1 Introdução

Para implementar os algorítmos, primeiramente, impletou-se um método na classe *PathPlanner*, que retorna uma lista de *Nodes* sucessores de um outro *Node*.

```
def get_sucessors(self, node):
    tuples_list = self.node_grid.get_successors(node.i, node.j)
    successor_list = []
    for tuple in tuples_list:
        successor_list.append(self.node_grid.get_node(tuple[0], tuple[1]))
    return successor_list
```

2 Algoritmo de Dijsktra

O algorítmo de Dijsktra foi implementado com base nos slides, chamando o método apresentado acima a cada iteração de loop *while*. Os resultados do Monte Carlo estão nas tabelas 1 e 2, e a figura 1 mostram os resultados obtidos. Abaixo, deixo um pseudo-código do algorítmo implementado:

```
resetar nodes
pq=[]
start_node = achar node na posição inicial
start_node.custo=0
inserir em pq (start_node.custo, start.node)

enquanto pq não for vazio:
    f,node=retirar o mínimo de pq
    fechar o node
    successor_list=self.get_sucessors(node)

    se node for igual a goal_position:
        retorne construir node, node.custo

    para cada sucessor na successor_list
        se o sucessor não estiver fechado
            if successor.custo>node.g+successor.distancia(node.i,node.j):
                successor.custo = node.custo + successor.distancia(node.i, node.j)
                successor.pai=node
                colocar (successor.custo, successor) em pq
```

Média (s)	**Desvio padrão** (s)
0.16184650182724	0.14711992634104026

Tabela 1: Custo computacional do método Dijsktra

Média	**Desvio padrão**
78.90967455478862	38.13194063776156

Tabela 2: Custo do caminho no método Dijsktra

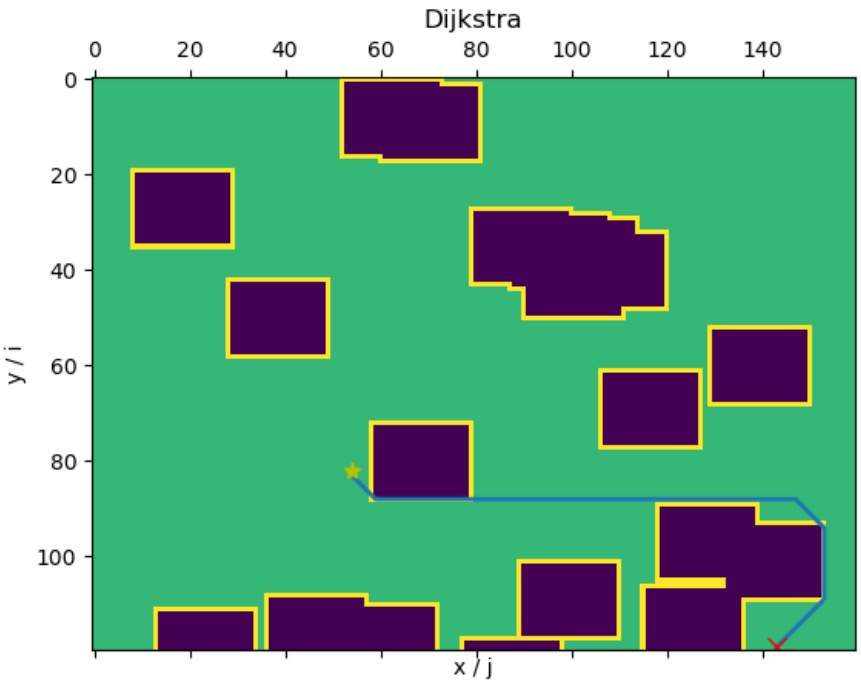

Figura 1: Representação do caminho ótimo achado pelo método Djisktra

3 Greedy Search

O algorítmo Greedy Search foi implementado com base nos slides, chamando o método apresentado acima a cada iteração de loop *while*. Os resultados do Monte Carlo estão nas tabelas 3 e 4, e a figura 2 mostram os resultados obtidos. Abaixo, deixo um pseudo-código do algoritmo implementado:

```
resetar os nodes

pq=[]
start_node=achar node na posição inicial
start_node.custo=distancia (start_position,goal_postition_)

adicionar (start_node.custo,start_node) em pq

enquanto pq não for vazio:

    f,node=retirar o mínimo de pq
    fechar o node
    successor_list=self.get_sucessors(node)

    para cada successor na successor_list:

        se o successor não estiver fechado:
            feche o sucessor
            successor.pai = node
            se o successor.position for igual ao goal_position:
                retorne o caminho, start_node.custo
            successor.custo=distancia do successor ao goal
            adicionar (successor.custo,successor) em pq
```

4 A*

O algorítmo A* foi implementado com base nos slides, chamando o método apresentado acima a cada iteração de loop *while*. Os resultados do Monte Carlo estão nas tabelas 5 e 6, e a figura 3 mostram os resultados obtidos.

