1. 题目背景

在地震灾害后，需要一些智能机器人到灾害现场救援。假设每个智能机器人可以从一个地方到达另一个地方（不存在路障等物体的阻碍而到达不了的情形）。在灾害现场，每个智能机器人只能和周围一定范围（是一个受限于通信设备的参数）的其它智能机器人点对点通信，中央控制系统通信也只能和周围的智能机器人通信。每个智能机器人无法通过GPS获得自身的绝对坐标。

要求设计多智能体系统，完成地震现场的救援任务。

2. 项目问题

①每一个智能机器人在出发前和运动中应该具有什么信息从而可以达到指定地点？安装什么传感器才能获得这样的信息？

答：（1）所需信息包括：目标地点信息：每个智能机器人需要在出发前知道其目标救援地点的相对位置或标识信息。自身状态信息：智能机器人需要知道自己的电量、健康状态、移动速度以及可以执行的功能等基本信息，从而适应任务要求，并评估自身完成任务的能力。环境感知信息：智能机器人需要能够感知周围的环境，包括地形、障碍物（该题中没有）、其他智能机器人的位置等，以便进行运动和避障。

（2）所需传感器包括：导航与定位传感器：虽然智能机器人无法通过GPS获得绝对坐标，但可以使用如指南仪、激光雷达、超声波传感器、视觉里程计等技术进行相对定位和导航。这些传感器可以帮助机器人确定自己在环境中的相对位置和方向。环境感知与通信传感器：使用摄像头、红外传感器、声呐、无线电（无线通信技术，如WiFi）等设备来感知周围环境，包括地形、障碍物和其他智能机器人的相对位置；使用点对点通信协议，如Ad Hoc网络协议，来建立和维护与周围机器人的通信。状态监测传感器：电池电量传感器、温度传感器、压力传感器等，用于监测机器人的工作状态和电量水平。

②如果一个智能机器人在救援中，发现需要更多的智能机器人到达该地点救援，它应该怎样和周围智能机器人通信？其它智能机器人应该如何响应才能更好地完成救援任务？

答：（1）怎样通信：利用无线电等设备，在通信范围内联系尽可能多的机器人参与救援。可能需要做到以下几点：1.确定需求：首先，该机器人需要明确所需支援的具体要求，包括需要的机器人数量、所需的专业功能（如医疗救助、重物搬运等）、紧急程度。2.告知方位：需要传输给其他机器人目标地点的具体描述和相对位置。3.广播消息：利用无线通信技术（如WiFi、ZigBee等），在通信范围内广播这条消息；或者采用信号烟之类的实体信号。广播和一定可见范围的信号弹可以确保所有在通信范围内的机器人都能接收到这条信息。4.更新信息：如果情况发生变化，如目标地点的变动或需求数量的减少，发起请求的机器人应及时更新并重新广播消息。

（2）其他智能机器人如何响应：根据目前已知的信息，结合自身工作进程和情况，再判断救援信号的紧急程度，选择是否临时加入救援。

③编程实现你的上述思路，展示智能机器人去灾害现场的过程，以及临时增加救援机器人的过程。

答：详情见下面的“3.多智能体系统实现思路”。

④是否可以通过学习的方法完成这个任务？如果使用学习的方法，你打算怎么做？有什么思路？

答：详情见“5.项目拓展延伸思路”

3. 多智能体系统实现思路

3.1 总体架构

我的设想是在一个1000*1000的坐标图上展示不同种类机器人在地图上的实时运动过程，形成动画，这个可以利用python中的matplotlib.pyplot库和matplotlib.animation库来实现。主要任务是使得搜救机器人在没有GPS、不知道地震点坐标的情况下能够正确地找到地震点的位置坐标并前往，到达后停留在原地进行救援持续一段时间，之后将伤员送回中央控制基地。

