Manual de Utilizador do Projeto Nº1: Época de Recurso

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal 2024/2025
Estudante: **Rodrigo Baptista**, número **202200217**

1. Introdução

Este manual tem como objetivo fornecer uma explicação clara e detalhada sobre a utilização do programa desenvolvido para o jogo Adji-boto* no âmbito da Unidade Curricular de Inteligência Artificial.
O Adji-boto* é um jogo de estratégia pertencente à família Mancala, onde o jogador deve capturar o maior número de peças possíveis utilizando regras específicas.
Nesta versão simplificada, o objetivo é averiguar qual a solução óptima para se capturar todas as peças do tabuleiro no menor número de jogadas, onde apenas existe um jogador e o mesmo pode jogar em qualquer célula.
O programa foi desenvolvido em Lisp e implementa vários métodos de procura, incluindo Procura em Largura (BFS), Procura em Profundidade (DFS), A* e SMA*. O objetivo principal do software é encontrar a melhor sequência de jogadas que conduz à solução do problema, analisando a eficiência dos algoritmos e heurísticas utilizadas, não sendo possível jogar diretamente.

2. Como Jogar

1. Fazer download dos quatro ficheiros e colocá-los na mesma pasta: **projeto.lisp**, **puzzle.lisp**, **procura.lisp**, **problemas.dat**.
2. Abrir o LispWorks e compilar o ficheiro projeto.lisp, que irá carregar automaticamente os outros três.
3. Abrir uma nova janela do Listener e escrever o comando `(initialize)`.
4. Será pedido o número do problema a resolver e, se desejar adicionar um problema à lista, basta abrir o ficheiro problemas.dat e colocar o mesmo (seguindo o formato presente) numa linha arbitrária.
5. Será perguntado o algoritmo de procura a utilizar, que são BFS, DFS, A* e SMA*. No caso do DFS, é pedida a profundidade máxima da árvore de pesquisa. No caso do A* e SMA*, como são algoritmos de procura informados, terá que ser escolhida a heurística a utilizar, cujas opções são a heurística base e a heurística avançada. Para o algoritmo SMA* também é pedido o número limite de nós a manter em memória.
6. Irá então aparecer o *output* na consola, em que a solução do problema é a leitura normal dos nós, e, aparecerá também no ficheiro log.dat o mesmo *output* (mais detalhes na secção sobre a informação produzida).

Exemplo de BFS:

```
CL-USER 1 > (initialize)

Available problems:
1: ((0 0 0 0 0 2) (0 0 0 0 4 0))
2: ((2 2 2 2 2 2) (2 2 2 2 2 2))
3: ((0 3 0 3 0 3) (3 0 3 0 3 0))
4: ((1 2 3 4 5 6) (6 5 4 3 2 1))
```

```
5: ((2 4 6 8 10 12) (12 10 8 6 4 2))
6: ((48 0 0 0 0 0) (0 0 0 0 0 48))
7: ((8 8 8 8 8 8) (8 8 8 8 8 8))
Problem number: 1

What algorithm to use?
1 - BFS
2 - DFS
3 - A*
4 - SMA*
Algorithm: 1

Time taken: 4ms
The number of total generated nodes is: 25
The number of total expanded nodes is: 11
The penetrance is: 0.16000
Solution depth is: 4
Initial State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
NODE:
- State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
- Depth: 0
NODE:
- State: ((0 0 0 0 1 0) (0 0 0 0 4 0))
- Depth: 1
NODE:
- State: ((0 0 0 0 2 1) (0 0 0 0 0 1))
- Depth: 2
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 1))
- Depth: 3
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 0))
- Depth: 4
```

Exemplo de DFS:

```
CL-USER 1 > (initialize)

Available problems:
1: ((0 0 0 0 0 2) (0 0 0 0 4 0))
2: ((2 2 2 2 2 2) (2 2 2 2 2 2))
3: ((0 3 0 3 0 3) (3 0 3 0 3 0))
4: ((1 2 3 4 5 6) (6 5 4 3 2 1))
5: ((2 4 6 8 10 12) (12 10 8 6 4 2))
6: ((48 0 0 0 0 0) (0 0 0 0 0 48))
7: ((8 8 8 8 8 8) (8 8 8 8 8 8))
Problem number: 1

What algorithm to use?
1 - BFS
2 - DFS
3 - A*
```

```
4 - SMA*
Algorithm: 2

What's the max depth of the search tree generated by the DFS algorithm?
30

Time taken: 2ms
The number of total generated nodes is: 11
The number of total expanded nodes is: 6
The penetrance is: 0.54545
Solution depth is: 6
Initial State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
NODE:
- State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
- Depth: 0
NODE:
- State: ((0 0 0 0 1 0) (0 0 0 0 4 0))
- Depth: 1
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 4 0))
- Depth: 2
NODE:
- State: ((0 0 0 0 1 1) (0 0 0 0 0 1))
- Depth: 3
NODE:
- State: ((0 0 0 0 0 1) (0 0 0 0 0 1))
- Depth: 4
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 1))
- Depth: 5
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 0))
- Depth: 6
```

