Symple

Relatório Intercalar

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 41:
André Duarte - 201100766
João Carlos Santos - 201106760

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

13 de Outubro de 2013

Resumo

O trabalho consistirá no desenvolvimento do jogo de tabuleiro "Symple" utilizando ProLog como linguagem de implementação. Posteriormente, o projeto terá ainda visualização gráfica em 3D recorrendo a OpenGL e C++.

Quando terminado o trabalho permitirá a dois jogadores humanos competirem entre si no mesmo computador.

1 Introdução

Temos por objetivo adquirir conhecimentos e sensibilidade na área de linguagens de Programação em Lógica (neste caso Prolog) deste forma desenvolvendo competências e raciocínios que de outra forma não apreenderíamos. Será tambem adquirido algum tacto no mundo do desenvolvimento de jogos.

Este jogo foi por nós escolhido principalmente devido à sua peculiaridade, pois o seu fluxo de jogo é bastante original. Apesar de aparentar ser bastante simples, o jogo apresenta um conjunto de regras algo extenso e baseia-se primariamente em princípios matemáticos. Apresentam-se então as seguintes características:

- Só existem dois tipos de jogadas possíveis (crescer e criar novo grupo).
- As peças de jogadores diferentes não interagem entre si, apenas bloqueiam posicionamento no lugar onde estão inseridas.
- Após o posicionamento as peças não podem mudar de sítio.
- O jogo acaba quando o tabuleiro se encontra preenchido.

Nota: Serão dadas mais informações sobre o fluxo do jogo na secção "O Symple".

2 O Symple

Symple é um jogo de estratégia abstrato em que dois jogadores competem para conseguir ocupar a maior área do tabuleiro com o menor número de grupos possível.

As jogadas são feitas alternadamente entre dois jogadores, de cor Branco e Preto. O jogador 1 é o primeiro a jogar e joga com as peças brancas. O jogador 2 joga com as peças pretas.

No seu turno, um jogador pode escolher entre as seguintes jogadas:

1. Colocar uma pedra no tabuleiro num sítio sem contacto com as outras pedras desta forma criando um novo grupo
2. Fazer crescer todos os grupos com uma pedra. Pedras que toquem em ambos os grupos contam como fazer crescer ambos os grupos. No entanto, se dois grupos crescem em uma pedra e apenas estas se tocam entre os dois grupos, a jogada é legal.

Para equilibrar o jogo existe uma regra extra: se nenhum jogador tiver crescido os seus grupos, o jogador 2, que joga com as peças pretas, pode, na mesma jogada, crescer todos os seus grupos e criar um novo grupo.

O jogo acaba quando o tabuleiro é preenchido. A pontuação é determinada pelo número de pedras que cada jogador tem no tabuleiro menos 'P' vezes o número de grupos que o jogador tem, em que 'P' é um número par maior ou igual a 4. Com uma penalidade par e um tamanho de tabuleiro ímpar não são possíveis empates.

3 Representação do Estado do Jogo

Visto tratar-se de um jogo com tabuleiro quadrado, representamos-o numa lista de listas dinâmica.

O valor em cada posição representará se o campo está vazio (valor 0), se tem uma peça branca (valor 1) ou se tem uma peça preta (valor 2).

Representação da lista de listas inicial, com um tamanho três:
B = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
Com possíveis jogadas:
B = [[0, 1, 0], [0, 2, 0], [1, 0, 0]]
Finalizado o jogo não podem haver campos vazios:
B = [[1, 1, 1], [2, 2, 2], [1, 1, 2]]

4 Visualização do Tabuleiro

A visualização do tabuleiro é feita através da chamada da função printBoard(B), em que B é o tabuleiro.