既然搜救机器人不具有GPS定位功能，那它就需要与其他智能体通信才能获取它下一个目标点的信息。但如果只有这一种机器人“地毯式”地盲目搜寻，交换信息，显然效率是很低的。所以这就需要少数另一种类的机器人拥有GPS定位功能、知道灾点的坐标，主要任务就是与搜救机器人分享信息而不需要移动至灾点搜救（把它称作巡逻机器人），当它在通信范围内与搜救机器人交换信息时，可以提供给搜救机器人2种坐标：巡逻机器人自身的绝对坐标(x0,y0)、离搜救机器人最近的灾点坐标(x1,y1)。搜救机器人虽然没有GPS，但仍可以通过红外线传感器或雷达之类的传感器获取它与巡逻机器人的相对坐标dx0、dy0,它自己可以计算出灾点的相对坐标方位(dx1,dy1)=(dx0+x1-x0,dy0+y1-y0)，所以能够精确地直接前往目标点。这是第一种寻路方式。

此外，搜救机器人本身也具有视觉传感器（摄像头之类），所以当它在搜寻时离灾点比较近就可以直接发现地震点，再根据视觉传感器传回的参数进行计算，直接得到相对坐标，所以能够直接前往而不需要其他机器人的通信。这是第二种寻路方式。

尽管搜救机器人主要功能是搜救而不是发送信号，但在紧急情况下也非常有必要和同类的搜救机器人进行交互，这一点对应题目要求的“一个智能机器人在救援中，发现需要更多的智能机器人到达该地点救援”。当它到达救援地点发现灾害非常严重需要增援时，会发射可见范围一定的信号弹，并在相同区域内广播，告示其他搜救机器人这里是严重灾点，需要增援。这是其他搜救机器人找到目标点的第三种方式。

最后一种方式是通过中央控制系统基地的通信获取灾区坐标，中央控制基地是在地图上固定的（位于原点），拥有GPS定位以及所有灾区的信息，并且可以维修机器人和补充物资。所有进入中央控制基地通信范围的机器人，可以获取最远的地震点相对坐标并前往。

3.2 代码模块1：地图及地震点分布

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# 定义地图大小
map_size = 1000

# 定义灾点类
class DisasterPoint:
    def __init__(self, coordinates, severity):
        self.coordinates = coordinates  # 灾点坐标
        self.severe = severity  # 灾点严重程度
        self.found = 0  # 被多少机器人救援
        self.exist = True  # 灾点是否存在
# 目标点
# 2-5个普通灾点，1个严重灾点，随机生成
num_targets = np.random.randint(2, 5)
target_points = [DisasterPoint((np.random.randint(0, map_size//5)*5, np.random.randint(0, map_size//5)*5), severity) for severity in [False] * num_targets + [True]]
```

首先需要定义地震点（或者灾点、救援目标点）具有的性质：横纵两个坐标值，是否是“严重灾点”，有多少机器人已经在救援，灾点是否还存在（意思是救援完成，但本次编程中我并没有使用这一点）

然后需要在地图上随机生成2-5个普通的灾点和1个严重灾点，模拟地震发生时的随机情形。需要注意的是，在生成随机数的过程中我使用了5的倍数，因为之后设置机器人的水平竖直移动速度为5，由于地图采用的是整数类型坐标而不是浮点类型，计算和运动路径都不会那么细致。（类似于像素游戏，每次移动方向只有上下左右）

3.3 代码模块2：两种机器人的起点和运动方式

```python
# 搜寻机器人起点：4个从别处（边界）加入增援，3个从中央控制基地出发
search_starts = [(0, np.random.randint(0, map_size//5)*5),
                 (np.random.randint(0, map_size//5)*5, 0),
                 (map_size, np.random.randint(0, map_size//5)*5),
                 (np.random.randint(0, map_size//5)*5, map_size),
                 (0,0),(0,150),(150,0)]

# 机器人运动函数（速度为5）
def move_robot(start, end):
    path = []
    current = start
    while current != end:
        path.append(current)
        next_x = current[0] + 5 * np.sign(end[0] - current[0])
        next_y = current[1] + 5 * np.sign(end[1] - current[1])
        current = (next_x, next_y)
    path.append(end)
    return path

