Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT-213
Aluno: Danilo de Farias Matos

Relatório do Laboratório 2 - Busca Informada

1. Breve Explicação em Alto Nível da Implementação

1.1. Algoritmo Dijkstra

Iniciei a implementação do *Djikstra* com o reset da grid e com a aquisição das coordenadas do start_node e goal_node. Além disso iniciei o custo do nó inicial com valor zerado. Após isso montei a fila de prioridades usando heapq que armazena os custos da função g e o nó em sí que serão usados para construir o caminho e atualizar os valores dos custos de cada nó.
Depois foi a vez de criar o loop que retira da fila de prioridades aquele nó com menor custo primeiro e depois adquire todos os nós vizinhos e adiciona na mesma fila para futuras iterações, marcando o nó com a flag “Closed” para que ele não seja visitado posteriormente se ele fizer parte de um caminho mais custoso.

Para a exploração dos sucessores ou vizinhos do 8 conectado foi feito mais um loop (dessa vez o for) para que o custo de cada nó fosse atualizado e inserido na fila de prioridades com a identificação de seu nó pai para que seja possível construir o caminho no método “construct_path” que é chamado apenas quando o nó atual é igual ao nó objetivo.

1.2. Algoritmo *Greedy Search*

Iniciei a implementação do *Greedy Search* com o reset da grid e com a aquisição das coordenadas do start_node e goal_node. Além disso iniciei o custo do nó inicial com valor zerado. Após isso montei a fila de prioridades usando heapq que armazena as distâncias euclidianas entre o nó objetivo e o nó atual que serão usados para escolher o nó com menor distância para expansão dos seus sucessores.
Depois foi a vez de criar o loop que retira da fila de prioridades aquele nó com menor distância primeiro e depois adquire todos os nós vizinhos e adiciona na mesma fila para futuras iterações, marcando o nó com a flag “Closed” para que ele não seja visitado posteriormente.

Para a exploração dos sucessores ou vizinhos do 8 conectado foi feito mais um loop (dessa vez o for) para que o custo real de cada nó, bem como o seu nó pai, fosse atualizado, mas os valores dos custos da função g não são considerados para identificação dos próximos nós prioritários, apenas sua distância euclidiana até o objetivo.

Por fim, o método “construct_path” que é chamado apenas quando o nó atual é igual ao nó objetivo.

1.3. Algoritmo A*

Iniciei a implementação do A* com o reset da grid e com a aquisição das coordenadas do start_node e goal_node. Além disso iniciei o custo do nó inicial com valor zerado e a função f (que é a soma da função g {distância já percorrida em termos de custos} e h {função heurística - distância euclidiana- entre o nó atual e o objetivo}).

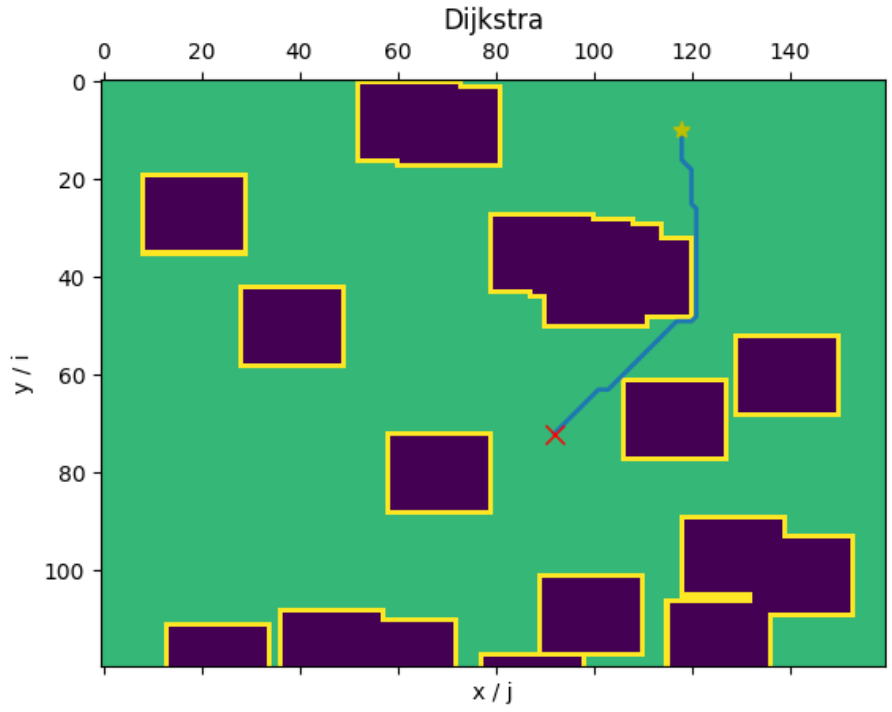
Após isso montei a fila de prioridades usando heapq que armazena os valores da função f que serão usados para escolher o nó com menor custo dessa função para expansão dos seus sucessores.
Depois foi a vez de criar o loop que retira da fila de prioridades aquele nó com menor valor de f primeiro e depois adquire todos os nós vizinhos e adiciona na mesma fila para futuras iterações, marcando o nó com a flag “Closed” para que ele não seja visitado posteriormente.

