Faculdade de Engenharia da
Universidade do Porto

Programação Lógica

Relatório Intercalar

Zurero

Autores:
Diogo Teixeira up201606124
Márcia Meira up201604350

Contents

1 Introdução

O objetivo do projeto é simular um jogo de Zurero, implementando toda a sua lógica através da liguagem de paradigma lógico Prolog.

2 Jogo

2.1 Origem

O jogo foi desenvolviddo por Jordan Goldstein em 2009 e é uma variação do jogo Go-Moku.

2.2 Regras

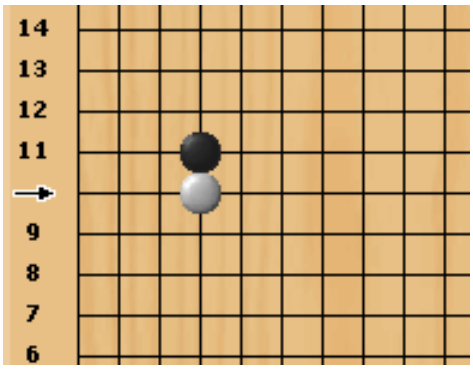
Zurero é jogado num tabuleiro 19x19, com dois jogadores, cada um com a sua cor, normalmente preto e branco. O jogador Preto joga primeiro, pondo uma peça preta no centro do tabuleiro.

Após esta jogada, os jogadores jogam alternativamente, *deslizando* uma peça da sua cor nas direções horizontal ou vertical. Uma pedra *desliza* até atingir uma outra pedra já no tabuleiro.

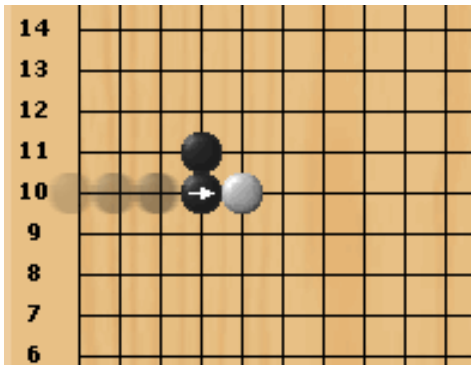
Não é permitido *deslizar* uma pedra numa linha ou coluna que não tenha pedra nela.

Se a pedra atingida tiver um espaço livre atrás de si, ela é *empurrada* para esse espaço e a pedra que a empurrou move-se para o espaço dela (Fig. 1a e 1b).

O objetivo do *Zurero* é colocar cinco pedras seguidas (verticalmente, horizontalmente ou diagonalmente). O primeiro jogador a alcançar o objetivo ganha o jogo. Se na mesma jogada ambos os jogadores ficam com pedras suas alinhadas (ambos alcançaram o objetivo), ganha o jogador que executou a última jogada.



(a) Antes da jogada.



(b) Depois da jogada (*A pedra preta empurra a pedra branca*).

Figure 1: Exemplo de uma jogada.

3 Lógica

3.1 Representação do Estado do Jogo

A criação do tabuleiro inicial é feita pelo predicado create_board e os seus auxiliares.

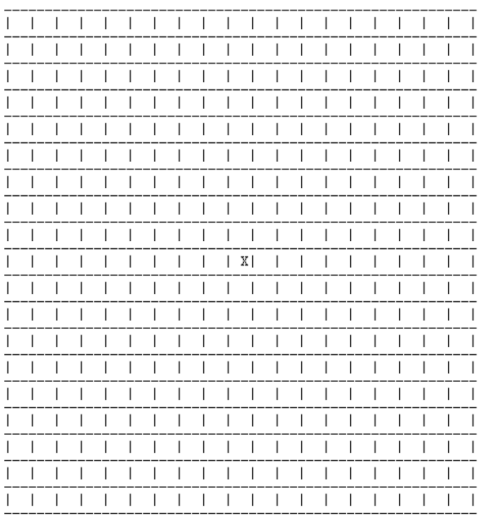
```
% Initializes empty board, with X piece on middle
create_board(PBoard) :-
        create_board_aux(19, [] , Board),
        set_piece(10,10,'X', Board, PBoard).

create_board_aux(0, Board, Board) :- !.
create_board_aux(N, L, Board) :-
        create_row(Row),
        N > 0,
        N1 is N-1,
        create_board_aux(N1, [Row|L], Board).

create_board_aux(N, L, Board) :-
        create_row(Row),
        N > 0,
        N1 is N-1,
        create_board_aux(N1, [Row|L], Board).

create_row(Row) :- create_row_aux(19, [], Row).

create_row_aux(0, Row, Row) :- !.
create_row_aux(N, L, Row) :-
        N > 0,
        N1 is N-1,
        create_row_aux(N1, [freeCell|L], Row).
```

Para a impressão do tabuleiro, é usado o predicado print_board e os seus auxiliares.

