OTRIO

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Logica

Grupo Otrio_2:
Paulo Sérgio da Silva Babo – up201404022
Nuno Filipe Sousa e Silva – up201404380

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

11 de Novembro de 2016

Resumo

Foi nos proposto para este trabalho prático criar um jogo utilizando a linguagem de programação prolog , no nosso caso escolhemos o jogo Otrio, que é uma espécie de jogo do galo em que cada célula e peças de jogo têm 3 tamanhos diferentes, dando asas a novas possibilidades de jogo e tornando o jogo mais competitivo. Abordamos então este problema proposto com o intuito de desenvolver o jogo da maneira mais facil e simples possível, tentando assim proporcionar uma interface compreensível para o utilizador enquanto mantinhamos todas as mecânicas do jogo original intactas. Decidimos então para atingir este objetivo utilizar um método baseado em listas representativas dos vários elementos do jogo (sets e tabuleiro), e com um conjunto de predicados aplicar todas as movimentações possíveis no jogo real, estas feitas tanto pelo jogador em si como feitas pela máquina. Conseguimos assim atingir esse objetivo de maneira satisfatória embora tenham surgido alguns inconvenientes, posteriormente ultrapassados, no decorrer do desenvolvimento.

Índice

Resumo 2
1 - Introdução 4
2 - O jogo: Otrio 4
3 - Lógica do Jogo 6
3.1 - Representação do Estado de Jogo 6
3.2 - Visualização do Tabuleiro 10
3.3 - Lista de Jogadas Válidas 11
3.4 - Execução de Jogadas 12
3.5 - Avaliação do Tabuleiro 12
3.6 - Final do Jogo 12
3.7 – Jogada do Computador 13
4 - Interface com o Utilizador 13
5 - Conclusões 14
6 - Bibliografia 15

1 - Introdução

Na disciplina de PLOG da faculdade de engenharia da universidade do Porto (FEUP) foi nos proposto como trabalho prático 1 desenvolver um jogo de lógica (otrio) utilizando a linguagem prolog como ferramenta principal para esse desenvolvimento. Neste relatório pretendemos então dar a conhecer o funcionamento do nosso jogo, regras, mecânicas, lógica e interface , dando assim a conhecer todos os aspetos relevantes do nosso projeto.

2 - O jogo: Otrio

Otrio é um jogo de tabuleiro criado pela empresa Marbles: the brain store que se inspirado no tao conhecido ” jogo do galo”, e lhe tenta dar um novo nÍvel de complexidade adicionando 3 camadas de peças de diferentes tamanhos em cada célula.
As regras são bastante simplistas o que faz com que o jogo seja bastante fácil de jogar, mas o número de estratégias e combinações é enorme o que o torna dificil de dominar.

Regras:

-Número de Jogadores: 2-4.

-Modo de jogo: Cada jogador na sua vez deve colocar uma das suas peças ainda disponíveis, numa célula do tabuleiro. Cada célula pode conter até 3 peças de tamanhos diferentes.

-Objetivo: Cada jogador deverá tentar colocar as peças numa das seguintes posições vitoriosas:

-Em linha (mesmo tamanho) : O jogador deve colocar 3 peças do mesmo tamanho em linha tabuleiro.

-Em linha (crescente ou decrescente): O jogador deve colocar 3 peças por ordem crescente ou decrescente em linha no tabuleiro.

-Concêntrica: O jogador deve colocar as 3 peças de tamanho diferente numa célula do tabuleiro.

3 - Lógica do Jogo

3.1 - Representação do Estado de Jogo

O tabuleiro do nosso jogo é representado por uma lista com 3 listas, em que cada uma dessas listas representa uma linha, que por sua vez contêm mais 3 listas que representam o tamanho das peças em cada linha (1ª lista- grande, 2ª lista- médio, 3ª lista- pequeno) que sao constituídas pelas peças na sua respectiva posição.

```
board([
        [[b3,r3,g3],
        [b2,r2,g2],
        [b1,r1,g1]],

        [[p3,b3,g3],
        [p2,b2,g2],
        [p1,b1,g1]],

        [[r3,n3,n3],
        [n2,r2,n2],
        [n1,n1,r1]]
]).
```

Linha do tabuleiro

Lista de peças grandes (na linha)

Lista de peças médias (na linha)

