UNIVERSIDADE
DE ÉVORA

Relatório do Trabalho Prático Final de Inteligência Artificial

Rúben Peixoto e Vanessa Santos
37514 e 34191

Junho de 2020

Introdução

O grupo decidiu ser avaliado pelas duas versões do trabalho. Neste projecto a linguagem de programação utilizada foi SWIProlog.

Relativamente ao relatório, na secção "Descrição do Problema" vamos falar na forma como interpretamos o problema e quais foram as estruturas de dados implementadas, na secção "1ª Versão do Trabalho" iremos falar sobre qual o algoritmo utilizado para resolver este problema, e, na secção "2ª Versão do Trabalho (Extensão)", falaremos não só do algoritmo utilizado, como também dos resultados obtidos da competição entre o algoritmo da primeira versão e o algoritmo implementado nesta versão.

Descrição do Problema

Para representar o tabuleiro do jogo mais a pontuação dos dois jogadores, foi um utilizado um termo composto com a seguinte estrutura:

```
game( [0, 0] , [[4,4,4,4,4,4],[4,4,4,4,4,4]] ).
```

em que o primeiro argumento deste termo representa uma lista com as pontuações dos dois jogadores em que o primeiro elemento da lista é a pontuação do algoritmo e o segundo elemento a pontuação do adversário, por fim, o segundo argumento do termo composto, que é uma lista de listas, representa o tabuleiro do jogo.

Os predicados mais importantes para o funcionamento do programa são:

- **initial_state(-Board)**: Devolve o termo composto "game" com as pontuações a zero e o tabuleiro em que cada espaço deste contém quatro sementes.

- **final_state(+Board)**: Recebe o termo composto "game" e verifica se a pontuação de algum dos jogadores é maior ou igual a 25 e, verifica também, se o jogo entrará em loop, isto é, este predicado tenta certificar-se se na jogadas seguintes, os jogadores conseguirão realizar jogadas de modo a algum deles conseguir obter mais sementes. Se esta última condição não se verificar o jogo pára.

- **utility(+Score, -Utility)**: Tendo em conta a pontuação de cada jogador, o valor "Utility" assume o valor 1 se o computador vencer, 0 se empatar e -1 caso o adversário ganhe.

- **pass_beans(+Board, +Player, -PassOrNot, -NewBoard)**: Algoritmo que verifica se um jogador tem ou não peças. Se não tiver e se for possível retirar do adversário, fá-lo. A variável "PassOrNot" funciona como uma flag para verificar se houve transferência de sementes caso tenha sido necessário.

- **op(+Board, +Col, +Player, -NewBoard)**: Algoritmo que representa a jogada do computador. Para isso, o computador retira sementes de espaço, verifica de acordo com as regras, se é possível retirar essa quantidade de sementes, se sim, coloca as sementes retiradas às seguintes posições uma semente em cada espaço, até que não haja mais

sementes. Por fim, este algoritmo tenta recolher as sementes, caso na ultima posição em que foi adicionado os feijões, contenha 2 ou 3 feijões. Se assim for, o algoritmo vai recolhendo as sementes até chegar ao fim do tabuleiro do adversário ou até não haver nenhum espaço com 2 ou 3 sementes. Recolhidas as sementes, essa quantidade é adicionada à pontuação do respectivo jogador.

1ª Versão do Trabalho

Para esta versão utilizamos o algoritmo "Minimax", pois este algoritmo é utilizado em jogos multiplayer.

Para executar o predicado que indica a melhor jogada basta executar o comando:

```
swipl -t best minimax.pl
```

Para executar o programa que permite jogar pessoa contra computador basta usar o comando:

```
swipl -t play minimax.pl
```

Este algoritmo tem um temporizador default de cinco segundos. Se for necessário alterar o tempo, é preciso ir ao predicado "main" no ficheiro "alpha_beta.pl" e alterar o "timer5(-Timer)" para o "timer15(-Timer)" ou o "timer30(-Timer)".

Numa tentativa de tornar o algoritmo mais "inteligente", o grupo achou melhor que o caso base para calcular o tanto o valor da utilidade mínimo como o valor da utilidade máximo corresponde-se às seguintes condições:

1. Passado o tempo do temporizador, o valor da utilidade é calculado com base na pontuação do jogador. Se a pontuação actual for maior que a pontuação anterior, o valor da utilidade é a diferença entre a pontuação actual e a pontuação anterior. Se for igual ou menor o valor é zero.

2. Se for estado final compara e verifica se a pontuação do computador é superior à do adversário. Se isto se verificar, o valor da utilidade é 1, se a pontuação for igual é zero e se for menor é -1.

2ª Versão do Trabalho (Extensão)

Para esta versão utilizamos o algoritmo "Alpha Beta Search" por este sair mais eficiente comparado com o "Minimax".

Tal como pedido no enunciado, o teve o cuidado para executar este programa na linha de comandos e, para isso, basta escrever no terminal:

```
swipl -t <parâmetro inicial> alpha_beta_none.pl
```

em que o parâmetro inicial pode ser: "-p" se querermos que o computador jogue em primeiro lugar ou "-s" se quisermos que o computador jogue em segundo.

Este algoritmo tem um temporizador default de cinco segundos. Se for necessário alterar o tempo, é preciso ir ao predicado "main" no ficheiro "alpha_beta.pl" e alterar o "timer5(-Timer)" para o "timer15(-Timer)" ou o "timer30(-Timer)".

Numa tentativa de tornar o algoritmo mais "inteligente", o grupo achou melhor que o caso base para calcular o tanto o valor da utilidade mínimo como o valor da utilidade máximo corresponde-se às seguintes condições:

1. Passado o tempo do temporizador, o valor da utilidade é calculado com base na pontuação do jogador. Se a pontuação actual for maior que a pontuação anterior, o valor da utilidade é a diferença entre a pontuação actual e a pontuação anterior. Se for igual ou menor o valor é zero.

2. Se for estado final compara e verifica se a pontuação do computador é superior à do adversário. Se isto se verificar, o valor da utilidade é 1, se a pontuação for igual é zero e se for menor é -1.

Numa simulação de seis jogos verificou-se, independentemente de se alterar os tempos dos temporizadores, que o algoritmo "Alpha Beta Search" venceu sempre com a pontuação. Este resultado é expectável pois, embora estes algoritmos funcionem de forma muito idêntica, o algoritmo "Alpha Beta Search", à medida que o algoritmo vai sendo executado, este aplica cortes aos ramos que não sejam interessantes calcular. Com isto, o algoritmo tem mais tempo para fazer outras operações e assim descer na árvore de pesquisa.

Por isso, e como foram usados temporizadores, aquele algoritmo que mais rápido chegar a um resultado, melhor.

Tendo em conta os resultados obtidos o grupo prefere utilizar o algoritmo "Alpha Beta Search" (ficheiro: alpha_beta_none.pl) para o torneio.

Uma ultima nota, a pedido do professor, enviamos dois ficheiros com o algoritmo "Alpha Beta Search" (alpha_beta.pl e alpha_beta_none.pl), a diferença entre eles é que o programa do ficheiro "alpha_beta.pl", ao contrário do ficheiro "alpha_beta_none.pl", imprime o tabuleiro para o output, a pontuação dos jogadores e as opções/acções que um jogador pode tomar.

Bibliografia

1. Slides das aulas teóricas.
2. Documentação do SWIProlog.
3. Artificial Intelligence, A Modern Approach.