Título do Trabalho

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Fuse 02:
Carlos Vieira - up201606868
Christopher Abreu - up201604735

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 do jogo de tabuleiro fuse utilizando uma linguagem de programação em lógica denominada Prolog.

Fuse é um jogo de estratégia abstrato para 2 jogadores criado por Néstor Romeral Andrés e publicado pela Nestorgames. O Fuse é derivado do Feed The Ducks feito pelo mesmo designer, que se enquadra na família de outros 'jogos de empurrar / atrair', como Avverso (Henrik Morast) ou Momentum (PhilLeduc). Nste trabalho foram implementados três modos de jogo perfeitamente funcionais: Humano/Humano, Humano/Computador e Computador/Computador. Nestes três modos de jogo, todas as regras foram implementadas com sucesso.

Este trabalho permitiu a consolidação dos conhecimentos adquiridos nas aulas tanto teóricas como práticas da cadeira de Programação e Lógica e confirmar o quão eficiente é usar a linguagem de Prolog para resolver problemas de decisão.

Inicialmente, o principal obstáculo a ultrapassar foi o facto de esta linguagem ser nova para nós, pelo que a adaptação à mesma demorou um pouco. Tivemos também alguns problemas de implementação em algoritmos mais complicados como minimax.

Assim, foi concebido com sucesso um jogo simples, intuitivo e de fácil interação com o utilizador, com três modos à escolha e dois níveis de dificuldade.

1 Introdução

Este projeto foi desenvolvido no Sistema de Desenvolvimento SICStus Prolog no âmbito da unidade curricular de Programação e Lógica de 3º ano do curso Mestrado Integrado em Engenharia Informática e de Computação e tem como tema o jogo de tabuleiro Fabrik. O objetivo deste trabalho foi implementar, em linguagem Prolog, um jogo de tabuleiro e de peças, pelas regras de movimentação das peças (jogadas possíveis) e pelas condições de terminação do jogo com derrota, vitória ou empate.

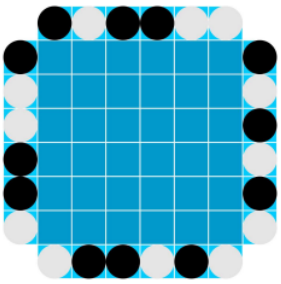
Estrutura do relatório:

- O Jogo Fabrik
- Lógica do Jogo
  - Representação do Estado do Jogo
  - Visualização do Tabuleiro
  - Lista de Jogadas Válidas
  - Execução de Jogadas
  - Final do Jogo
  - Avaliação do Tabuleiro
  - Jogada do Computador
- Conclusões

2 O Jogo Fuse

O Fuse é um jogo derivado do Feed The Ducks feito pelo mesmo designer, que se enquadra na família de outros 'jogos de empurrar / atrair', como Avverso (Henrik Morast) ou Momentum (PhilLeduc). Ele também usa o mecanismo de inserção de peças de jogos como 9tka (Adam Kałuża).

Fuse é um jogo de estratégia abstrato para 2 jogadores criado por Néstor Romeral Andrés e publicado pela Nestorgames, uma editora independente de jogos de tabuleiro, localizada na Espanha e administrada pelo próprio criador do jogo. É Jogado num tabuleiro 8 por 8 sem cantos, por turnos, alternando entre o jogador das peças brancas (ice) e o jogador das peças pretas (black).

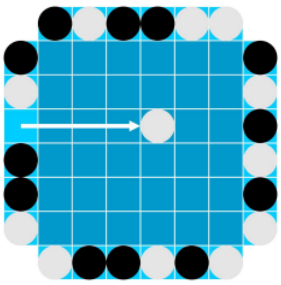


Inicialmente, o tabuleiro está vazio. São colocadas 24 peças aleatóriamente na parte exterior do tabuleiro (12 de cada cor), de forma que não haja mais

do que duas peças da mesma cor seguidas, incluindo nos cantos do tabuleiro. Podemos observar na imagem acima um exemplo de posicionamento inicial.