Lista de peças pequenas (na linha)

Peça

Legenda das peças:

bX – peça azul

rX – peça vermelha

gX – peça verde

pX – peça roxa

nX – espaço vazio

X=1 – peça pequena

X=2 – peça média

X=3 – peça grande

Posições de jogo (exemplos):

1- <u>Posição inicial</u>: Tabuleiro vazio e todas as peças no set inicial do respectivo jogador.

```
%Tabuleiro
board([
    [[n3,n3,n3],
    [n2,n2,n2],
    [n1,n1,n1]],

    [[n3,n3,n3],
    [n2,n2,n2],
    [n1,n1,n1]],

    [[n3,n3,n3],
    [n2,n2,n2],
    [n1,n1,n1]]
]).
```

```
%Conjunto das peças do jogador 1
p1Set([
    [[b3,b3,b3],
    [b2,b2,b2],
    [b1,b1,b1]]
    ]).

%Conjunto das peças do jogador 2
p2Set([
    [[r3,r3,r3],
    [r2,r2,r2],
    [r1,r1,r1]]
    ]).

%Conjunto das peças do jogador 3
p3Set([
    [[p3,p3,p3],
    [p2,p2,p2],
    [p1,p1,p1]]
    ]).

%Conjunto das peças do jogador 4
p4Set([
    [[g3,g3,g3],
    [g2,g2,g2],
    [g1,g1,g1]]
    ]).
```

2- Posição intermedia: Tabuleiro já com peças e as respectivas peças "retiradas" do set inicial do jogador.

```
%Tabuleiro
board([
    [[b3,g3,n3],
    [n2,n2,p2],
    [n1,n1,n1]],

    [[g3,n3,n3],
    [n2,p2,r2],
    [p1,n1,n1]],

    [[r3,n3,b3],
    [g2,n2,r2],
    [n1,b1,n1]]
]).
```

```
%Conjunto das peças do jogador 1
p1Set([
    [[n3,n3,b3],
    [b2,b2,b2],
    [n1,b1,b1]]
    ]).

%Conjunto das peças do jogador 2
p2Set([
    [[n3,r3,r3],
    [n2,n2,r2],
    [r1,r1,r1]]
    ]).

%Conjunto das peças do jogador 3
p3Set([
    [[p3,p3,p3],
    [n2,n2,p2],
    [n1,p1,p1]]
    ]).

%Conjunto das peças do jogador 4
p4Set([
    [[n3,n3,g3],
    [n2,g2,g2],
    [g1,g1,g1]]
    ]).
```

3- Posições finais:

3.1-Vitória em linha (mesmo tamanho).

```
%Tabuleiro
board([
    [[b3,g3,n3],
    [n2,n2,p2],
    [n1,n1,n1]],

    [[g3,b3,n3],
    [n2,p2,r2],
    [p1,n1,n1]],

    [[r3,n3,b3],
    [g2,n2,r2],
    [n1,b1,n1]]
]).
```

```
%Conjunto das peças do jogador 1
p1Set([
    [[n3,n3,n3],
    [b2,b2,b2],
    [n1,b1,b1]]
    ]).

%Conjunto das peças do jogador 2
p2Set([
    [[n3,r3,r3],
    [n2,n2,r2],
    [r1,r1,r1]]
    ]).

%Conjunto das peças do jogador 3
p3Set([
    [[p3,p3,p3],
    [n2,n2,p2],
    [n1,p1,p1]]
    ]).

%Conjunto das peças do jogador 4
p4Set([
    [[n3,n3,g3],
    [n2,g2,g2],
    [g1,g1,g1]]
    ]).
```

3.2- Vitória em linha (crescente ou decrescente).

```
%Tabuleiro
board([
    [[b3,p3,g3],
    [n2,n2,n2],
    [n1,n1,n1]],

    [[g3,b3,n3],
    [n2,p2,r2],
    [n1,n1,n1]],

    [[r3,n3,n3],
    [g2,n2,r2],
    [b1,p1,n1]]
]).
```

```
%Conjunto das peças do jogador 1
p1Set([
    [[n3,n3,b3],
    [b2,b2,b2],
    [n1,b1,b1]]
    ]).

%Conjunto das peças do jogador 2
p2Set([
    [[n3,r3,r3],
    [n2,n2,r2],
    [r1,r1,r1]]
    ]).

%Conjunto das peças do jogador 3
p3Set([
    [[n3,p3,p3],
    [n2,p2,p2],
    [n1,p1,p1]]
    ]).

%Conjunto das peças do jogador 4
p4Set([
    [[n3,n3,g3],
    [n2,g2,g2],
    [g1,g1,g1]]
    ]).
```

