Relatório do Trabalho Prático

Jogo do Nim

Trabalho 3

Inteligência Artificial

João Oliveira, 21133 E.I.

Filipe Sezões, 21171 E.I.

Objectivo

Este trabalho tem como objectivo a representação de um jogo de dois jogadores como um problema de pesquisa no espaço de estados. O jogo escolhido foi o jogo do Nim.

Este jogo consiste em, dado um número de filas com tamanho variável, os jogadores, de forma alternada, retirarem um número arbitrário de elementos de uma fila à escolha.

O jogador que retirar o último elemento perde o jogo.

1. Representação de Estados

Como o número de filas é variável, decidiu-se representar cada estado do jogo como uma lista de elementos, onde cada elemento da lista corresponde a uma fila. O primeiro elemento da lista indica o número de elementos da primeira fila, o segundo elemento da lista indica o número de elementos da segunda fila, e assim sucessivamente.

O número inicial de elementos em cada fila é calculado da seguinte forma: **2*n-1**, onde o n representa o numero da fila. Por exemplo um jogo com 4 filas irá ser representado como: **[1,3,5,7]**.

2. Predicado Terminal

O perdicado terminal ocorre sempre que o estado e terminal. No caso do jogo do Nim um estado é quando todas as filas estiveram vazias, ou seja, a soma dos elementos da lista que representa o jogo seja 0.

```
terminal(A,Res):-
        soma(A,Res),
        Res = 0.

soma([],0).
soma([A|R],Res):-
        soma(R,Res1),
        Res is (A+Res1).
```

3.

Função de Utilidade

A função de utilidade é utilizada quando se atinge um estado é terminal.

Quando se atinge um estado terminal verifica-se a profundidade a que está. Se estiver a uma profundidade par, sabe-se que o ultimo jogador a tirar uma peça foi o adversário, logo a vitória é do jogador principal. Assim sendo a função retorna o valor 15 que significa que o jogador principal ganhou.

Se o estado terminal corresponder a uma profundidade impar, significa que o jogador vitorioso foi o adversário, logo a função devolve o valor -15.

```
valor(0, 15, P):-
    X is P mod 2,
    X = 0,
    !.

valor(0, -15, _):-!.
```

4.

Algoritmo MiniMax

O algoritmo MiniMax utilizado foi o disponibilizado nas aulas práticas alterando alguns aspectos de forma a adaptar-se ao nosso jogo.

5.

Algoritmo AlfaBeta

O algoritmo AlfaBeta utilizado corresponde a uma adaptação do algoritmo disponibilizado no livro "***Prolog programming for artificial intelligence"*** de *Ivan Bratko*. No entanto tivemos algumas dificuldades em conseguir fazer essa adaptação pelo que o seu funcionamento não é o mais correcto. Verificámos, em relação ao Minimax, que são consultados menos nós, no entanto demora mais tempo.

6.

Corte em Profundidade e Função de Avaliação

De modo a tornar a pesquisa mais optimizada acrescentamos, a ambos os algoritmos, um predicado que, ao atingir um limite de profundidade previamente definido, retorna um valor relativo ao estado que se encontra nesse limite. Assim impede-se que o programa "rebente" caso seja necessário ir muito fundo na pesquisa do estado terminal.

- Predicado de Corte no MiniMax:

```
minimax_valor(Ei, Val, P):-
                profundidade_limite(P1),
                P = P1,!,
                f_avalia(Ei,Val,P), !.
```

- Predicado de Corte no AlfaBeta:

```
alphabeta( Ei,_,_,_, Val,P,1) :-
                 profundidade_limite(P1),
                P1 = P,!,
                f_avalia(Ei,Val,P).
```

Em relação à função de avaliação esta foi feita com base na teoria de vitória do jogo do **Nim**. Esta teoria diz que um estado é estado de vantagem se o xor entre o número de elementos de cada uma das filas for igual a 0. A esta "soma" dá-se o nome de *nim-sum*. Portanto se a *nim-sum* der um resultado 0 numa profundidade par, significa que o jogador principal atingiu um estado de vantagem. Se a *nim-sum* não der 0 e estiver numa profundidade impar significa que o adversário não atingiu um estado de vantagem, o que também é bom para o jogador principal. Nestes dois casos, favoráveis ao jogador principal, a função de avaliação devolve o valor 3. Nas outras combinações possíveis devolve o valor -3, se o jogador adversário atingir um estado de vantagem. Caso nenhum dos jogadores atinja um estado de vantagem, a função devolve o valor 0.

Nesta teoria existe a excepção para o caso onde todas as filas têm 0 ou 1 elemento. Neste caso se o número de filas com 1's for impar e a profundidade par, então o estado é de vantagem para o jogador principal. Também é estado de vantagem para o jogador principal se o numero de filas com 1's for par e a profundidade do estado seja impar. Nestes casos devolve o valor 3. Nos outros casos, onde o estado de vantagem corresponde ao jogador adversário, devolve o valor -3.

- <u>Regra de filas só com 0's ou 1's:</u>

```
f_avalia(Estado,Val,P):- so_uns(Estado,Val1),!,
                         bom_mau(Val1,Val,P).

so_uns([],0).
so_uns([A|R],V):-  A < 2,
                   so_uns(R,V1),
                   V is A+V1.

bom_mau(Val1,3,P):-  Y is P mod 2,Y \=0,
                     X is Val1 mod 2, X \=0,!.

bom_mau(Val1,3,P):-  Y is P mod 2, Y =0,
                     X is Val1 mod 2, X =0,!.

bom_mau(_,-3,_).
```

- Regra de *nim-sum*:

```
nim_sum([V],V).
nim_sum([A,B|R],V):-
        C is A^B,
        nim_sum([C|R],V),!.

bom_mau2(0,3,P):-
        Y is P mod 2,
        Y =0,!.

bom_mau2(0,-3,P):-
        Y is P mod 2,
        Y \=0,!.

bom_mau2(X,0,_):-
        X\=0,!.
```

7.

Agente Inteligente

Foi implementado, também, um Agente Inteligente. Este agente possibilita visualizar um jogo entre dois PC's ou então o utilizador jogar contra o Pc.

A dificuldade do jogo depende do nível do corte, logo quanto maior for a profundidade limite definida mais difícil será ganhar ao PC.

Em relação ao jogo PC vs PC ganha aquele que conseguir primeiro atingir um estado de vantagem, pois quando o conseguir vai impedir o adversário de alcançar um outro estado de vantagem.