na primeira jogada o tabuleiro está vazio e a peça é sempre jogada na mesma posição:

playFirstPiece(+InitialBoard, -CurrentBoard, +Pieces, +PieceNumber, -NewPieces).

De notar que se for a vez do computador a jogar, estes parâmetros não são pedidos ao utilizador mas sim gerados com base na lista de posições possíveis a jogar, como é explicado na secção 3.7.

Este predicado é chamado por outro que existente na interface:

play(+PiecesPlayer1, +PiecesPlayer2, +Board, -NewPiecesPlayer1, -PiecesPlayer2, -NextBoard, player1, player2,Player1Name,_)

play(+PiecesPlayer1, +PiecesPlayer2, +Board, -PiecesPlayer1, -NewPiecesPlayer2, -NextBoard, player2, player1,_,Player2Name)

Este predicado é chamado num ciclo de jogo que vai alternando entre o jogador 1 e o jogador 2:

```
playCycle(PiecesPlayer1, PiecesPlayer2, CurrentBoard, CurrentPlayer,Player1Name,Player2Name) :-


  play(PiecesPlayer1, PiecesPlayer2, CurrentBoard, NewPiecesPlayer1, NewPiecesPlayer2, NextBoard, CurrentPlayer, NextPlayer,Player1Name,Player2Name),
  calculateGlobalScore(NextBoard,GlobalScore1,player1),
  calculateGlobalScore(NextBoard,GlobalScore2,player2),
  nl,write('Current Score:'),nl,
  write(Player1Name), write(': '), write(GlobalScore1),nl,
  write(Player2Name), write(': '), write(GlobalScore2),nl,nl,
  playCycle(NewPiecesPlayer1,NewPiecesPlayer2, NextBoard, NextPlayer,Player1Name,Player2Name).
```

Figura 7 - Ciclo de Jogo

3.5. AVALIAÇÃO DO TABULEIRO

A cada jogada, o tabuleiro é avaliado para a obtenção da pontuação atual de cada jogador. Cada padrão que o jogador consiga construir vale 1 ponto. Essa avaliação é realizada através dum predicado que chama, para todas as peças do tabuleiro dum jogador, outro predicado que avalia se aquela peça tem ou não o padrão construído à sua volta:

playerGoodPiece(+Board, +Row, +Column,+Player). Sucede se o padrão está recriado.

```
playerScoreInPiece(Board, Row, Column, 1,Player) :-
  playerGoodPiece(Board,Row,Column,Player).
playerScoreInPiece(Board, Row, Column, 0,Player) :-
  \+ playerGoodPiece(Board,Row,Column,Player).
```

Figura 8 – Calcular Resultado da Peça

Fazendo uso destes predicados existe um outro mais global que calcula o score para todas as peças de um determinado jogador:

calculateGlobalScore(+Board,-GlobalScore,+Player).

A cada jogada, o resultado atual é mostrado no ecrã:

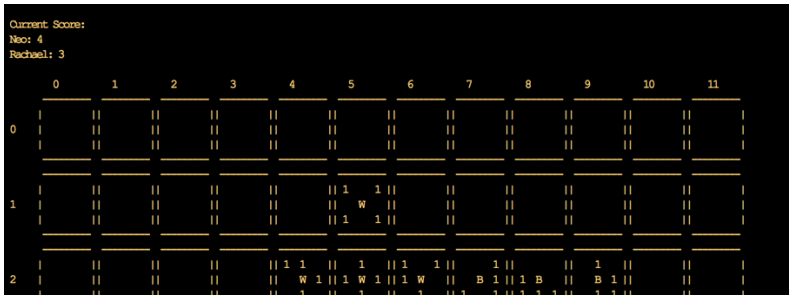


Figura 9 - Resultado Atual

3.6. FINAL DO JOGO

Como foi explicado, nesta versão do jogo, o final do jogo é determinado pelo número de peças restantes, ou seja, termina quando os dois jogadores esgotam as peças. Na prática, quando o jogador 2 esgota as peças já que as jogadas são sempre alternadas e o jogador 1 começa sempre.