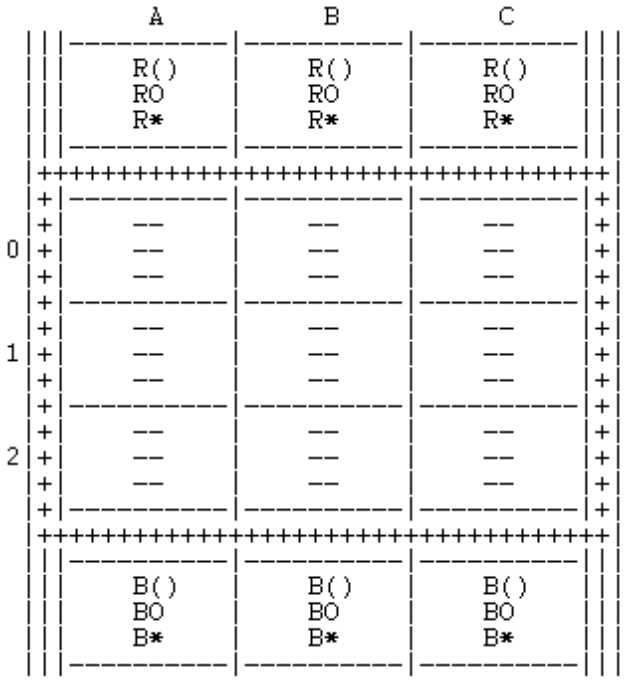
3.3- Vitória concentrica.

```
%Tabuleiro
board([
    [[b3,p3,g3],
    [n2,p2,n2],
    [n1,p1,n1]],

    [[g3,b3,n3],
    [n2,n2,r2],
    [n1,n1,n1]],

    [[r3,n3,n3],
    [g2,n2,r2],
    [b1,n1,n1]]
]).
```

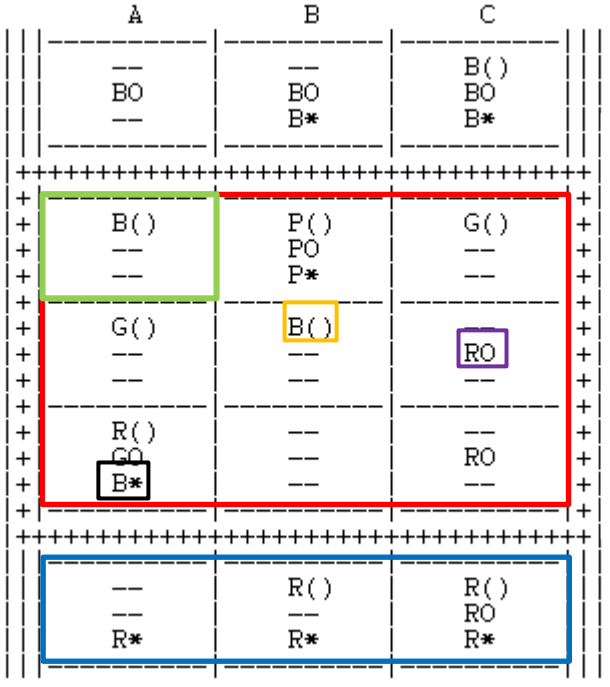
```
%Conjunto das peças do jogador 1
p1Set([
    [[n3,n3,b3],
    [b2,b2,b2],
    [n1,b1,b1]]
    ]).

%Conjunto das peças do jogador 2
p2Set([
    [[n3,r3,r3],
    [n2,n2,r2],
    [r1,r1,r1]]
    ]).

%Conjunto das peças do jogador 3
p3Set([
    [[n3,p3,p3],
    [n2,p2,p2],
    [n1,p1,p1]]
    ]).

%Conjunto das peças do jogador 4
p4Set([
    [[n3,n3,g3],
    [n2,g2,g2],
    [g1,g1,g1]]
    ]).
```

3.2 - Visualização do Tabuleiro

A visualização do tabuleiro é uma aproximação ao tabuleiro real em que temos no centro a área de jogo (3x3) e á volta desta área a representação das peças que cada jogador possui no momento. As peças são representadas com a letra da respectiva cor e uma simbologia a indicar o seu tamanho.