Código da implementação:

```
printBoard(B) :-
        printBoardAux(B).

printBoardAux([]).
printBoardAux([H]) :-
        printList(H).
printBoardAux([X | Y]) :-
        printList(X),
        write(' '),
        nl,
        printBoardAux(Y).

printList(L) :-
        printListAux(L).

printListAux([]).
printListAux([H]) :- write(H).
printListAux([X | Y]) :-
    write( X ),
    write(' '),
    printListAux(Y).
```

Visto que repáramos que seria contraprodutivo estar a contar as colunas e linhas para proceder à jogada, pusemos a numeração no topo e no lado esquerdo.

Ficando da seguinte forma:

```
| ?- playSymple(9).
  1 2 3 4 5 6 7 8 9
1 0 0 0 0 0 0 0 0 0
2 0 0 0 0 0 0 0 0 0
3 0 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0
6 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 0 0
8 0 0 0 0 0 0 0 0 0
9 0 0 0 0 0 0 0 0 0
```

Código de implementação:

```
printBoard(B) :-
        N1 is 1,
        N2 is 1,
        write('␣'),
        printBoardTop(B, N1),
        nl,
        printBoardAux(B, N2).

printBoardTop([], N).
printBoardTop([X | Y], N) :-
        write('␣'),
        write(N),
        N1 is N+1,
        %N is N1+1,
        printBoardTop(Y, N1).


printBoardAux([], N).
printBoardAux([H], N) :-
        write(N),
        printList(H).
printBoardAux([X | Y], N) :-
        write(N),
        printList(X),
        nl,
        N1 is N+1,
        printBoardAux(Y, N1).


printList(L) :-
        printListAux(L).

printListAux([]).
printListAux([H]) :- write('␣'), write(H).
printListAux([X | Y]) :-
   write('␣'),
   write( X ),
   printListAux(Y).
```

5 Movimentos

Os movimentos possíveis são apenas dois.. Expandir todos os seus grupos, ou criar um novo. Existe ainda um movimento especial, para o jogador 2, quando o jogador 1 (das peças brancas) ainda não tenha efectuado a jogada de expansão nenhuma vez, que é a combinação dos dois movimentos simples.

Expandir: expand(Player).

Criar novo grupo: createGroup(Player, X, Y).

Movimento especial: specialBlack(X, Y).

6 Conclusões e Perspectivas de Desenvolvimento

Concluímos que será desafiador realizar este projecto para a unidade curricular PLOG e que a escolha do jogo foi a acertada.

Neste momento falta-nos ainda grande parte do trabalho, aproximadamente 75%.

Bibliografia

[1] Board Game Geek, *Symple*. http://boardgamegeek.com/boardgame/106341/symple, 2010. Online em Outubro de 2013.

[2] MindSports, *Symple*. http://www.mindsports.nl/index.php/arena/symple, 2010. Online em Outubro de 2013.

A Código

```
% Create

createBoard(B, S) :-
        length( B, S ),
        createLists(B,S).

createLists([], S).
createLists([H|T], S) :-
        length( H, S ),
        fillList( H ),
        createLists(T, S).

fillList( [] ).
fillList( [X|Xs] ) :-
    X is 0,
    fillList( Xs ).

% Print

printBoard(B) :-
        N1 is 1,
        N2 is 1,
        write('␣'),
        printBoardTop(B, N1),
        nl,
        printBoardAux(B, N2).

printBoardTop([], N).
printBoardTop([X | Y], N) :-
        write('␣'),
        write(N),
        N1 is N+1,
        %N is N1+1,
        printBoardTop(Y, N1).


printBoardAux([], N).
printBoardAux([H], N) :-
        write(N),
        printList(H).
printBoardAux([X | Y], N) :-
        write(N),
        printList(X),
        nl,
        N1 is N+1,
        printBoardAux(Y, N1).


printList(L) :-
        printListAux(L).
```

```
printListAux([]).
printListAux([H]) :- write(' '), write(H).
printListAux([X | Y]) :-
    write(' '),
    write( X ),
    printListAux(Y).

% Play

playSymple :-
        write('Choose board size: '),
        read(S),
        playSympleAux(S).

playSymple(S) :-
        playSympleAux(S).

playSympleAux(S) :-
        createBoard(B, S),
        printBoard(B).
```