Exemplo de A*:

```
CL-USER 1 > (initialize)

Available problems:
1: ((0 0 0 0 0 2) (0 0 0 0 4 0))
2: ((2 2 2 2 2 2) (2 2 2 2 2 2))
3: ((0 3 0 3 0 3) (3 0 3 0 3 0))
4: ((1 2 3 4 5 6) (6 5 4 3 2 1))
5: ((2 4 6 8 10 12) (12 10 8 6 4 2))
6: ((48 0 0 0 0 0) (0 0 0 0 0 48))
7: ((8 8 8 8 8 8) (8 8 8 8 8 8))
Problem number: 1

What algorithm to use?
1 - BFS
2 - DFS
3 - A*
```

```
4 - SMA*
Algorithm: 3

What heuristic function use?
1 - Base Heuristic
2 - Advanced Heuristic
Heuristic: 1

Time taken: 5ms
The number of total generated nodes is: 12
The number of total expanded nodes is: 7
The penetrance is: 0.33333
Solution depth is: 4
Initial State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
NODE:
- State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
- Depth: 0
- Heuristic: 0
- Cost: 0
NODE:
- State: ((0 0 0 0 1 0) (0 0 0 0 4 0))
- Depth: 1
- Heuristic: 5
- Cost: 6
NODE:
- State: ((0 0 0 0 2 1) (0 0 0 0 0 1))
- Depth: 2
- Heuristic: 4
- Cost: 6
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 1))
- Depth: 3
- Heuristic: 1
- Cost: 4
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 0))
- Depth: 4
- Heuristic: 0
- Cost: 4
```

Exemplo de SMA*:

```
CL-USER 1 > (initialize)

Available problems:
1: ((0 0 0 0 0 2) (0 0 0 0 4 0))
2: ((2 2 2 2 2 2) (2 2 2 2 2 2))
3: ((0 3 0 3 0 3) (3 0 3 0 3 0))
4: ((1 2 3 4 5 6) (6 5 4 3 2 1))
5: ((2 4 6 8 10 12) (12 10 8 6 4 2))
6: ((48 0 0 0 0 0) (0 0 0 0 0 48))
7: ((8 8 8 8 8 8) (8 8 8 8 8 8))
```

```
Problem number: 1

What algorithm to use?
1 - BFS
2 - DFS
3 - A*
4 - SMA*
Algorithm: 4

What heuristic function use?
1 - Base Heuristic
2 - Advanced Heuristic
Heuristic: 1

What's the memory limit of SMA* (number of nodes to keep), number in range [100, 500]?
100

Time taken: 4ms
The number of total generated nodes is: 12
The number of total expanded nodes is: 7
The penetrance is: 0.33333
Solution depth is: 4
Initial State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
NODE:
- State: ((0 0 0 0 0 2) (0 0 0 0 4 0))
- Depth: 0
NODE:
- State: ((0 0 0 0 1 0) (0 0 0 0 4 0))
- Depth: 1
NODE:
- State: ((0 0 0 0 2 1) (0 0 0 0 0 1))
- Depth: 2
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 1))
- Depth: 3
NODE:
- State: ((0 0 0 0 0 0) (0 0 0 0 0 0))
- Depth: 4
```

Nota: É crucial que, sempre que se inicie uma nova sessão de procura, abra uma nova janela do Listener e volte a correr o `(initialize)`.

3. Informação Produzida

No mesmo diretório em que colocou os quatro ficheiros do puzzle (puzzle.lisp, projeto.lisp, procura.lisp e problemas.dat), irá ser produzido um ficheiro denominado log.dat, este ficheiro mantém um registo de todas as procuras efetuadas no sistema que está a correr o programa e oferece uma análise estatística de todos os elementos presentes em cada procura feita, tais como caminho até à solução, profundidade da mesma e medidas de desempenho.

4. Limitações

Não existem limitações de natureza do utilizador no programa.