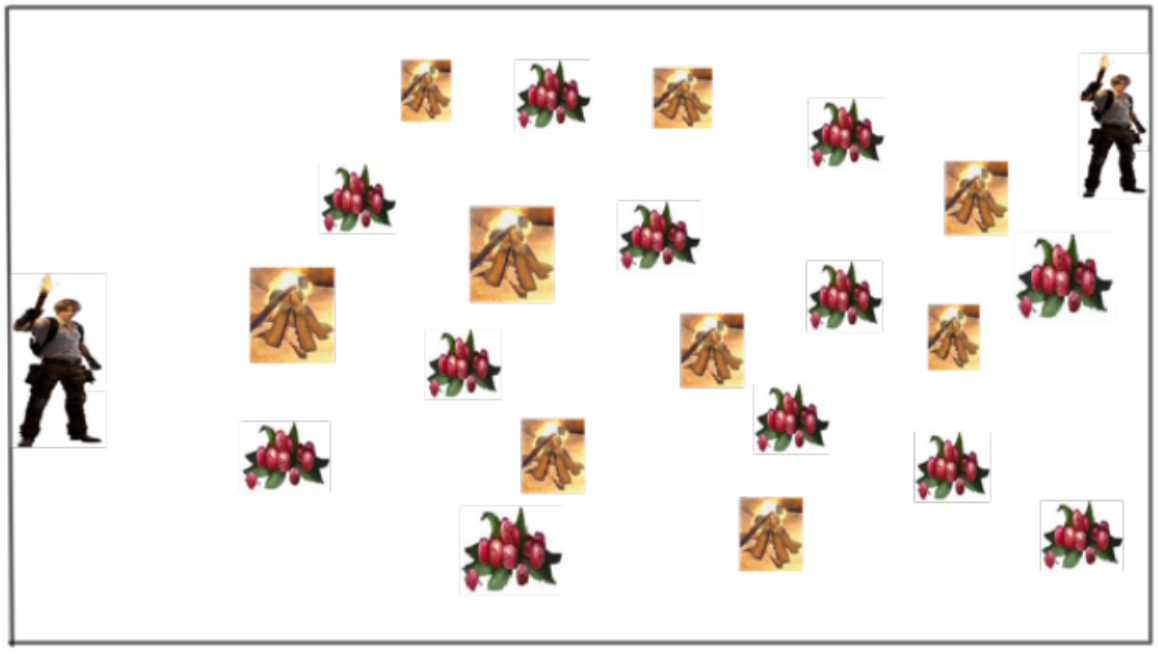
期末大作业

问题描述

《明日之后》是网易开发的一款生存类游戏，讲述这样一个游戏情节：

病毒肆虐各国，人类文明险些毁灭，为了能够在末世中“活下去”，志同道合的伙伴集结起来，一起在病毒蔓延、感染者遍地、资源有限、天气严酷的世界中求生。在游戏中，人物有两类重要的行为，一类是要通过寻找各种食物来维持自身生存，另一类是通过各种御寒措施来降低自身的寒冷程度从而提高生存能力。我们把这两类行为对应到人物的两个特征，前者称为饱食度，刻画人物饥饿的状态，饱食度为负表示处于饥饿状态，饱食度越小，人物生存越难；后者称为舒适度，刻画人物寒冷的状态，舒适度为负表示处于寒冷状态，舒适度越小，游戏人物生存越难。

以该游戏为背景，我们设计这样一个简化的场景。假设游戏人物活动区域为一个空间区域，空间区域中不同位置分布有一些食物和篝火，人物从该区域某一个位置进入，从区域另一个位置出去，如下图所示：



游戏人物在该区域行走的规则如下：

> （1）人物到达食物点吃到食物，其饱食度将提高，提高程度依赖于食物数量。
>
> （2）人物到达篝火位置，其舒适度将提高，提高程度依赖于篝火的大小。
>
> （3）人物在平路（即Z坐标相同）行走100米，饱食度和舒适度均降低5个单位，若走上坡路（Z坐标增加），饱食度和舒适度每走100米均降低6个单位，若走下坡路（Z坐标减少），饱食度和舒适度每走100米均降低4个单位。假设上、下坡已经等效为两点之间直线行走。
>
> （4）当人物到达食物点或篝火点，若饱食度和舒适度中任意一个小于-5，人物将死亡，无法通过食物或篝火提高饱食度或舒适度。
>
> （5）假设人物在开始位置时的饱食度和舒适度均为10。
>
> （6）要求人物到达终点时，饱食度和舒适度均不小于-3。