Como esta é a única condição para o final do jogo, o predicado de ciclo de jogo tem esta condição de terminação:

```
playCycle([],[],FinalBoard,_,Player1Name,Player2Name) :-

  printFullBoard(FinalBoard),

  calculateGlobalScore(FinalBoard,GlobalScorePlayer1,player1),
  calculateGlobalScore(FinalBoard,GlobalScorePlayer2,player2),
  chooseFinalMessage(GlobalScorePlayer1,GlobalScorePlayer2,Player1Name,Player2Name).
```

O predicado chooseFinalMessage é responsável por imprimir o vencedor no ecrã, que será o jogador com a pontuação mais elevada:

3.7. JOGADA DO COMPUTADOR

Para as jogadas do computador, foram desenvolvidos três níveis de dificuldade, além de um modo de jogo 'Computador vs Computador'. É de salientar que o ciclo de jogo para os modos de 'Humano vs Computador' e 'Computador vs Computador' são em tudo semelhantes ao ciclo de jogo de 'Humano vs Humano' nomeadamente as chamadas aos predicados play. No caso de humano contra humano, o humano é o jogador 1 e o computador o jogador 2, no caso de Computador contra Computador, existe um jogador 1 e um jogador 2 , são ambos máquinas.

- Nível fácil

Para o nível fácil, a jogada do computador é completamente aleatória na seleção da posição. Existe um predicado que devolve uma lista de todas as possíveis posições válidas no tabuleiro. O computador escolhe uma aleatoriamente:

getAllValidPositions(+Board,-ValidPositionsList).

- Nível médio

No nível médio de dificuldade, o computador alterna entre ataque e defesa (30% defesa, 70% ataque). No caso de não haver posições para defender opta sempre pelo ataque.

  - Ataque

    Obtém a lista de posições possíveis, mas escolhe a melhor delas. De salientar que esta lista é obtida através do predicado:

    getGoodPositionsBoard(+Board,+Player,-GoodPositionsList).

    A lista obtida está ordenada pela prioridade da posição sendo a prioridade calculada da seguinte forma:

      - Peças que tenham à sua volta, nas posições do seu padrão, peças do jogador adversário têm prioridade 4

      - Peças que não tenha à sua volta nenhuma peça do jogador adversário têm prioridade 'Número de Espaços vazias do seu padrão ". Desta forma, por exemplo, uma peça que tenha apenas um espaço vazio a faltar para completar o padrão tem prioridade 1.

    Assim, o computador jogará na primeira posição da lista que corresponderá a uma posição com a prioridade mais elevada.

  - Defesa

    Quando existir possibilidade de defesa, o computador vai defender em 70% das vezes. A defesa é efetuada da seguinte forma: é obtida uma lista de posições em perigo. Uma posição em perigo corresponde a uma posição livre que, ao ser deixada livre, o adversário completa um padrão ao jogar nessa posição. O seguinte predicado verifica se uma dada posição de uma peça tem uma posição em perigo:

    checkDangerPiece(+Board,+Piece,+PieceRow,+PieceColumn,+OpponentPlayer,-Row,-Column).

    Se existir uma posição em perigo para a peça, essa posição é devolvida em Row e Column. Se não existir, Row e Column tomam o valor de -1. (checkDangerPiece(_,_,_,_,_,-1,-1).).

    Para obter a lista de todas as posições em perigo, é chamado um predicado que verifica todas as peças do jogador no tabuleiro e devolve a lista de todas as posições em perigo:

    checkDefense(+Board ,+Opponent,-DangerList).