# 巡逻机器人在地图中往返游荡，不具有搜救功能，但却是唯一拥有GPS定位的机器人（所以拥有全部灾区的坐标信息），作用是给搜救机器人汇报灾点
# 巡逻机器人路径
patrol_starts = [(300, 700), (700, 700), (700, 300),(300, 300)]
patrol_path = []
for i in range(len(patrol_starts)):
    next_point = patrol_starts[(i + 1) % len(patrol_starts)]
    patrol_path.extend(move_robot(patrol_starts[i], next_point))
patrol_path.extend(move_robot(patrol_starts[0], patrol_starts[0]))  # 循环四边形路径
```

搜寻机器人共有7个，其中4个是从地图的4条边上某个点随机生成并向地图内45°斜线移动，表达的意思是从“四面八方”来支援的搜救机器人，另外3个是从中央控制基地（原点）附近派出的搜救机器人，其中有一个（0,0）坐标派出的机器人就是从基地出发，在基地的通信范围内，所以它会直接前往最远的灾点进行救援。

机器人运动函数适用于两种机器人，向目标点移动时在水平方向和竖直方向都是5的速度（这个设计使得搜救机器人大部分时候比巡逻机器人快，这是比较符合现实的。

接下来规定巡逻机器人的路径沿着（300,700），（700,700），（700,300），（300,300）的矩形方框周期性巡逻并通信，这样的话它的速度始终为5（因为不走斜线），而搜救机器人常走斜线，速度常为$5\sqrt{2}$.

3.4 代码模块3：规定搜寻机器人的路径

3.4.1 寻路方式1：与巡逻机器人通信

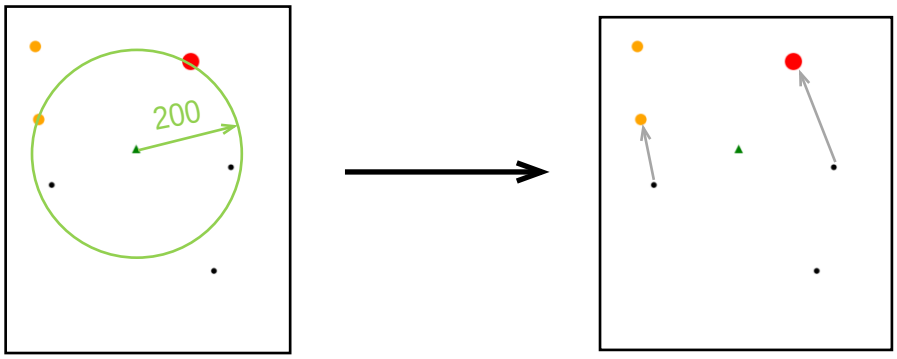
```python
# 更新搜寻机器人路线
def update_search_path(start, targets, patrol_path, search_range=50, communication_range1=200, communication_range2=100, severe_rescue_range=200):
    patrol_index = 0
    path = []
    current = start
    right = np.random.choice([-5, 5])
    up = np.random.choice([-5, 5])

    while True:
        path.append(current)
        patrol_position = patrol_path[patrol_index % len(patrol_path)]