附件中给出了食物点和篝火点的信息，第一列为点的编号，第2-4列为食物点或篝火点的位置坐标，第一行为起点信息，最后一行为终点信息；第5列为点的类型，1表示该点为食物点，0表示该点为篝火点，第6列为人物位于该点时可通过补充食物或利用篝火提高其饱食度或舒适度的大小，间接代表了该处食物的数量或篝火的大小。

请建立数学模型和算法解决以下问题：问题1：规划该人物从起点到终点的路线（用序号表示），使其经过的食物点和篝火点的次数最少。问题2：在第一问的基础上，进一步考虑人物行走的路径尽可能短，规划其从起点到达终点的路线（用序号表示）。问题3：在篝火点，游戏人物可以制作火把携带，制作火把将使得饱食度降低0.5个单位，但携带的火把能支持人物行走20米而不降低舒适度。请在第一问和第二问的规划路线基础上，进一步考虑人物可制作火把携带的方案，使得人物到达终点后的饱食度和舒适度尽可能高。

问题1：规划该人物从起点到终点的路线（用序号表示），使其经过的食物点和篝火点的次数最少。

问题2：在第一问的基础上，进一步考虑人物行走的路径尽可能短，规划其从起点到达终点的路线（用序号表示）。

问题3：在篝火点，游戏人物可以制作火把携带，制作火把将使得饱食度降低0.5个单位，但携带的火把能支持人物行走20米而不降低舒适度。请在第一问和第二问的规划路线基础上，进一步考虑人物可制作火把携带的方案，使得人物到达终点后的饱食度和舒适度尽可能高。

建模过程

准备工作

第一个工作是要读取数据并在三维空间中绘制出来。

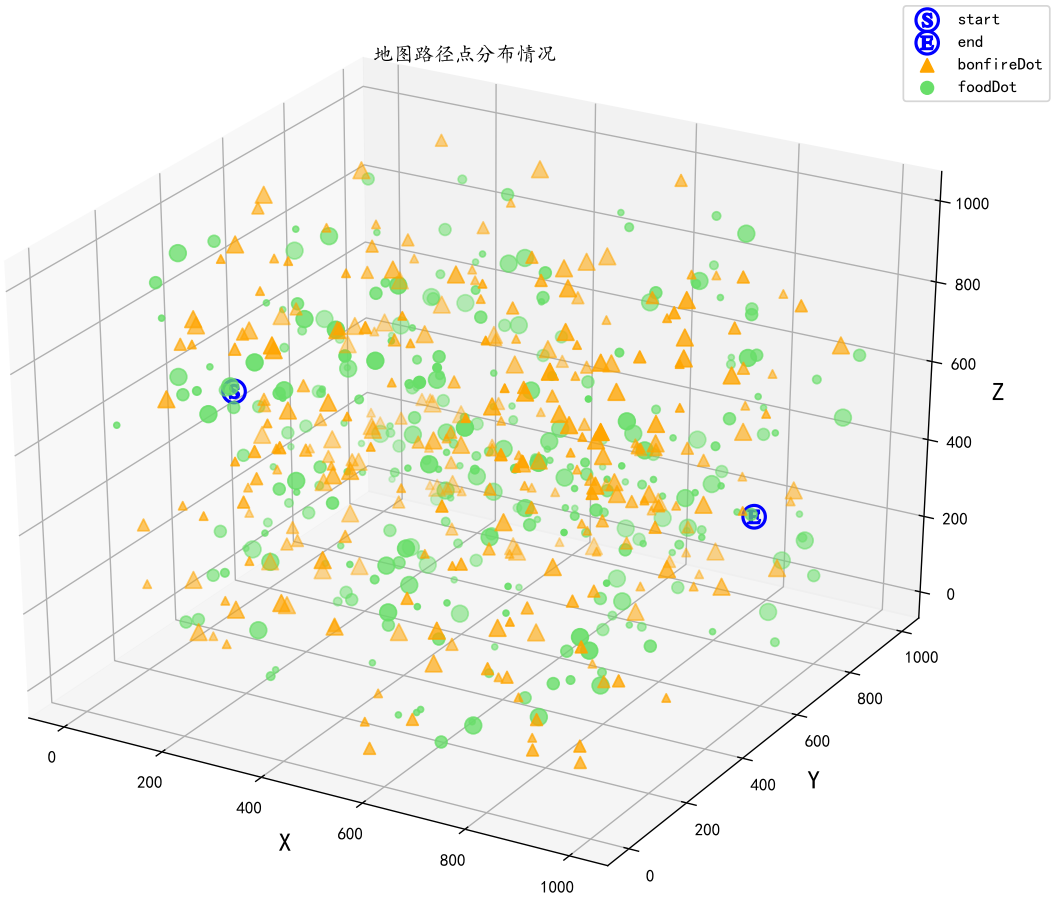
需要用到python的绘图库(matplotlib和Axes3D)，示例代码如下：