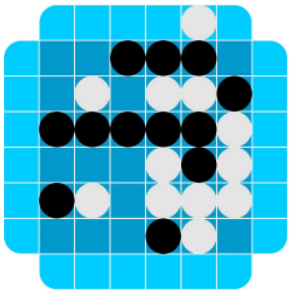
Começando com o branco, os jogadores executam a sua jogada alternadamente de forma a entrar no perímetro interior do tabuleiro (6x6). Nesta fase os jogadores são obrigados a cumprir as seguintes regras:

1. Os jogadores devem colocar uma peça na zona interior do tabuleiro, se possível. Caso contrário devem passar a jogada.
2. A jogada deve ser feita segundo a linha ou coluna inicial da peça, o número de casas desejado, de modo que:
   (a) Um ou mais discos possam ser empurrados para a frente não mais do que uma casa, como resultado da movimentação da peça (não obrigatório).
   (b) Todas as peças jogadas devem permanecer na área 6x6 do tabuleiro, o que significa que nenhuma peça pode ser jogada ou empurrada para fora da zona interior.



A imagem acima representa uma jogada inicial válida por parte do jogador das peças brancas.

O jogo acaba quando não houver jogadas possíveis para nenhum dos jogadores e o vencedor é aquele que conseguir ter mais peças ligadas ortogonalmente (é possível empatar).



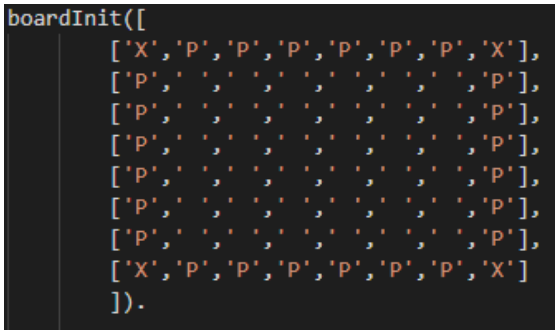
Acima encontra-se um exemplo de um tabuleiro no estado final em que o jogador das peças brancas é o vencedor (7-6).

3 Lógica do Jogo

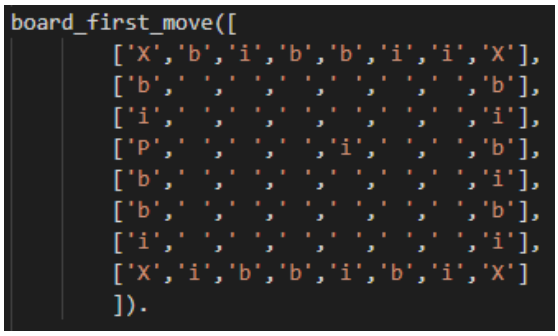
3.1 Representação do Estado do Jogo

O nosso tabuleiro é tratado em prolog como uma lista de listas

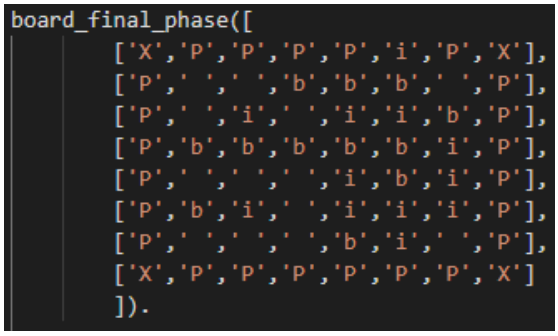
Situação inicial



Segunda jogada



Situação final



3.2 Visualização do Tabuleiro

Código utilizado para a visualização do tabuleiro na consola.

```
%%% VIEW BOARD

print_board([]).
print_board([H|T]) :-
    print_list(H),
    print_board(T).

print_list([]) :-
    nl.
print_list([H|T]) :-
    write(H),
    write(' | '),
    print_list(T).

%%%

display_game(_Board,1):-
    display_game_name,
    print_board(_Board).
```

3.3 Lista de Jogadas Válidas

Uma jogada válida é uma jogada que respeita as regras do jogo, ou seja, toda a jogada em que a peça se desloca ortogonalmente desde a sua posição inicial até no máximo à posição de uma peça no mesmo eixo, empurrando a mesma e as seguintes, se existirem. Para obtenção da lista das jogadas possíveis utilizamos o predicado move, que contém toda a lógica de uma jogada e falha no caso de esta ser inválida.

```
valid_moves(Board, Player, ListOfMoves) :-
    get_player_piece(Player, PlayerPiece),
    setof(NewBoard, X^Y^Steps^(move([X, Y, Steps], Board, NewBoard), get_piece(Board, X, Y, Piece), Piece == PlayerPiece), ListOfMoves).

valid_moves(Board, ListOfMoves) :-
    setof(NewBoard, X^Y^Steps^move([X, Y, Steps], Board, NewBoard), ListOfMoves).
```