Média (s)	Desvio padrão (s)
0.01564026117324829	0.0063823529196777445

Tabela 3: Custo computacional do método Greedy

Média	Desvio padrão
72.59205760235245	34.65664688132339

Tabela 4: Custo do caminho no método Greedy

Abaixo, deixo um pseudo-código do algoritmo implementado:

```
resetar todos os nodes
pq=[]
start_node=achar node na posição inicial
start_node.g=0
start_node.f=distancia (start_position,goal_position)

adicionar (start_node.g,start_node) em pq

enquanto pq não for vazia:

    f,node=retirar o mínimo de pq
    fechar o node

    se node.position() for igual a goal_position:
        retorne o caminho, node.g

    successor_list=self.get_sucessors(node)

    para cada successor na successor_list:
        se o successor estiver aberto:
            fechar o successor
            se successor.f>node.g+distância(node.position(),successor.position())
                +distância (successor.position(),goal_position):
                successor.g=node.g+distância(node.position(),successor.position())
                successor.f=successor.g+distância(successor.position(),goal_position)
                successor.pai=node
                adicionar (successor.g,successor) em pq
```

Média (s)	Desvio padrão (s)
0.2810178017616272	0.152489794921259

Tabela 5: Custo computacional do método A*

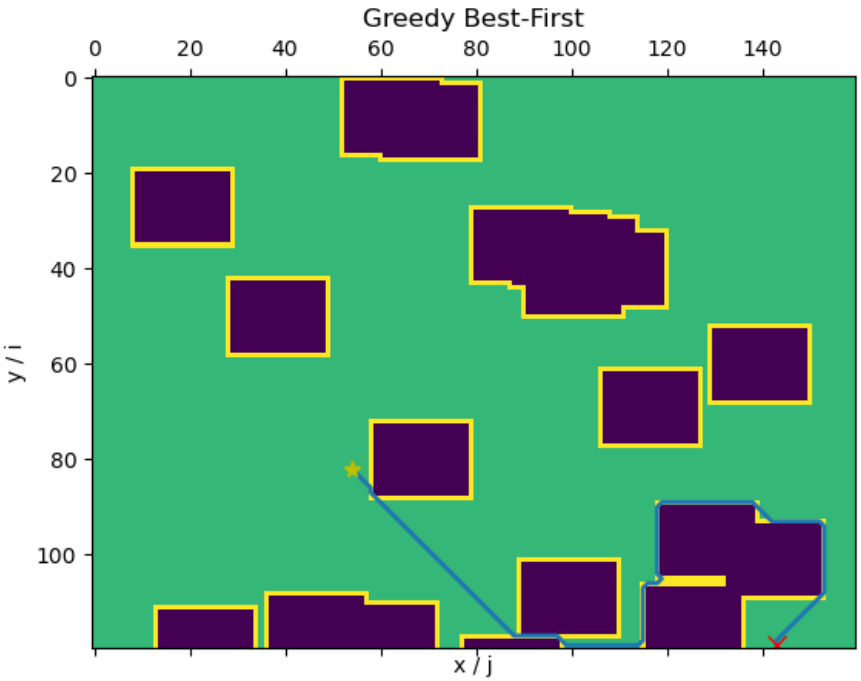

Figura 2: Representação do caminho achado pelo Greedy Search

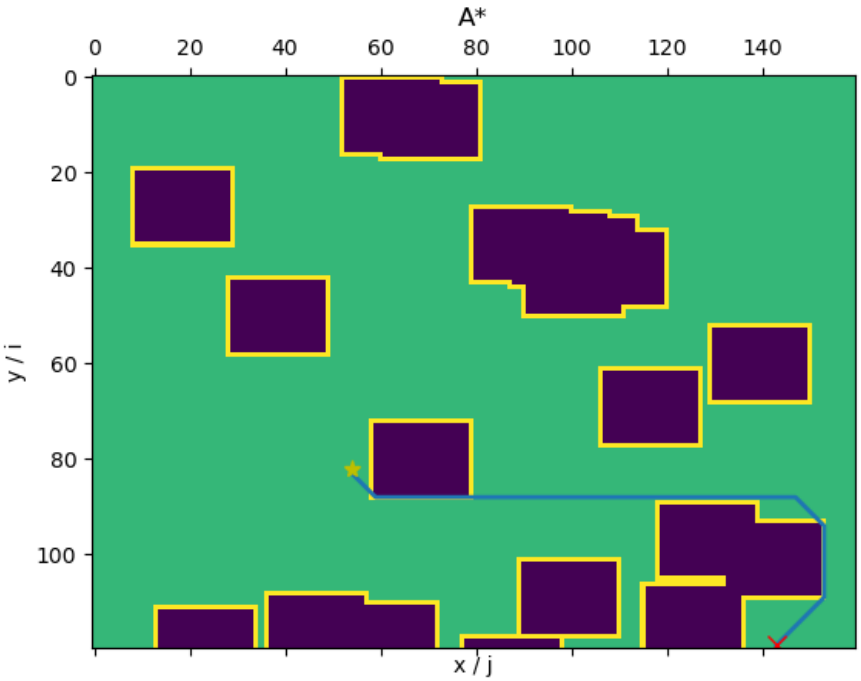

Figura 3: Representação do caminho ótimo achado pelo A*

Média	Desvio padrão
78.90967455478864	38.13194063776158

Tabela 6: Custo do caminho no método A*