        # 寻路方式1：通过与巡逻机器人通信（通信范围200），获得最近的灾区相对坐标，如果需要救援，直接前往
        if np.sqrt((current[0] - patrol_position[0])**2 + (current[1] - patrol_position[1])**2) < communication_range1:
            # 获取最近的灾点相对坐标，如果小于3个机器人在救援，就前往（普通灾点）；严重灾点直接前往
            # existing_targets = [t for t in targets if t.exist]
            # if  existing_targets:
            closest_target = min(targets, key=lambda t: np.sqrt((current[0] - t.coordinates[0])**2 + (current[1] - t.coordinates[1])**2))
            if (((closest_target.found < 3) and (closest_target.severe == False)) or (closest_target.severe == True)):
             x1, y1 = closest_target.coordinates  # 从DisasterPoint实例中获取坐标
             path.extend(move_robot(current, closest_target.coordinates))
             path.append(target.coordinates) # 到达灾点
             closest_target.found +=1   # 标记灾点被救援+1
             for _ in range(120):  # 模拟停留120个时间步（假设救援时间为120，就需要回基地维修）
              path.append(closest_target.coordinates)
             #target.exist = False # 另一种想法：该灾点救援完成，灾点消失（没有采用）
             patrol_index = 0
             path.extend(move_robot((x1,y1), (0,0))) # 救援结束回中央控制基地
             return path # 下一轮的机器人出发
```

搜寻机器人的路径需要实时更新，因为随时可能接收到其他机器人的信号或者路上可能发现灾点，这一操作在函数update_search_path中实现，需要的参数包括起点、灾点、巡逻机器人位置以及一些受限于通信设备和传感设备的固定参数。（需要注意的是，虽然提供了灾点参数，但并不是搜救机器人随时知道灾点坐标信息）

寻路方式1：在距离巡逻机器人的通信范围200以内，可以获得最近的灾区坐标，并进行判断是否需要前往。判断条件为：严重灾点直接前往；普通灾点若已经在救援的机器人小于3，则加入救援，否则认为救援力量足够，不再前往。到达灾点后将灾点受救援数量+1，并停留120时钟周期，之后返程回到基地维修。回到基地后假设新的搜救机器人再次从相同的起点出发，从而维持地图内始终有7个搜救机器人工作。

寻路方式1实验示例：

（黑色圆点为搜救机器人，绿色三角为巡逻机器人，红色大点为严重灾点，橙色圆点为普通灾点）

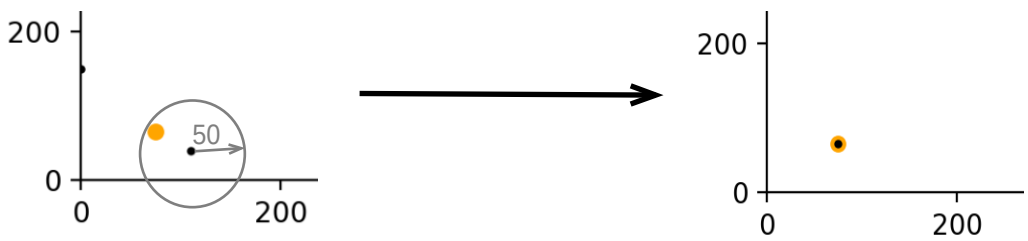


3.4.2 寻路方式2：自行寻找灾点

