15 - Puzzle

Tarefa - IA

Chris G. Chinedozie

O problema:

O 15-puzzle é um puzzle deslizante, que consiste em um uma matriz 4x4 de 15 peças numeradas de 1 a 15 de forma não ordenada e um espaço em branco. O objetivo do puzzle é organizar as peças em ordem numérica movendo-as uma a uma, utilizando o espaço em branco para alternar entre os estados. É uma variante do n-puzzles, um problema famoso na ciência da computação que incorpora heurística. estudos anteriores concluíram que os puzzles com um número ímpar de permutações são insolúveis, enquanto todos os puzzles com permutações pares são sempre solucionáveis (Johnson e Story 1879.) Portanto, depois de identificar a natureza da permutação do puzzle, retornar uma resposta apropriada é simples e direto.

Objetivo:

Meu objetivo é implementar um algoritmo que pega um puzzle aleatório como um estado inicial, avalia sua solubilidade e calcula um caminho ótimo para atingir o objetivo, que é um padrão do puzzle resolvido. Comecei pesquisando algoritmos práticos para fórmulas mais amplas e sofisticadas. A implementação desses conceitos por meio do prolog deve ser uma tarefa desafiadora; entretanto, uma vez que tenha programado um algoritmo de trabalho, iremos imediatamente mover para a otimização para que possamos completar o puzzle com o menor número de etapas possível.

O algoritmo principal implementado seria “mover”, que especifica o movimento do ladrilho. Para ser mais específico, nossas ações serão a sequência de movimentos de um ladrilho em uma direção específica que permite sair do estado inicial e chegar no estado final.

O tabuleiro inicial é retirado do enunciado da tarefa e representado no programa, para testar o programa com outras configurações iniciais, bastará substituir no programa a cláusula.

Desenvolvimento das regras:

- Localize o ladrilho 1 (o ladrilho a ser colocado na primeira posição) e coloque-o em sua posição correta. Encontrar o espaço em branco e mover os ladrilhos para obter o espaço em branco mais próximo do ladrilho 1 para conseguir isso.

- Colocando todas as peças em uma linha em suas posições corretas.
    - No puzzle 15, todos os 4 ladrilhos da primeira linha serão colocados em suas posições seguindo a técnica acima para o ladrilho 2 e obtendo o espaço em branco para a quarta posição e o ladrilho 4 para a terceira posição e o ladrilho 3 sob sua posição real. fazendo isso para a próxima linha também.

- Colocando todos os ladrilhos na primeira coluna em suas posições corretas.
    - No puzzle 15, todas as 4 peças da primeira coluna serão colocadas em suas posições. Uma vez que as duas primeiras peças da coluna já estão

colocadas. Movemos o espaço em branco entre as duas últimas filas e colocamos os dois últimos ladrilhos da coluna em suas respectivas posições, usando a técnica semelhante que usamos para colocar os ladrilhos 3 e 4 acima. O mesmo para as duas últimas peças na próxima coluna.

- Solução para o problema
    - Isso inclui fornecer a saída final para o problema que nos dá o movimento do ladrilho em branco para atingir o estado final, um pattern do puzzle resolvido.

- Otimização do número de etapas envolvidas no movimento de um ladrilho para sua posição correta.
    - Isso envolve encontrar o número mínimo de movimentos necessários para levar um ladrilho de sua posição atual para sua posição original entre muitas soluções diferentes para ele.

- verifique se o puzzle é solucionável.
    - Retornará uma mensagem de “false” se não houver caminho do estado inicial ao estado final.

Test Cases/ Avaliação do sucesso do trabalho:

- Dado que o tabuleiro inicial foi passado como parâmetro da consulta, teste a solução do primeiro ladrilho a ser colocado na primeira posição a partir de sua posição atual.
- Dado que o tabuleiro inicial foi passado como parâmetro da consulta, teste para a solução da primeira linha que também testa a segunda, assim como ambas seguem a mesma técnica.
- Dado que o tabuleiro inicial foi substituído no programa com 2 primeiras linhas (isto é, os primeiros oitos ladrilhos nas suas posições corretas), isto é, os primeiros oito ladrilhos nas suas posições corretas, teste para a solução da colocação dos últimos dois ladrilhos (ladrilho 9 e ladrilho 13) da primeira coluna. Isso também testará a colocação do ladrilho 10 e 14.
- Teste final - Dado que o tabuleiro inicial foi dado no programa, teste para a solução do problema que retornará o movimento do ladrilho em branco para atingir o objetivo.
- Teste de solvabilidade - dado um estado inicial insolúvel, o programa deve retornar uma mensagem de “falso”.