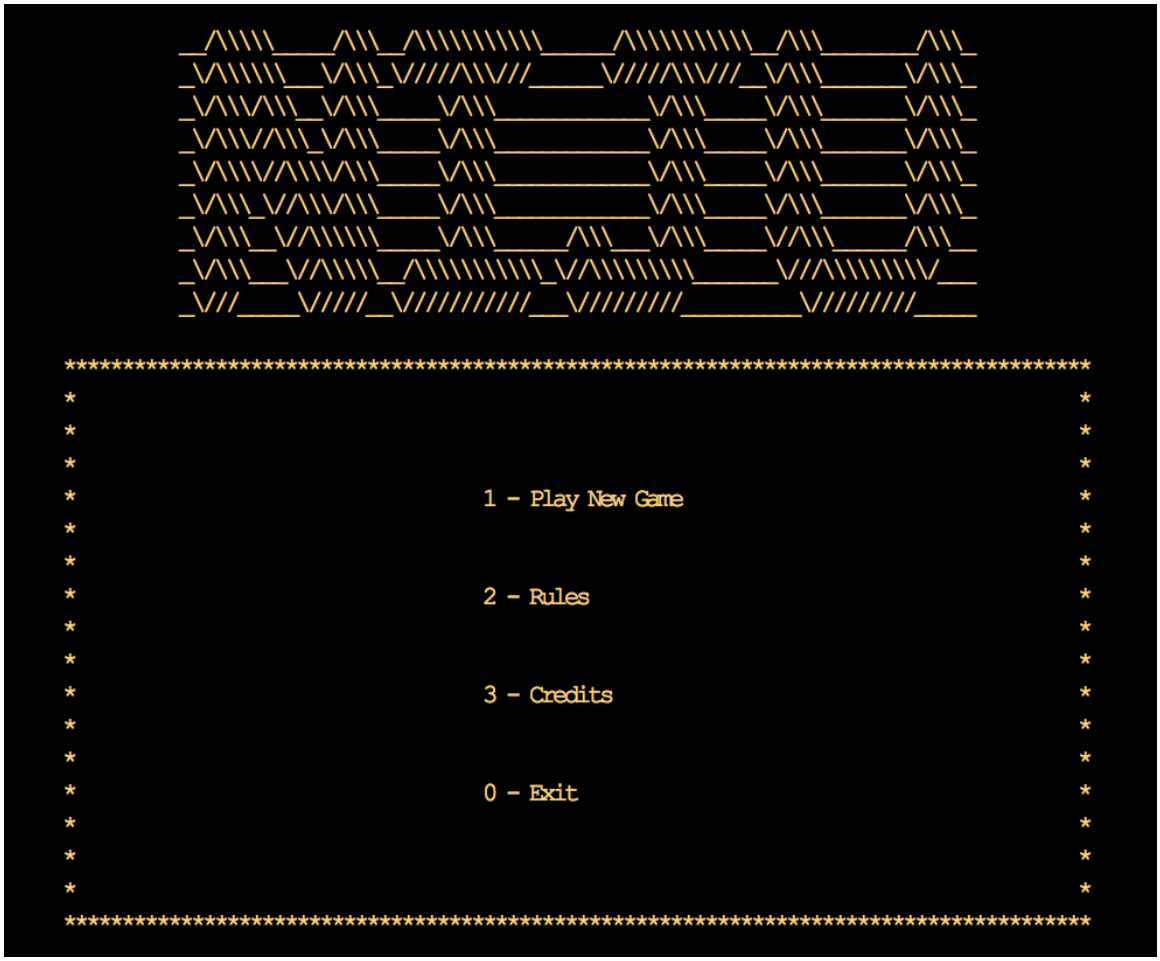
Através desta lista, e se a mesma não for vazia, quando o computador joga à defesa, jogará na primeira posição desta lista.

- Nível médio

No nível difícil, no caso de haver uma lista de posições a defender, a defesa é sempre privilegiada em relação ao ataque. Desta forma, para poder fazer um ponto, o jogador tem que conseguir ter, na mesma jogada mais do que um padrão em que necessita apenas de mais uma peça para o recriar.

4. INTERFACE COM UTILIZADOR

A interface com o utilizador é muito simples e é baseada em menus e opções que o utilizador seleciona:



Todos os menus de jogo são visualizados desta forma e a navegação é feita selecionando a opção desejada.

5. CONCLUSÕES

Após a realização deste trabalho prático conclui-se que uma abordagem em programação em lógica para a implementação deste género de jogos é vantajosa em relação a uma linguagem funcional ou orientada a objetos, pela flexibilidade, pelo modo generativo de soluções que a linguagem proporciona e pelo tamanho reduzido do código desenvolvido. Por outro lado, no que diz respeito à interface com utilizador fica aquém de outro tipo de linguagens, tendo sido a parte do trabalho mais afetada pela abordagem feita.

BIBLIOGRAFIA

- Cardoso, Henrique Lopes; Programação em Lógica: Fundações, DEI,FEUP,2013
- Cardoso, Henrique Lopes; Programação em Lógica: Conceitos, DEI,FEUP,2013
- SWI Prolog Reference Manual. Disponível em http://www.swi-prolog.org/pldoc/doc_for?object=manual
- StackOverflow. https://stackoverflow.com/