Straight4

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo Straight_4_2:
Gonçalo Fernandes Pereira - up201705971
Maria João Senra Viana - up201604751

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

17 de Novembro de 2019

Resumo

Este trabalho consiste na conceção de um jogo de tabuleiro utilizando uma linguagem de programação lógica denominada Prolog.

O jogo escolhido foi o Straight4, jogo inspirado no Cinco em Linha, que neste caso vão ser quatro em linha. Foram implementados três modos de utilização: Humano/Humano, Humano/Computador e Computador/Computador. Nestes três modos todas as regras do jogo foram implementadas com sucesso.

Inicialmente, o principal obstáculo foi ultrapassar o facto de que esta era uma linguagem completamente nova com a qual nunca tinhamos trabalhado e como tal, a adaptação foi bastante morosa, mas ao longo do desenvolvimento do trabalho foi-se tornando mais intuitiva.

Com este trabalho conseguimos consolidar todos os conhecimentos adquiridos nas aulas teóricas e práticas e conseguimos conlcuir o quão eficiente a linguagem de Prolog é para resolver problemas de decisão.

Conteúdo

1 Introdução

Os objetivos deste trabalho foram implementar, em linguagem Prolog, um jogo de tabuleiro, com as regras de movimentação de peças (jogadas possíveis) e pelas condições de terminação de jogo com derrota ou vitória.

A estrutura deste relatório vai consistir na explicação das regras do jogo e a lógica deste e como tudo foi implementado.

2 O Jogo Straight 4

Straight 4 é um jogo de estratégia abstrata criado em 2019 por Binary Cocoa, feito para ser jogado por dois jogadores. O seu tempo de jogo é entre 5-10 minutos.

O material necessário para o jogo é um tabuleiro quadrado com 5x5 espaços e oito peças (4 de cor branca e 4 de cor preta).

2.1 Regras

Inicialmente o tabuleiro está vazio. Cada jogador tem uma cor associada (branca ou preta) e quatro pedras dessa mesma cor. Depois de decidido quem joga primeiro, cada jogador vai colocar, alternadamente, uma peça no tabuleiro.

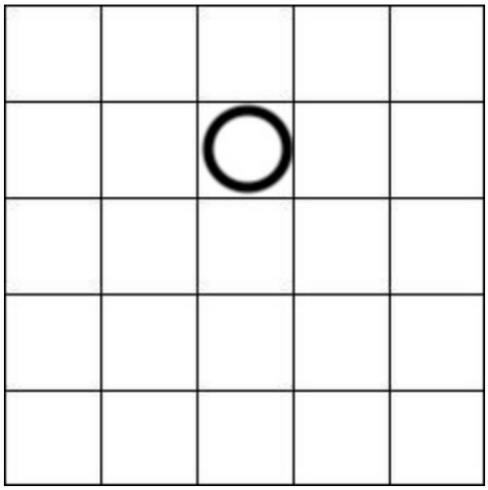

Figura 1: Tabuleiro do jogo com a peça do jogador inicial colocada (neste caso, o branco).

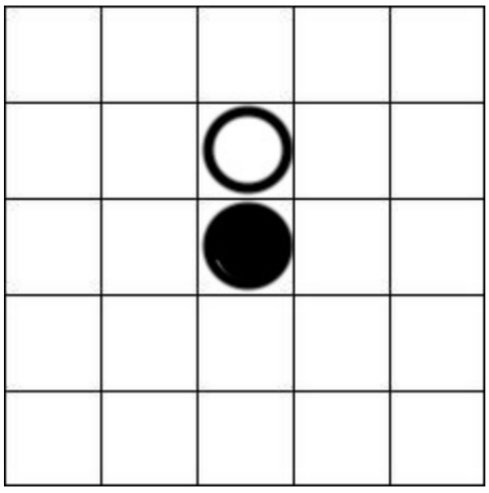

Figura 2: Tabuleiro do jogo após duas jogadas.

O objetivo do jogo é um jogador conseguir colocar as suas 4 peças numa linha reta (horizontal, vertical ou diagonal).

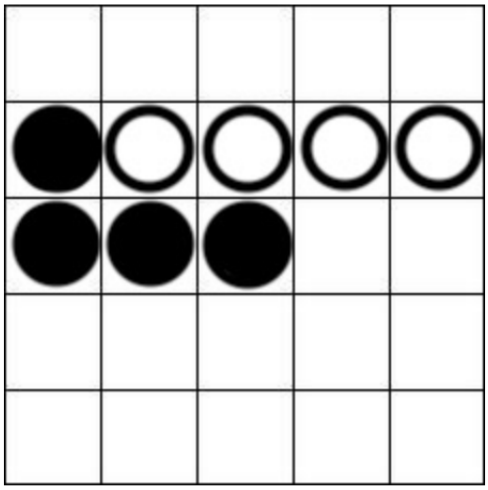

Figura 3: Exemplo de fim de jogo em que o jogador com as peças brancas ganha.