Progresso do projeto:
O código final começa calculando o caminho mais curto possível para o estado final em relação ao espaço em branco, utilizando a heurística de distância de Manhattan. Depois de calcular uma solução ótima, o programa imprime uma representação gráfica(uma matriz do estado) dos estados inicial, final e intermediário, e apresenta sua solução; uma lista de movimentos do espaço em branco. Soluções múltiplas e abaixo do ideal podem ser calculadas dessa maneira.

Nos parágrafos a seguir, apresento uma série de soluções de teste -:

1. **Movimento Fundamental:**

Nosso primeiro teste apresenta a capacidade básica do programa para detectar disparidade entre os estados de início e objetivo considerados como argumentos, bem como a funcionalidade de suas heurísticas de movimento.

```
Start
estado:'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'/'10'/'11'/'12'/'13'/'14'/0/'1
5'
Final
estado:'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'/'10'/'11'/'12'/'13'/'14'/’15’
/0
```

Notas: Esta é uma solução de 1 movimento que move o ladrilho 15 um espaço para seu destino final. Cada letra corresponde à sua ordem lexicográfica, com exceção de (15).

```
[1] 12 ?- solucaopuzzle(X).
1   2   3   4
5   6   7   8
9   10  11  12
13  14  0  15

1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  0

[direita]
X = [direita] ;
RESULTADO: Sucesso
```

2. **Otimização e interação dos ladrilhos:**

```
Start
estado:'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'/'10'/'15'/'12'/'13'/'14'/'11'
/0

Final
estado:'1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'/'10'/'11'/'12'/'13'/'14'/'15'/0
```

Notas: Esta é uma solução de 4 movimentos destinada a testar não apenas a capacidade do programa de detectar uma solução envolvendo vários estados de objetivo, mas para determinar sua capacidade de detectar a solução mais ideal possível. O programa deve passar por vários estados de falha antes de chegar a uma solução e deve recursar apropriadamente.

```
[1] 5 ?-
solucaopuzzle('1'/'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'/'10'/'11'/'15'/'13'/'1
```

```
4'/'12'/0, X).
1   2   3   4
5   6   7   8
9   10  11  15
13  14  12  0

1   2   3   4
5   6   7   8
9   10  11  0
13  14  12  15

1   2   3   4
5   6   7   8
9   10  0  11
13  14  12  15

1   2   3   4
5   6   7   8
9   10  11  12
13  14  0  15

1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  0
```

3. **O puzzle Randomizado e Complexo**

```
Start
:'1'/'2'/'3'/'4'/'5'/'6'/0/'8'/'9'/'10'/'11'/'12'/'13'/'14'/'15'/'7'
Final:'1'/'2'/'3'/'4'/'5'/'6'/’7’/'8'/'9'/'10'/'11'/'12'/'13'/'14'/'15'/
0
```

Notas: Este é um puzzle totalmente aleatório e complexo mantido em uma solução de 11 etapas, destinado a testar a habilidade de resolução do puzzle em uma situação prática.

```
[1] 5 ?-
solucaopuzzle('1'/'2'/'3'/'4'/'5'/'6'/0/'8'/'9'/'10'/'11'/'12'/'13'/'14'
/'15'/'7', X).
1   2   3   4
5   6   0   8
9   10  11  12
13  14  15  7
```

```
1   2   3   4
5   6   8   0
9   10  11  12
13  14  15  7

1   2   3   4
5   6   8   12
9   10  11  0
13  14  15  7

1   2   3   4
5   6   8   12
9   10  11  7
13  14  15  0

1   2   3   4
5   6   8   12
9   10  11  7
13  14  0  15

1   2   3   4
5   6   8   12
9   10  0  7
13  14  11  15

1   2   3   4
5   6   8   12
9   10  7  0
13  14  11  15

1   2   3   4
5   6   8   0
9   10  7  12
13  14  11  15

1   2   3   4
5   6   0   8
9   10  7  12
13  14  11  15

1   2   3   4
5   6   7   8
9   10  0  12
13  14  11  15
```

```
1   2   3   4
5   6   7   8
9   10  11  12
13  14  0  15

1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  0

[direita,baixo,baixo,esqueda,cima,direita,cima,esqueda,baixo,baixo,direi
ta]
X = [direita, baixo, baixo, esqueda, cima, direita, cima, esqueda,
baixo|...]
```