```
        # 寻路方式2: 自行寻找灾点（能看见的范围50），看见了且正在救援的机器人小于3（普通灾点），就前往；严重灾点直接前往
        for target in targets:
         #if target.exist:
            if np.sqrt((current[0] - target.coordinates[0])**2 + (current[1] - target.coordinates[1])**2) < search_range:
                if ((target.found < 3 and target.severe == False) or (target.severe == True)):
                 x2, y2 = target.coordinates  # 从DisasterPoint实例中获取坐标
                 path.extend(move_robot(current, target.coordinates))
                 path.append(target.coordinates) # 到达灾点
                 target.found +=1   # 标记灾点被救援+1
                 for _ in range(120):  # 模拟停留120个时间步
                  path.append(target.coordinates)
                 #target.exist = False
                 patrol_index = 0
                 path.extend(move_robot((x2,y2), (0,0)))
                 return path

```

寻路方式2：在距离自身的视野范围50以内，可以直接前往，但要进行判断是否需要前往。判断条件为：严重灾点直接前往；普通灾点若已经在救援的机器人小于3，则加入救援，否则认为救援力量足够，不再前往。到达灾点后将灾点受救援数量+1，并停留120时钟周期，之后返程回到基地维修。回到基地后假设新的搜救机器人再次从相同的起点出发。

寻路方式2实验示例：

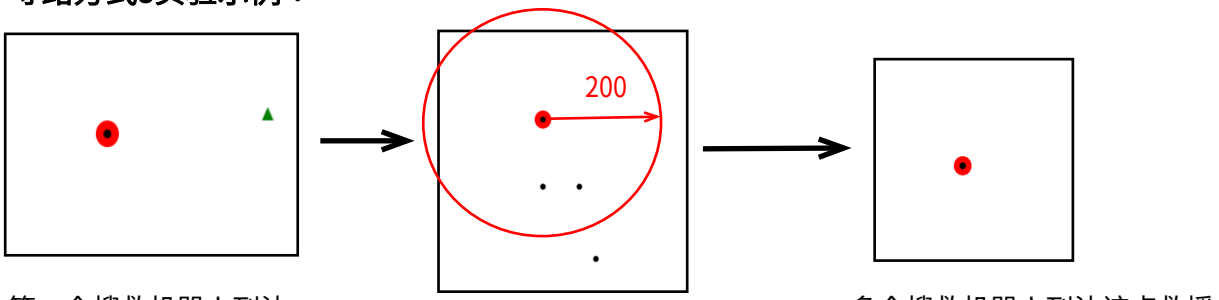


3.4.3 寻路方式3：接收到严重灾点救援信号

```
        # 寻路方式3：发现严重灾点救援信号（范围200），需要更多机器人救援，立即前往
        for target in targets:
         #if target.exist:
           if np.sqrt((current[0] - target.coordinates[0])**2 + (current[1] - target.coordinates[1])**2) < severe_rescue_range:
              if (target.found > 0) and (target.severe == True):
                 x3, y3 = target.coordinates  # 从DisasterPoint实例中获取坐标
                 path.extend(move_robot(current, target.coordinates))
                 path.append(target.coordinates) # 到达灾点
                 target.found +=1   # 标记灾点被救援+1
                 for _ in range(250):  # 模拟停留250个时间步（更久）
                  path.append(target.coordinates)
                 #target.exist = False
                 patrol_index = 0
                 path.extend(move_robot((x3,y3), (0,0)))
                 return path
```

寻路方式3：在距离自身的信号接收范围200以内，收到了其他救援机器人的求助信号，表明这里有一个严重灾点。代码的逻辑条件为：是严重灾点（灾点的severity属性为True)并且已经有救援机器人到达救援（灾点的found属性>0），所以会发出求助信号。到达灾点后将灾点受救援数量+1，并停留250时钟周期（严重灾点的救援更久），之后返程回到基地维修。回到基地后假设新的搜救机器人再次从相同的起点出发。

寻路方式3实验示例：



第一个搜救机器人到达并发出求助信号

在求助信号范围200以内的其他不在执行救援任务的救援搜救机器人前往

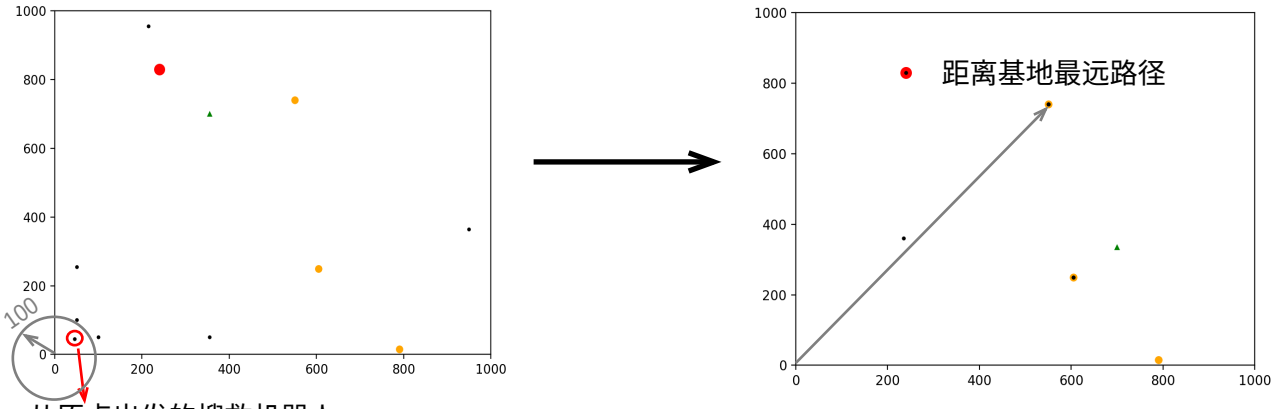
多个搜救机器人到达该点救援（黑点重合）

3.4.4 寻路方式4：与中央控制基地通信