3.4 Execução de Jogadas

O jogo tem um ciclo principal, denominado game_loop, que está encarregue de executar as jogadas, alternar o jogador, cálculo do score de cada jogador em tempo real e a verificação do estado do jogo.

```
game_loop(Board, Score, Player, Mode,CPU1,CPU2):-
    game_over(Board, Winner);
    print_board(Board),
    greet_player(Player),
    ask_for_move(Board,Player,NewBoard,CPU1,CPU2),
    change_player(Player,Mode,AuxPlayer),
    game_loop(NewBoard, Score, AuxPlayer, Mode,CPU1,CPU2).
```

3.5 Final do Jogo

Após cada jogada é fundamental verificar o estado do jogo, pois, a qualquer momento, um dos jogadores pode ganhar ou ocorrer um empate, isto é, caso não existam mais jogadas possíveis. De forma a poder verificar isto, foi implementado o predicado game_over, que recebe o tabuleiro atual e devolve o vencedor no caso de haver uma situação de fim de jogo. Este predicado faz uso do predicado já implementado, valid_moves, para verificar se num dado board há ou não ainda jogadas possíveis.

```
game_over(Board,Winner):-
    \+valid_moves(Board,  _ListOfMoves),
    value(Board,[IceScore,BlackScore]),
    IceScore > BlackScore,
    Winner = 'i'.

game_over(Board,Winner):-
    \+valid_moves(Board, _ListOfMoves),
    value(Board,[IceScore,BlackScore]),
    IceScore < BlackScore,
    Winner = 'b'.

game_over(Board,Winner):-
    \+valid_moves(Board,  _ListOfMoves),
    value(Board,[IceScore,BlackScore]),
    IceScore == BlackScore,
    Winner = 'Tie'.
```

3.6 Avaliação do Tabuleiro

Para calcular o estado de jogo, temos um predicado que itera sobre o tabuleiro interior a qualquer altura do jogo até encontrar uma peça de um jogado pretendido. Assim que encontra uma peça do tipo pretendido procura todas as ligações ortogonais com outras peças do mesmo tipo e soma o seu valor. Repete o processo para todo o tabuleiro e todos os conjuntos de peças da cor pretendida e no final retorna o maior aglomerado de peças ortogonalmente connectadas.

```
value(Board, [IceScore,BlackScore]):-
    value(Board, 0, IceScore),
    value(Board, 1, BlackScore).

value(Board, Player, HighestScore) :-
    get_inner_board(Board, InnerBoard),
    get_player_piece(Player, PlayerPiece),
    \+ clearCells,
    set_max,
    iterate_board(InnerBoard, InnerBoard, PlayerPiece, 0, 0), !,
    max(Max),
    HighestScore is Max.
```

3.7 Jogada do Computador

A jogada do cpu é escolhida com base nas jogadas válidas do jogador atual no tabuleiro atual. Este perdicado devolve o novo tabuleiro com uma jogada efetuada, que no caso do primeiro nível de dificuldade é aleatório.

```
choose_move(Board, Player, 1, Move) :-
    random_ai(Board, Player, Move).
```

```
random_ai(Board, Player, NewBoard):-
    valid_moves(Board,Player,ListofMoves),
    length(ListofMoves,Number),
    random(1,Number,RandomNumber - 1),
    nth1(RandomNumber,ListofMoves,NewBoard).
```

4 Conclusões

O projeto teve como principal objetivo aplicar o conhecimento adquirido nas aulas teóricas e práticas, assim como foi realizado no âmbito da unidade curricular de Programação em Lógica.

Ao longo do desenvolvimento deste projeto, foram encontradas algumas dificuldades, nomeadamente estruturação do código, pensamento recursivo e que melhor caminho tomar em cada predicado. Todas estas foram eventualmente superadas.

Evidentemente, existem aspetos que podiam ser melhorados no trabalho desenvolvido, como o uso de inteligência artificial, que não chegou a ser implementada como um nível de dificuldade no modo de jogo com o computador, devido a falta de tempo e a geração aleatória de um tabuleiro inicial que se demonstrou quase impossível de fazer com o nosso conhecimento atual e exigiu um enorme custo de tempo.

Em suma, o trabalho foi concluído com sucesso, e o seu desenvolvimento contribuiu positivamente para uma melhor compreensão da linguagem Prolog, que se demonstrou ser bastante complexa.