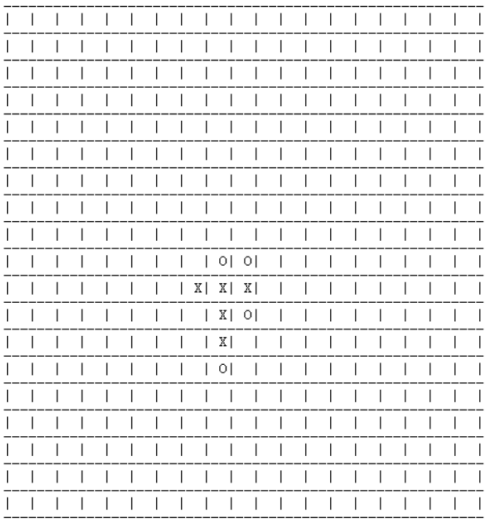
```
% Prints the Board, freeCell replaced with ' '
print_board(Board):-
        print_row_divider(29),
        print_board_aux(Board).

print_board_aux([]).
print_board_aux([Row|Rest]):-
        write('|  '),
        print_row(Row),
        print_board_aux(Rest).

print_row([]):-
        print_row_divider(29).

print_row([freeCell|Tail]):-
        write('  |  '),
        print_row(Tail).

print_row([Piece|Rest]) :-
        Piece \= freeCell,
        write(Piece),
        write('|  '),
        print_row(Rest).

print_row_divider(N):-
        nl,
        print_row_divider_aux(N),
        nl.

print_row_divider_aux(0).
print_row_divider_aux(N):-
        write('--'),
        NN is N-1,
        print_row_divider_aux(NN).
```

3.2 Visualização do Tabuleiro

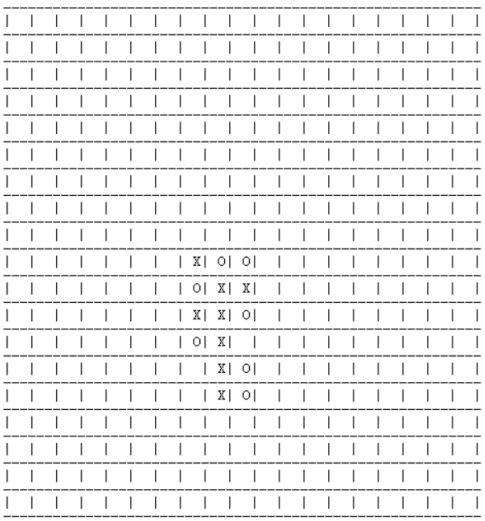
O estado do jogo é representado por uma lista de listas. As pedras Brancas são representadas por um 'O'e as Pretas por um 'X' (Fig.2).



(a) Tabuleiro inicial.



(b) Exemplo de um tabuleiro a meio de um jogo.



(c) Exemplo de um tabuleiro final. (*Preto ganhou.*).

Figure 2: Estados de um jogo.

3.3 Lista de Jogadas Válidas

Atendendo à natureza do jogo implementado, a obtenção de jogadas possíveis retorna duas listas, uma de jogadas verticais e outra de horizontais, cada uma com dois elementos, sendo eles os extremos (colunas ou linhas) entre os quais existem peças no tabuleiro. Tal sucede pois, em nenhum momento, o tabuleiro terá colunas ou linhas entre estes extremos que não estejam ocupadas. Desconsideramos o sentido da jogada pois, sendo válido na direção, poderemos jogar em ambas os sentidos a ela associados.

3.4 Execução de Jogadas

O predicado que executa a jogada possibilita, também, a deteção se a mesma é inválida, falhando se o for. Ou seja, no caso de o jogador poder, de facto, executar a jogada, este retorna o novo tabuleiro, caso contrário, falha.

3.5 Final do Jogo

A verificação do final do jogo dá-se após cada jogada, verificando primeiro o jogador que acabou de jogar e depois o seu adversário, devido à possibilidade de ambos os jogadores ganharem na mesma jogada, o vencedor é o último que jogou. A declaração do vencedor é feita dentro do predicado.

3.6 Avaliação do Tabuleiro

Devido à fluidez do jogo, e considerando a possibilidade de mover peças já posicionadas, aliado ao facto de não posicionarmos as peças e sim atirarmo-las, não conseguimos considerar uma jogada mais ou menos eficiente utilizando uma análise ao tabuleiro. Como tal, implementamos vários predicados que nos possibilitam ter em conta a disposição das peças, tanto do jogador como do seu adversário, e o sentido adequado duma jogada, de modo a chegarmos às jogadas pretendidas e que nos possibilitarão a vitória.

3.7 Jogada do Computador

Para este trabalho decidimos implementar três bots, todos eles com diferentes níveis de complexidade na escolha de jogadas. O primeiro faz somente uma escolha aleatória, tendo em conta todas as jogadas possíveis. O segundo, de

pensamento ligeiramente mais complexo, verifica primeiro se existe alguma jogada que alcançará a vitória, de modo a efectua-la. O último faz uma consideração do estado de jogo do adversário, de modo a tentar impedir que este progrida no posicionamento das suas peças, tendo em consideração primeiro se irá ganhar, e só depois o seu adversário.

4 Conclusão

A implementação deste jogo possibilitou-nos desenvolver um pensamento mais ligado a programação lógica e não imperativa. Apesar dos limitações que esta linguagem traz, a sua utilidade é evidente. Considerámos, contudo, que os predicados apresentados como obrigatórios limitam um pouco a liberdade de raciocínio e não se adequam totalmente a alguns dos jogos propostos, visto que a própria linguagem oferece melhores soluções de implementação. Sentimos, também, uma certa dificuldade em desenvolver os bots, visto que, até agora, não tivemos contacto com este tipo de implementações. Considerando, finalmente, a experiência como um todo, podemos simplesmente afirmar que gostaríamos de ter conseguido implementar uma inteligência artificial mais complexa, mas acreditamos que o trabalho em si teve em consideração as qualidades da linguagem e os utilizou da melhor forma encontrada.

5 Bibliografia

- BoardGameGeek
- igGameCenter