A\* Algorithm

import numpy as np  
class Cell:  
 def \_\_init\_\_(self):  
 self.position = (0, 0)  
 self.parent = None  
 self.g = 0  
 self.h = 0  
 self.f = 0  
  
 def \_\_eq\_\_(self, cell):  
 return self.position == cell.position  
  
 def showcell(self):  
 print(self.position)  
  
class Gridworld:  
  
 def \_\_init\_\_(self, world\_size=(5, 5)):  
 self.w = np.zeros(world\_size)  
 self.world\_x\_limit = world\_size[0]  
 self.world\_y\_limit = world\_size[1]  
  
 def show(self):  
 print(self.w)  
  
 def get\_neigbours(self, cell):  
  
 neughbour\_cord = [  
 (-1, -1),  
 (-1, 0),  
 (-1, 1),  
 (0, -1),  
 (0, 1),  
 (1, -1),  
 (1, 0),  
 (1, 1),  
 ]  
 current\_x = cell.position[0]  
 current\_y = cell.position[1]  
 neighbours = []  
 for n in neughbour\_cord:  
 x = current\_x + n[0]  
 y = current\_y + n[1]  
 if 0 <= x < self.world\_x\_limit and 0 <= y < self.world\_y\_limit:  
 c = Cell()  
 c.position = (x, y)  
 c.parent = cell  
 neighbours.append(c)  
 return neighbours  
  
  
def astar(world, start, goal):  
  
  
 \_open = []  
 \_closed = []  
 \_open.append(start)  
  
 while \_open:  
 min\_f = np.argmin([n.f for n in \_open])  
 current = \_open[min\_f]  
 \_closed.append(\_open.pop(min\_f))  
 if current == goal:  
 break  
 for n in world.get\_neigbours(current):  
 for c in \_closed:  
 if c == n:  
 continue  
 n.g = current.g + 1  
 x1, y1 = n.position  
 x2, y2 = goal.position  
 n.h = (y2 - y1) \*\* 2 + (x2 - x1) \*\* 2  
 n.f = n.h + n.g  
  
 for c in \_open:  
 if c == n and c.f < n.f:  
 continue  
 \_open.append(n)  
 path = []  
 while current.parent is not None:  
 path.append(current.position)  
 current = current.parent  
 path.append(current.position)  
 return path[::-1]  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 world = Gridworld()  
 start = Cell()  
 start.position = (0, 0)  
 goal = Cell()  
 goal.position = (4, 4)  
 print(f"path from {start.position} to {goal.position}")  
 s = astar(world, start, goal)  
 for i in s:  
 world.w[i] = 1  
 print(world.w)