Se depois de colocadas todas as peças ninguém conseguir vencer, vai-se alterando a posição de uma peça de cada vez tentando colocar quatro peças em linha.

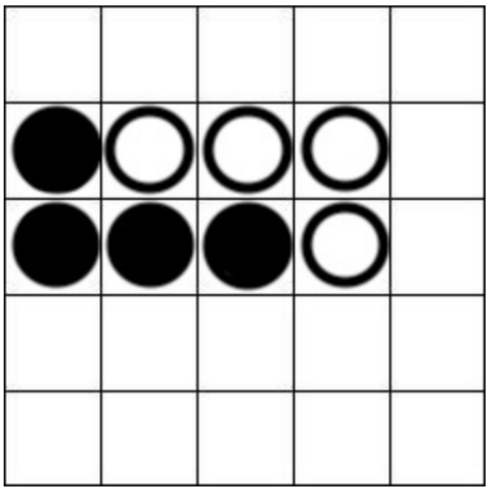

Figura 4: Exemplo com todas as peças colocadas e nenhum jogador ganha.

Vence o primeiro jogador a conseguir colocar 4 peças em linha.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

A representação interna do tabuleiro é feita através de uma lista de listas de tamanho fixo, sendo os espaços vazios identificados por '0', as peças pretas por '1' e as peças brancas por '2'. As figuras seguintes representam uma exemplificação do estado interno do jogo em situações distintas do jogo.

```
table([[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]
       ]).
```

Figura 5: Representação interna da situação inicial.

```
table([[0,0,0,0,0],
       [0,1,1,2,0],
       [0,2,2,1,0],
       [0,0,0,0,0],
       [0,0,0,0,0]
       ]).
```

Figura 6: Representação interna da situação intermédia.

```
table([[0,0,1,0,0],
       [0,1,1,2,0],
       [0,2,1,2,0],
       [0,0,1,0,0],
       [0,0,2,0,0]
       ]).
```

Figura 7: Representação interna da situação final.

3.2 Visualização do Tabuleiro

Para a visualização na consola são impressas duas linhas nos extremos do tabuleiro que indicam as colunas e as as linhas. Os espaços vazios são representados por '-', as peças pretas por uma bola preta e as peças brancas por uma bola branca.

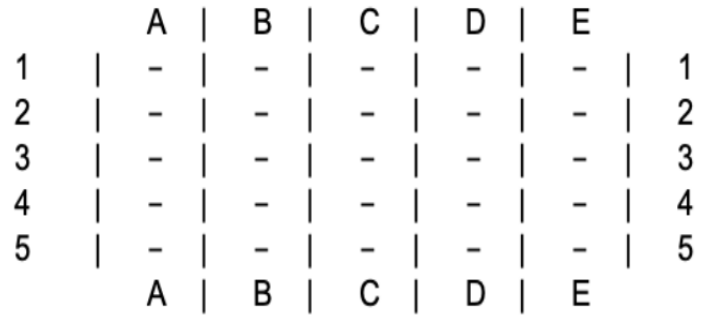

Figura 8: Representação do tabuleiro da situação inicial.

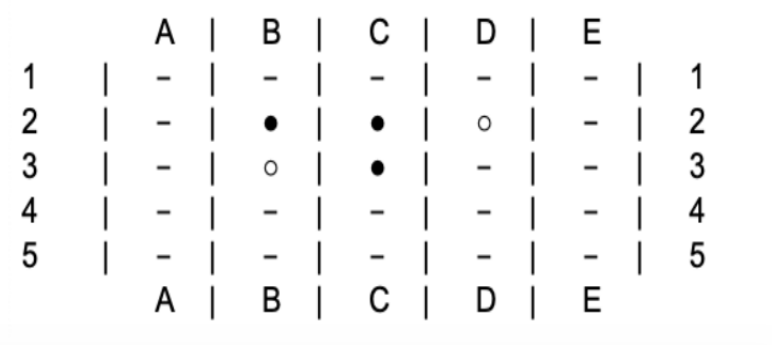

Figura 9: Representação do tabuleiro da situação intermédia.

