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

Meerkats

Relatório – Programação Lógica

Luís Miguel Gonçalves – 201207141, Mário André Macedo Ferreira - 201208066

11/8/2015

Índice

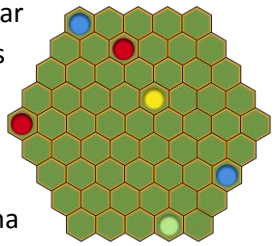
Introdução

Pretende-se neste trabalho implementar, em linguagem Prolog, um jogo de tabuleiro para 1 ou mais jogadores. O jogo deverá ter três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador bem como uma interface gráfica adequada ao utilizador. Com este trabalho é pretendido que os estudantes utilizem os seus conhecimentos da linguagem Prolog e adequiram novos consoante as necessidades.

O relatório está estruturado consoante o template fornecido pelos docentes no Moodle seguindo cada conteúdo como especificado.

O Jogo “Meerkats”

O jogo Meerkats, criado em 2014 por Rey Alicea, é um jogo de tabuleiro baseado em estratégia e *bluff*. Pode ser jogado por um mínimo de 2 pessoas e um máximo de 4. O jogo consiste num tabuleiro hexagonal (5 hexágonos em cada um dos 6 lados), 64 peças, 16 de cada cor (Azul, Verde, Amarelo e Vermelho).



A vitória é dada ao jogador que no final (quando não se puderem executar mais movimentos) tiver o grupo de peças, da sua cor, maior entre os maiores de cada cor, sendo que cada grupo é representado pelo numero de peças da mesma cor com uma aresta adjacente (1 peça é um grupo). Cada jogador sabe a sua cor quando, antes do iniciar o jogo, é colocado num saco opaco uma peça de cada cor e cada jogador tira uma peça aleatória sem a revelar até ao final do jogo.

Na primeira jogada um jogador coloca uma peça qualquer num espaço qualquer do tabuleiro, no entanto, nos turnos seguintes, os jogadores têm de colocar uma peça de uma cor qualquer no tabuleiro (desde que tenha pelo menos um espaço adjacente livre) e, de seguida, mexer uma outra peça qualquer em linha recta na diagonal, vertical ou horizontal o número desejado de casas, tendo em conta que não é possível saltar por cima de outras peças nem passar as fronteiras do tabuleiro. Quando um jogador não puder executar um destes movimentos o jogo termina.

Logica do Jogo

Representação do Estado do Jogo e Visualização

Como referido acima, o tabuleiro é hexagonal sendo que cada lado tem uma largura de 5 células (cada célula sendo um hexágono), assim sendo, a abordagem que considerámos mais apropriada para este casocaso foi criar uma lista de listas em que cada lista consiste numa linha. Assim sendo existem 9 listas dentro da lista que contem o tabuleiro com diferentes dimensões e um total de 61 células, ou seja 61 espaços inicialmente vazios para as peças.

```
emptyBoard([[0,0,0,0,0],
            [0,0,0,0,0,0],
            [0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0,0],
            [0,0,0,0,0,0,0],
            [0,0,0,0,0,0],
            [0,0,0,0,0]]).
```

Tendo em conta que as peças podem ser até 4 tipos diferentes de peças (consoante o número de jogadores) estas são diferenciadas por números, sendo que cada número corresponde a uma peça diferente. Será necessário percorrer cada lista e imprimir os valores presentes na mesma por forma a ser o mais natural possível para os jogadores avaliarem o tabuleiro.

Figura 2: Posição inicial, intermédia e final do jogo - Ascii

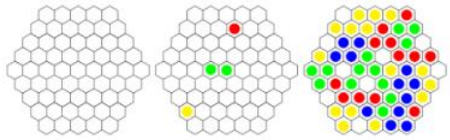


Figura 3: Posição inicial, intermédia e final do jogo - Ilustração

Por forma a imprimir a lista foram declarados predicados em Prolog e chegámos a uma representação que achamos adequada e perceptivel pelo jogador que pode ser verificada na figura 2 acima.

Execução de Jogadas e Verificação

Como referido acima, cada jogada consiste de duas partes: primeiramente o jogado coloca uma peça e de seguida arrasta uma peça já no tabuleiro para outra célula (isto respeitando as regras do jogo). Tendo isto em conta o grupo criou um conjunto de predicados que achou apropriados para ambas as partes da jogada.

