Inteligência Artificial

Solucionador do jogo Bloxorz

Antonio Carrilho Neto - Aluno UATI
Eduardo Souza Rocha - 11218692
Fábio Verardino de Oliveira - 12674547
Olavo Morais Borges Pereira - 11297792

Profª: Solange Oliveira Rezende

Sumário

1. Introdução

Uma das aplicações mais comuns e conhecidas dos algoritmos de busca é a construção de *solvers* para jogos em que as jogadas podem ser vistas como caminhos e o estado do jogo, como posições.

A fim de exemplificar essa aplicação, escolhemos implementar um *solver* para o jogo de quebra-cabeça *Bloxorz* (https://bloxorz.io/), que consiste em manipular um paralelepípedo, tombando-o para ficar em pé ou deitado, com o objetivo de levá-lo para um um ponto demarcado.

Assim, implementamos mais de uma estratégia de busca e heurísticas (Busca em profundidade, Busca em largura, Busca A*), com o intuito de compará-las e analisar os prós e contras de cada uma.

2. Modelagem

A fase é representada por uma matriz e seus possíveis valores são:

- 0: Vazio
- 1: Chão
- 2: Objetivo/Buraco
- 3: Chão Laranja (Não pode ficar com o bloco em pé nessa coordenada, apenas com ele deitado)

Para modelar o espaço do problema, optamos por usar um grafo onde cada nó é uma posição possível do bloco (suas coordenadas e orientação).

Optamos por armazenar as coordenadas do bloco dessa forma, pois ela diminui o espaço usado por cada nó do grafo em comparação com o armazenamento de toda a matriz em cada nó. Nesse grafo, todas as arestas são, por natureza, não direcionadas, pois sempre que o jogador faz um movimento, ele pode desfazê-lo. No entanto, por questões de implementação, optamos por torná-las direcionadas para armazenar em cada aresta a direção na qual o bloco deve se mover para ir de um nó para outro.

É importante destacar que usamos a biblioteca NetworkX para implementar o grafo. A biblioteca é usada apenas para facilitar a implementação e exibição do grafo. Toda a parte de busca foi implementada pela equipe.

3. Busca não Informada

Optamos por implementar duas buscas não informadas: Busca em Profundidade e Busca em Largura.

3.1. Busca em Largura

Optamos por implementar a busca em largura pois, dada a natureza do problema, ela sempre encontra o caminho com menor custo/menor comprimento: a busca em largura tenta todos os caminhos com comprimento 1, depois todos os caminhos com comprimento 2, depois com comprimento 3, e assim por diante. Acreditamos que essa é uma característica desejável em um *solver* desse tipo de jogo.

3.2. Busca em Profundidade

Optamos também pela implementação da busca em profundidade pois é muito utilizada e pode facilmente ser implementada de forma recursiva. Além disso, a busca em profundidade se apresenta como uma ferramenta de comparação com a busca em largura, outra abordagem de busca cega citada.

4. Busca Informada

Para busca informada, escolhemos implementar a busca A* com 4 heurísticas:

1. Metade da Distância Manhattan da posição atual até o destino
2. Distância Manhattan da posição atual até o destino
3. Metade da Distância Euclidiana da posição atual até o destino
4. Heurística Trivial (h(n) = 0)

É necessário dividir a distância de Manhattan e a Distância Euclidiana por 2 pois o bloco pode andar duas casas em um único movimento. Sendo assim, é necessário dividir por 2 para que a heurística seja admissível. A heurística 2 não faz a divisão pois leva em conta esses movimentos duplos para calcular a distância, o que torna a heurística mais próxima do custo real porém apresenta maior custo computacional.

5. Resultados

Usamos duas métricas para medir as diferentes buscas: tamanho do caminho encontrado e número de nós visitados. Os resultados estão dispostos nas tabelas abaixo:

Fase 1		
Algoritmo de Busca	**Tamanho da Solução**	**Número de Nós Visitados**
Busca em Profundidade	50	67
Busca em Largura	7	56
Busca A* (Manhattan/2)	7	27
Busca A* (Euclidiana/2)	7	30
Busca A* (Trivial)	7	71
Busca A* (Manhattan)	7	28

Fase 3		
Algoritmo de Busca	**Tamanho da Solução**	**Número de Nós Visitados**
Busca em Profundidade	49	100
Busca em Largura	19	97
Busca A* (Manhattan/2)	19	82
Busca A* (Euclidiana/2)	19	85
Busca A* (Trivial)	19	91
Busca A* (Manhattan)	19	68

Fase 4		
Algoritmo de Busca	**Tamanho da Solução**	**Número de Nós Visitados**
Busca em Profundidade	28	63
Busca em Largura	28	70
Busca A* (Manhattan/2)	28	67
Busca A* (Euclidiana/2)	28	68
Busca A* (Trivial)	28	71
Busca A* (Manhattan)	28	66

Fase 6		
Algoritmo de Busca	**Tamanho da Solução**	**Número de Nós Visitados**
Busca em Profundidade	46	103
Busca em Largura	35	108
Busca A* (Manhattan/2)	35	105
Busca A* (Euclidiana/2)	35	106
Busca A* (Trivial)	35	111
Busca A* (Manhattan)	35	107

Fase 13		
Algoritmo de Busca	**Tamanho da Solução**	**Número de Nós Visitados**
Busca em Profundidade	79	124
Busca em Largura	46	139
Busca A* (Manhattan/2)	46	127
Busca A* (Euclidiana/2)	46	128
Busca A* (Trivial)	46	134
Busca A* (Manhattan)	46	126

6. Conclusão

Analisando os resultados, é possível notar que não existe uma busca perfeita ou uma busca que sempre visite menos nós. As buscas com heurísticas ajudaram, mas a depender do mapa da fase, as buscas não informadas encontram soluções visitando aproximadamente o mesmo número de nós. É importante destacar que em muitas fases é necessário voltar para “manobrar” o bloco, isso é, se afastar do objetivo para colocar o bloco em uma posição que será necessária no futuro. Por essa razão, concluímos que as heurísticas tradicionais não garantem ser as melhores heurísticas para resolver o jogo. A melhor busca depende da fase a ser resolvida.