```
        # 寻路方式4：通过与中央控制基地(0,0)通信（通信范围100），获得最远的灾区相对坐标，如果需要救援，直接前往
        if np.sqrt((current[0] - 0)**2 + (current[1] - 0)**2) < communication_range2:
            # 获取最远的灾点相对坐标，如果小于3个机器人在救援，就前往（普通灾点）；严重灾点直接前往
            farthest_target = max(targets, key=lambda t: np.sqrt((current[0] - t.coordinates[0])**2 + (current[1] - t.coordinates[1])**2))
            if (((farthest_target.found < 3) and (farthest_target.severe == False)) or (farthest_target.severe == True)):
             x4, y4 = farthest_target.coordinates  # 从DisasterPoint实例中获取坐标
             path.extend(move_robot(current, farthest_target.coordinates))
             path.append(target.coordinates) # 到达灾点
             farthest_target.found +=1   # 标记灾点被救援+1
             for _ in range(120):  # 模拟停留120个时间步（假设救援时间为120，就需要回基地维修）
              path.append(farthest_target.coordinates)
             #target.exist = False # 另一种想法：该灾点救援完成，灾点消失（没有采用）
             patrol_index = 0
             path.extend(move_robot((x4,y4), (0,0))) # 救援结束回中央控制基地
             return path # 下一轮的机器人出发
```

寻路方式4：在距离中央控制基地100范围以内，可以接收到基地的信号和指令。比较经典的就是从基地出发的机器人，它将直接收到基地的信息。基地将最远的灾点信息告知机器人，并使其前往。到达灾点后将灾点受救援数量+1，并停留120时钟周期，之后返程回到基地维修。回到基地后假设新的搜救机器人再次从相同的起点出发。

寻路方式4实验示例：



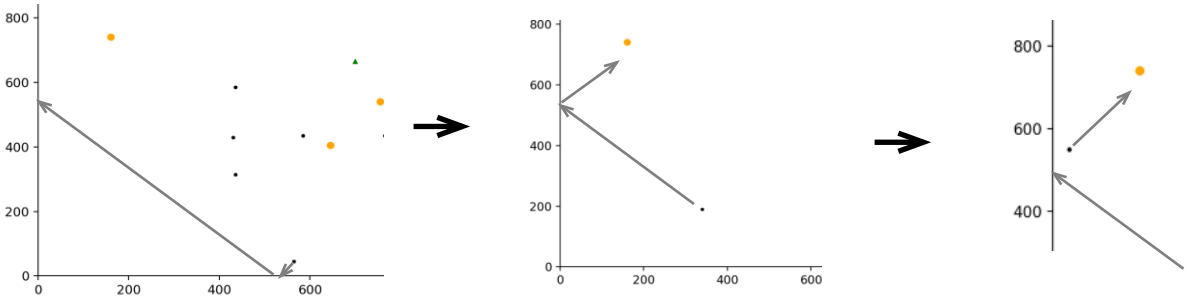
从原点出发的搜救机器人

3.4.5 默认搜寻方式