Primeira Parte:

Para inserir uma peça no tabuleiro foi criado o predicado *tryToAddPieceToBoard*. Responsavel pela verificação da jogada pretendida pelo jogador sendo que se esta for valida coloca a peça no sitio desejado.

tryToAddPieceToBoard recebe o estado do tabuleiro, a cor da peça a ser inserida bem como a linha e a coluna a inserir. Inicialmente são feitas verificações para ver se o destino se encontra numa esquina do hexagono (tabuleiro), de seguida nas margens e por fim se é no meio. E só depois são feitas as verificações tendo em conta a posição relativa (canto, berma ou meio).

```
tryToAddPieceToBoard(BoardState, Color, Row, Column):-
    emptyCell(BoardState, Row, Column, C),
    Row = 4, !,
    Column = 8, !,
    checkCorner(BoardState, 3),
    addPieceToBoard(BoardState, Color, Row, Column, 0, [], FinalBoard).
tryToAddPieceToBoard(BoardState, Color, Row, Column):-
    emptyCell(BoardState, Row, Column, C),
    Row = 4, !,
    Column = 0, !,
    checkCorner(BoardState, 6),
    addPieceToBoard(BoardState, Color, Row, Column, 0, [], FinalBoard).
tryToAddPieceToBoard(BoardState, Color, Row, Column):-
    emptyCell(BoardState, Row, Column, C),
    Row = 4, !,
    checkMiddleCell(BoardState ,Row, Column),
    addPieceToBoard(BoardState, Color, Row, Column, 0, [], FinalBoard).
tryToAddPieceToBoard(BoardState, Color, Row, Column):-
    emptyCell(BoardState, Row, Column, C),
    Row < 4, !,
    checkAdjacentCells(BoardState ,Row, Column),
    addPieceToBoard(BoardState, Color, Row, Column, 0, [], FinalBoard).
tryToAddPieceToBoard(BoardState,Color, Row, Column):-
    emptyCell(BoardState, Row, Column, C),
    Row > 4, !,
    RevRow is 8 - Row,
    reverse(BoardState, ReversedBoard),
    checkAdjacentCells(ReversedBoard ,RevRow, Column),
    addPieceToBoard(BoardState, Color, Row, Column, 0, [], FinalBoard).
```

Sendo que depois de feitas as verificações necessárias é executado o *addPieceToBoard* em que a peça é colocada na célula especificada.

Segunda Parte:

Para mover a peça foi criado o predicado *move*. Responsável por, primeiro, ir buscar a peça (cor) que o jogador pretende mexer consoante as coordenadas inseridas pelo mesmo e, de seguida, tentar mover a peça casa a casa (consoante o numero de casas indicado pelo jogador) verificando a validade da jogada a cada iteração.

```
move(BoardState, RowSource, ColumnSource, Moves, Orientation, OK):-
    getPiece(BoardState, RowSource, ColumnSource, Color),
    tryToMovePiece(BoardState, Color, RowSource, ColumnSource, Moves, Orientation, Board).
```

getPiece é responsável por ir buscar a peça (como referido acima). Depois de obtida a cor da peça *tryToMovePiece* recebe a orientação e os moves (número de casa a andar) bem como a posição da peça sendo que esta função é chamada recursivamente até não haverem mais casas a andar.

Determinação do final de jogo

O final de jogo não é determinado.

Interface com o utilizador

No início do jogo é apresentado ao(s) jogador(s) o título do jogo e um menu onde pode(m) escolher o modo de jogo, isto é, Single Player ou Multiplayer (2, 3 ou 4 jogadores), como se pode ver na imagem abaixo.

Depois de escolher o modo de jogo o(s) jogador(s) escolhe(m) o(s) seu(s) nome(s) e aleatoriamente é-lhe(s) atribuída uma cor.

Durante o jogo, na primeira parte da jogada é pedido ao jogador uma Cor, o número da Linha e da Coluna onde deseja colocar a peça.

Na segunda parte, é pedida uma direção (de 1 a 6) e o número de casas para avançar.

Se em algum momento a jogada for inválida é pedido nova introdução de dados.

Conclusões

Apesar de todas as funções de funcionamento de jogo estarem completas o grupo não conseguiu concluir o trabalho. Infelizmente e devido especialmente à existência de demasiadas entregas de diferentes cadeiras ao mesmo tempo e o facto de não termos começado ou andiantado suficientemente o trabalho no inicio.