Para a exploração dos sucessores ou vizinhos do 8 conectado foi feito mais um loop (dessa vez o for) para que o custo real de cada nó, bem como o seu nó pai, fosse atualizado. Além disso, os valores de f de cada nó são utilizados para priorizar os menores valores na hora de fazer a busca.

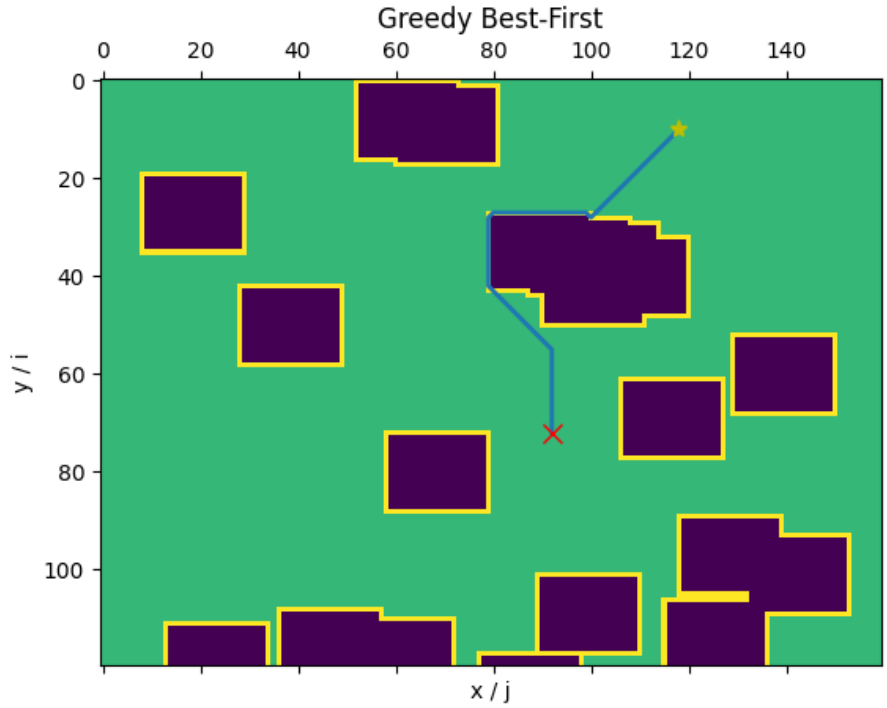
Por fim, o método “construct_path” que é chamado apenas quando o nó atual é igual ao nó objetivo.

2. Figuras Comprovando Funcionamento do Código

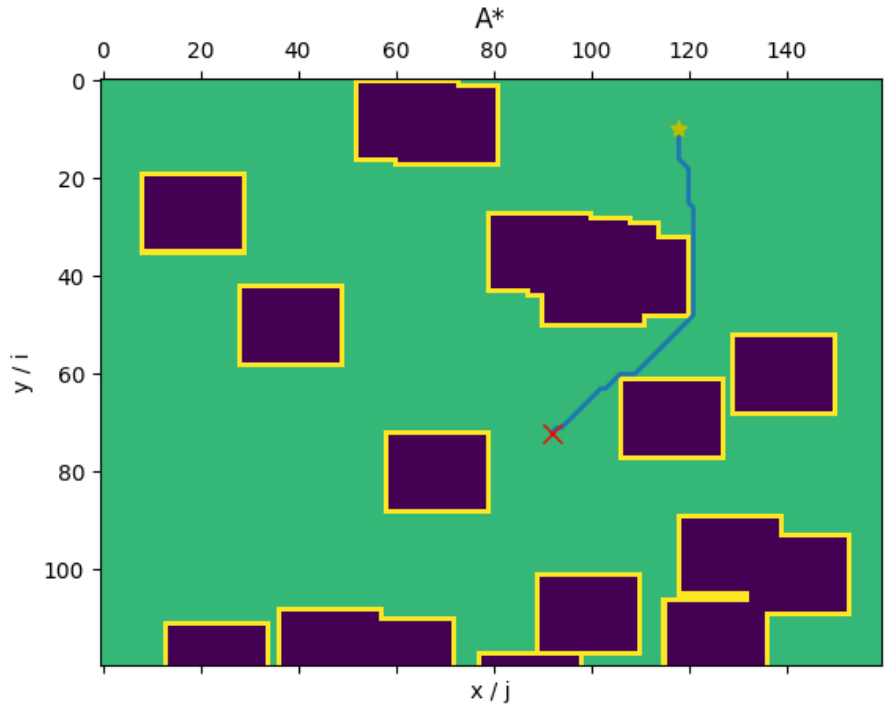
2.1. Algoritmo Dijkstra



2.2. Algoritmo *Greedy Search*



2.3. Algoritmo A*



3. Comparação entre os Algoritmos

Tabela 1 com a comparação do tempo computacional, em segundos, e do custo do caminho entre os algoritmos usando um Monte Carlo com 100 iterações.

Algoritmo	**Tempo computacional (s)**		**Custo do caminho**	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Dijkstra	0.0885	0.0493	79.8292	38.5710
Greedy Search	0.0034	0.0009	103.1175	58.7940
A*	0.0176	0.0152	79.8292	38.5710

Tabela 1: tabela de comparação entre os algoritmos de planejamento de caminho.

Valores obtidos num computador com AMD Ryzen 5 4600H e 8,00 GB de RAM