```
126         # 默认出发方式：弹性斜线往返，等待通信信号或救援信号
127         next_position = (current[0] + right, current[1] + up)
128         if next_position[1] > map_size:
129             up = -5
130             next_position = (current[0] + right, current[1] + up)
131         if next_position[0] > map_size:
132             right = -5
133             next_position = (current[0] + right, current[1] + up)
134 
135         if next_position[1] <= 0:
136             up = 5
137             next_position = (current[0] + right, current[1] + up)
138         if next_position[0] <= 0:
139             right = 5
140             next_position = (current[0] + right, current[1] + up)
141 
142         current = next_position
143         patrol_index += 1
```

默认搜寻和出发方式：沿45°斜线运动，如果碰到边界就像弹球一样，碰撞方向速度反向，平行方向速度不变，在地图中反复运动，以此扩大搜素面积。

默认搜寻方式实验示例：



3.5 **代码模块4**：生成实时动画

规定完各类机器人、基地和地图的相应功能与操作、运动、交互信息的逻辑后，还需要将这些功能在动画地图中体现出来，这些除了依靠python自带的动画图形功能库以外，还需要自行规定一些显示逻辑与功能。

```python
# 初始化界面
fig, ax = plt.subplots()
ax.set_xlim(0, map_size)
ax.set_ylim(0, map_size)

# 实时更新动画
def update(num):
    global patrol_index
    ax.clear()
    ax.set_xlim(0, map_size)
    ax.set_ylim(0, map_size)

    # 绘制普通灾点坐标，橙色，大小为5
    for target in target_points:
     if (target.severe == False):
      ax.plot(target.coordinates[0], target.coordinates[1], 'o', color='orange', markersize=5)
    # 绘制严重灾点坐标，红色，大小为8
     if (target.severe == True):
      ax.plot(target.coordinates[0], target.coordinates[1], 'o', color='red', markersize=8)

    # 更新巡逻机器人位置
    patrol_pos = patrol_path[patrol_index % len(patrol_path)]
    ax.plot(patrol_pos[0], patrol_pos[1], '^', color='green', markersize=3)
    patrol_index += 1

    # 更新搜索机器人位置
    for i in range(len(search_starts)):
        start = search_starts[i]
        if len(search_paths) < len(search_starts):
            path_t= update_search_path(start, target_points, patrol_path)
            # print(path_t)
            search_paths.append(path_t)
        path = search_paths[i]
        idx = num % len(path)
        ax.plot(path[idx][0], path[idx][1], 'o', color='black', markersize=2)