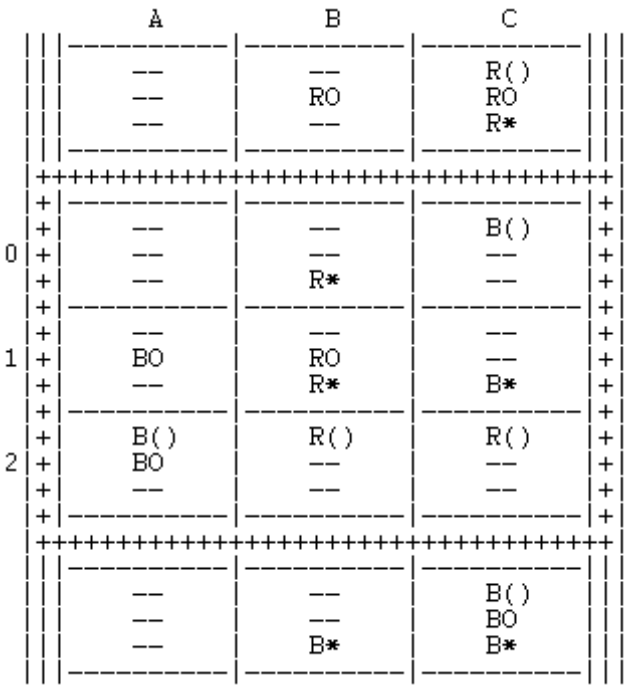
Img - Visualização de tabuleiro vazio.



- Area de jogo.
- Set de peças de jogador
- Célula do tabuleiro
- Peça grande
- Peça media
- Peça pequena

3.3 - Lista de Jogadas Válidas

Podemos obter uma lista de jogadas válidas (desocupadas por peças) utilizando o predicado jogadapossivelTAMANHO(Lista) em que TAMANHO toma o nome do tamanho da peça (em minúsculas) para que é desejado ver as jogadas disponíveis, devolvendo cada um destes predicados uma Lista com uma lista das coordenadas de posições disponíveis para jogar, como podemos observar no seguinte exemplo:




```
| ?- jogadapossivelmedia(Lista).
Lista = [[0,0],[1,0],[2,0],[2,1],[1,2],[2,2]] ?


| ?- jogadapossivelgrande(Lista).
Lista = [[0,0],[1,0],[0,1],[1,1],[2,1]] ?


| ?- jogadapossivelpequena(Lista).
Lista = [[0,0],[2,0],[0,1],[0,2],[1,2],[2,2]] ?
```

3.4 - Execução de Jogadas

As jogadas são essencialmente executadas pelo predicado jogadaX(X,Y,Peca,Set,NewSet), que verifica se a peça existe no set, se as coordenadas estão livres e se ambas estas condições se verificarem, retira a peça do set e coloca-a no tabuleiro, devolve NewSet, que é o Set modificado. Atualiza logo o tabuleiro utilizando o predicado asserta(L).

3.5 - Avaliação do Tabuleiro

Na avaliação do tabuleiro e na aplicação de diferentes jogadas decidimos aplicar apenas um método de avaliação do tabuleiro devido á quantidade de jogadas possíveis e estratégias tanto á defesa como ao ataque. Criamos então um predicado tentaMelhorJogadaCOR, em que COR toma o valor R ou B (consoante a cor do jogador, vermelho ou azul), que analisa o tabuleiro e se for possível acabar o jogo na próxima jogada ele executa essa jogada, tomando assim decisões mais eficientes no decorrer do jogo.