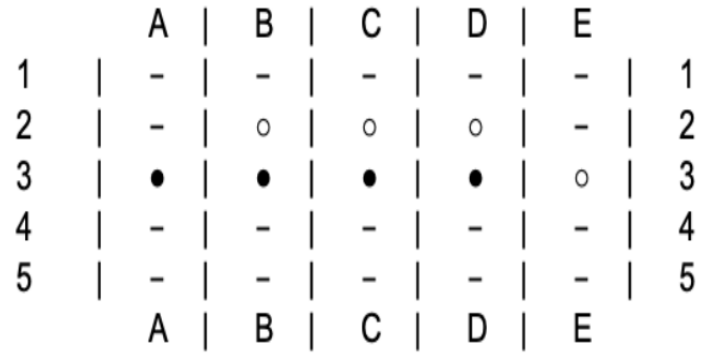

Figura 10: Representação do tabuleiro da situação final.

3.3 Lista de Jogadas Válidas

Para a validação das jogadas foram usados os predicados *checkPlay(+ Row, + Column, + Board, -Res)* que verifica se a jogada pretendida é valida se for retorna 0; Se falhar retorna 1. Este recorre ao predicado *isEmptyCell(+ Board,*

+ Row, + Column, - Res) que verifica se a célula (Row, Column) está vazia recorrendo a chamadas ao predicado *getValueFromMatrix(+ [H—T], + Row, + Column, - Value)*.

3.4 Execução de Jogadas

Para a validação e execução das jogadas, é usado o predicado *move(+Move, +Board, -NewBoard)*. Este predicado recebe como argumentos a jogada a validar e executar e o tabuleiro do jogo. Irá retornar um novo tabuleiro já com a jogada executada (se esta for válida, caso contrário irá falhar).

Para efeitos de validação, o predicado *checkPlay(+ Row, + Column, + Board, -Res)*. De seguida, com o predicado *replaceInMatrix(+ Board, + Row,+ Column, + Player, +NewBoard)*, a jogada vai ser introduzida no tabuleiro.

```
move(Board, Player, NewBoard):-
  readInput(Row, Column),
  checkPlay(Row, Column, Board, Bool),
  Bool =:= 0,
  replaceInMatrix(Board,Row,Column,Player,NewBoard).
```

Figura 11: Predicado *move(+Move, +Board, -NewBoard)*.

3.5 Final do Jogo

Após cada jogada é fundamental verificar o estado do jogo, pois, a qualquer momento, um dos jogadores pode ganhar. Esta verificação é feita pelo predicado *game_over(+Board, -Winner)*. Este recebe como argumento o tabuleiro e retorna o simbolo corresponde ao jogador que ganhou.

```
game_over(Board,Winner):-
    check_player_game_over(Board,1,Res1),
    check_player_game_over(Board,2,Res2),

    (Res1 =:= 1 -> Winner is 1;
        (Res2 =:= 1 -> Winner is 2;
            Winner is 0
        )
    ).
```

Figura 12: Predicado *game_over(+Board, -Winner)*.

Este predicado vai chamar o predicado *check_player_game_over(+ Board, + Player, - Res)* que por sua vez chama outros 4 predicados e caso algum deles se verifique, o jogo acaba.

```
check_player_game_over(Board,Player,Res):-
    check_row_win(Board,Player,1,1,0,_,Res1),
    check_column_win(Board, Player, 1,1,0,_,Res2),
    check_diagonal_right_win(Board,Player,Res3),
    check_diagonal_left_win(Board,Player,Res4),
    (Res1 =:= 1 -> Res is 1;
        (Res2 =:= 1 -> Res is 1;
            (Res3 =:= 1 -> Res is 1;
                (Res4 =:= 1 -> Res is 1;
                    Res is 0
                )
            )
        )
    ).
```

Figura 13: Predicado *game_over(+Board, -Winner).*

Existem 3 condições vencedoras: o jogador ter 4 peças seguidas na mesma linha, coluna ou diagonal. Deste modo cada predicado verifica se existe esse padrão no tabuleiro.

4 Conclusões

O presente trabalho exigiu um grande empenho por parte de ambos os elementos do grupo, tendo sido uma experiência recompensadora.

Consideramos que o nosso conhecimento de programação em lógica foi amplamente aumentado, tendo sido alcançado tudo o que objetivamos. Ao longo do desenvolvimento deste trabalho foram encontradas algumas dificuldades, como o pensamento recursivo e a melhor forma de desenvolver certos predicados.

Em suma, apesar de Prolog se modelar por um paradigma diferente do que estamos habituados, rapidamente nos acostumamos e aprendemos a apreciar a facilidade e a dificuldade que este apresenta, tendo culminado num trabalho no qual nos orgulhamos.

Bibliografia

- https://www.boardgamegeek.com/boardgame/284636/straight-4
- http://www.jbobsmoviereviews.com/wordpress/straight-4-review/
- https://binarycocoa.com/portfolio/straight-4/
- https://binarycocoa.com/2019/02/23/straight-4-now-available/