# 创建动画
ani = animation.FuncAnimation(fig, update, frames=1000, interval=50)
plt.show()
```

地图的大小为1000*1000，动画以一定的速率不断更新（决定于最后创建动画的函数参数interval，100是刷新率较低的情况，运行较慢，录制的视频中为50）。

普通灾点为橙色的大小为5的圆点，严重灾点则为大小8的红色圆点，搜救机器人为大小2的黑色运动圆点，巡逻机器人则是大小3的绿色三角形。该模块中还需要实现机器人位置的不断更新，巡逻机器人的移动逻辑较为简单，前面已经阐述过，作周期性矩形运动即可，不会接收信号而改变路径；救援（搜索）机器人的路径较为复杂，需要调用update_search_path函数（也就是上面说的5种方式）。

最后给定参数，创建动画。

4. 项目存在的问题及潜在改进

经过动画实验验证，程序能够较好地完成以上功能。但是，这次项目仍然存在很多问题需要重新考虑或者改进。具体有如下几个方面：

① 代码的书写不规范不美观，实现的功能不够完美和稳定，存在很多引发BUG的可能性（实际上在编写和调试的过程中确实存在很多Bug，每编写一个新的模块都要反复修改新写的代码和原先的代码才能实现预期功能，按照设想运行。反复地修改和填补导致最终的代码比较不美观，有些地方的代码执行逻辑我也没有完全搞明白，但是修改和调试后能够运行出预期结果（其实可能有些地方多余了）。

② 最终的动画效果不稳定，大部分运行可以正常显示动画，但有少数偶尔几次会出现“ 卡死 ”的情况，动画界面无法正常加载出来，程序运行强制退出并报错。猜测原因可能与代码的书写习惯有关，很多地方没有封装好，有些地方的运行逻辑没有考虑周全，可能漏了一些情况，还有地方甚至存在数据溢出的可能。

③ 在多智能体系统的运行结果上，存在明显的瑕疵：有些灾点可能很久都无法被找到，有些救援机器人可能一直处于搜寻状态（默认运动路径）却无法发现灾点。原因有以下几点：

1.巡逻机器人的数量和运动轨迹限制了救援机器人能接收的信号，虽然信号覆盖面积较大，但在周期性巡逻运动下有可能出现“ 每次循环路径都与某个救援机器人错开 ”的情况，这一点通过数学推导计算不难得到。我想到的解决方式是适当调整两种机器人的速度，通过数学计算得出一个或几个能够减少上述“ 周期性错开 ”的情形，但还没有尝试。

2.救援机器人的默认运动轨迹太过简单，如果没有接收到信号时只能在地图中斜线往返并分“ 反弹 ”，确实不太容易自己在50视野范围内发现灾点，或进入200范围的巡逻机器人信号范围。我的想法是可以设置更为复杂的函数控制救援机器人的默认运动轨迹，例如不定时地转弯，甚至曲线运动（但在1000*1000的整数坐标图中较难实现和发挥作用，如果改成浮点坐标类型也许会更适合）。

④ 还可以改进代码实现更加人性化、更加现实以及变化更多的情形，例如：

1.可以把在一个灾点救援的机器人工作量叠加计算，达到一定量后救援就完成、灾点消失，在此处的机器人四散继续寻找新的救援点；相应地，地震过程中也可能随时产生新的灾点，需要机器人救援。

2.可以临时增加更多的机器人加入地图中进行搜救，比如基地可以每隔一段时间生产出新机器人，外边可以来更多的增援，而不是一直保持7个；也可以让机器人回基地后经过一段时间维修后再重新由基地出发继续搜救。

5. 项目拓展延伸思路

这个题目可以通过学习的方法来实现。其实上述实现的代码并没有用到人工智能的一些方法（尤其是花费时间使用数据进行学习和训练），只是单纯地规定了这个多智能体系统的运行逻辑而已。如果想通过学习的方法来完善代码，提高这个系统的功能水平，我的思路有以下几个方面：

①强化学习：利用强化学习改善救援机器人的移动路径，使智能机器人学会如何在复杂的环境中进行导航、搜索和救援，从而提高救援机器人发现灾点并救助的能力。通过与环境的交互，机器人能够学习如何根据当前的观测（如传感器数据）来做出决策（如移动方向、执行动作等）。具体实现可以通过设计奖励函数：根据机器人完成任务的情况，设计合理的奖励函数，例如，成功到达灾点找到幸存者并获得信息则给予较高的奖励，未能完成任务或走错方向则给予一定的惩罚。让机器人在模拟环境中进行训练（例如1000*1000的地图中随机分布灾点和严重灾点），通过不断地尝试和试错，学习如何根据当前的观测来做出最优的决策，优化它的移动路径。

②分布式学习：由于这是个多智能体系统，有很多相同功能的机器人在一起学习，所以可以试着让它们交互信息（包括已经学习到的信息、已知的地图信息、搜索策略等），从而更快地提升机器人群体的整体搜救水平。

③使用路径规划算法：例如A*算法、BFS（广度优先算法）、DFS（深度优先搜索）等，这些方法可能在存在障碍物或地图条件、地形因素复杂时更有作用，能够更好地优化搜救机器人的搜索路径。尤其是当不同的灾点搜救价值不同时，可以给不同的目标点位赋予不同的权重，使得机器人可能放弃近处的权重较低的灾点，而前往较远处的权重高的灾点。