3.6 - Final do Jogo

O final do jogo é verificado pelo predicado verSeGanhou. que compara o estado do tabuleiro atual com as diferentes formas de ganhar o jogo, e se suceder escreve uma mensagem de vitória juntamente com o jogador vencedor e falha, fazendo assim com que esse jogo acabe.

```
###   Jogador 1 ganhou !   ###

    ********    VITORIA!    ********
[[r3,n3,n3],[r2,n2,n2],[r1,n1,n1]]
                      A          B          C
                  |||----------|----------|----------|||
                  |||    --    |    --    |    R()   |||
                  |||    --    |    --    |    RO    |||
                  |||    --    |    --    |    R*    |||
                  |||----------|----------|----------|||
                  |++++++++++++++++++++++++++++++++++++|
                  |+|----------|----------|----------|+|
                  |+|    --    |    --    |    --    |+|
                0 |+|    --    |    BO    |    RO    |+|
                  |+|    --    |    --    |    --    |+|
                  |+|----------|----------|----------|+|
                  |+|    R()   |    --    |    --    |+|
                1 |+|    RO    |    --    |    --    |+|
                  |+|    R*    |    --    |    --    |+|
                  |+|----------|----------|----------|+|
                  |+|    B()   |    R()   |    --    |+|
                2 |+|    --    |    --    |    BO    |+|
                  |+|    R*    |    B*    |    B*    |+|
                  |+|----------|----------|----------|+|
                  |++++++++++++++++++++++++++++++++++++|
                  |||----------|----------|----------|||
                  |||    --    |    B()   |    B()   |||
                  |||    --    |    --    |    BO    |||
                  |||    --    |    --    |    B*    |||
                  |||----------|----------|----------|||
____GAME OVER____
```

3.7 – Jogada do Computador

O nível de dificuldade 1 da jogada de computador utiliza o predicado escolherPeca(Set,Peca) que escolhe uma Peça aleatória existente em Set, e com o predicado jogadacomputadorX(Peca,Set,NewSet), escolhe uma posição aleatória que esteja disponível (utilizando o predicado anteriormente descrito jogadapossivelTAMANHO(Lista), sendo TAMANHO o tamanho da peça a colocar, e escolhendo um par de coordenadas aleatório de Lista), e posteriormente colocando a peça com jogadaX(X,Y,Peca,Set,NewSet) com as coordenadas obtidas anteriormente.

O nível de dificuldade 2 da jogada de computador utiliza a estratégia de avaliar o tabuleiro (predicado anteriormente descrito tentaMelhorJogadaR/tentaMelhorJogadaB) e ve se na próxima jogada consegue acabar o jogo, se sim fá-lo senao faz uma jogada aleatória como faria no nivel de dificuldade um. Sendo assim mais eficiente que uma jogada de nível um.

4 - Interface com o Utilizador

Para entrar no jogo basta utilizar o predicado playGame. e é desde logo lançado um menu em que é possível escolher o modo de jogo desejado. Utilizando os números descritos o utilizador pode então começar um jogo no modo desejado.

<u>Modos:</u>

PvP- Neste modo o jogador pode jogar contra outro jogador no mesmo computador, as jogadas são feitas à vez, e cada jogador tem então de introduzir a peça a jogar e as coordenadas da posição que a peça será jogada, ganha aquele que chegar a um estado final primeiro.

PvC- Neste modo o jogador vai enfrentar o computador no seu nível de dificuldade escolhido, em que após uma jogada do jogador , o computador faz a sua jogada automaticamente, ganha aquele chegar a um estado final primeiro.

CvC-Neste modo o jogo vai ser jogado apenas pelo computador no seu nível de dificuldade escolhido, jogando cada um á vez, até um ganhar ou ambos acabarem sem peças para jogar.

5 - Conclusões

Podemos então concluir que embora tenhamos pouca experiência na linguagem prolog, é uma linguagem muito poderosa para desenvolver jogos deste tipo, porém a nossa inexperiencia foi algo que se fez demonstrar em várias fases do projeto o que condicionou de certa forma o decorrer do projeto. Podemos então dizer que conseguimos superar algumas dessas dificuldades e fomo-nos familiarizando com a linguagem , produzindo então um produto final acabado.

6 - Bibliografia

Marbles the brain store website: http://www.marblesthebrainstore.com/otrio.htm;
VAT19 website: https://www.vat19.com/item/otrio;
Remover primeira ocorrencia de um elemento numa lista:
http://stackoverflow.com/questions/15857382/removing-first-occurrence-in-list-prolog
Por todas as soluções numa lista:
http://stackoverflow.com/questions/1468150/how-do-i-find-all-solutions-to-a-goal-in-prolog