（参考python绘制3D散点图）

```
#!usr/bin/env python
# -*- coding: utf-8 -*-
# @author:ygj
# @file: 01-dataVisuliation.py
# @time: 2021/04/23


import numpy as np
```

```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

if __name__ == "__main__":
    # Load data
    dots = np.loadtxt('./data.csv', delimiter=',')
    routeDots = dots[1:-1, :]
    bonfireIndex = [i for i, x in enumerate(routeDots[:, 4].tolist()) if x == 0]
    foodIndex = [i for i, x in enumerate(routeDots[:, 4].tolist()) if x == 1]

    startDot = dots[0, :]  # 起点
    endDot = dots[-1, :]  # 终点
    bonfireDots = routeDots[bonfireIndex, :]  # 篝火点
    foodDots = routeDots[foodIndex, :]  # 食物点

    '''
    可视化
    '''
    #解决中文显示问题
    plt.rcParams['font.sans-serif'] = ['KaiTi']
    plt.rcParams['axes.unicode_minus'] = False

    x1, y1, z1 = bonfireDots[:, 1], bonfireDots[:, 2], bonfireDots[:, 3]
    x2, y2, z2 = foodDots[:, 1], foodDots[:, 2], foodDots[:, 3]

    fig = plt.figure()
    ax = Axes3D(fig)
    # 图例设置
    startMarker = '$\circledS$'
    endMarker = '$\circledE$'

    # 散点绘制
    ax.scatter(startDot[1], startDot[2], startDot[3], c='b', s=200,
marker=startMarker, label='start')
    ax.scatter(endDot[1], endDot[2], endDot[3], c='b', s=200, marker=endMarker,
label='end')
    ax.scatter(x1, y1, z1, c='#FFA500', s=np.exp2(bonfireDots[:, 5]) * 6,
marker='^', label='bonfireDot')
    ax.scatter(x2, y2, z2, c='#68DE69', s=np.exp2(foodDots[:, 5]) * 3,
label='foodDot')

    # 添加坐标轴(顺序是Z, Y, X)
    ax.set_zlabel('Z', fontdict={'size': 15, 'color': 'black'})
    ax.set_ylabel('Y', fontdict={'size': 15, 'color': 'black'})
    ax.set_xlabel('X', fontdict={'size': 15, 'color': 'black'})
    # 添加图例
    ax.legend(loc='best')

    plt.title('地图路径点分布情况')
    plt.show()
    # fig.savefig('scatter.svg', dpi=600, format='eps')
```

可以查看到路径点数据分布如下:



蓝色点代表起点和终点，**黄色点**是篝火点，**绿色点**是食物点，**点大小**代表补给量的多少。

可以观察到：

1. 路径点分布较为均匀
2. 边缘的路径点较为稀疏
3. 没有个别点补给过大的情况

经过可视化分析和小组讨论，我们认为这个题是路径规划问题。通常可以通过状态空间搜索的方法解决。

传统的搜索方法有深度优先搜索、广度优先搜索、A*搜索（启发式搜索）智能的搜索方法有蚁群搜索、粒子群搜索等。这就是我们的基本思路。

问题1

> 规划该人物从起点到终点的路线，使其经过的食物点和篝火点的次数最少。

蚁群算法：

A*算法：

```
*初始化open_set和close_set；
*将起点加入open_set中，并设置优先级为0（优先级最高）；
*如果open_set不为空，则从open_set中选取优先级最高的节点n：
    *如果节点n为终点，则：
        *从终点开始逐步追踪parent节点，一直达到起点；
        *返回找到的结果路径，算法结束；
    *如果节点n不是终点，则：
        *将节点n从open_set中删除，并加入close_set中；
```

```
        *遍历节点n所有的邻近节点：
            *如果邻近节点m在close_set中，则：
                *跳过，选取下一个邻近节点
            *如果邻近节点m不在open_set中，则：
                *设置节点m的parent为节点n
                *计算节点m的优先级
                *将节点m加入open_set中
```

经典的A*算法无法直接套用到本题当中，原因如下：

1. 传统的状态空间搜索算法是先搜索到终点再反推路径的，对路径没有记忆性。但本题中需要先记住当前路径的历史才能够根据玩家当前舒适度和饱食度推算下一步能够到达哪些路径点。
2. 传统A*算法中使用单一的open_set优先级队列保存待搜索节点，每个节点保存parent节点。但本题中前置路径变化会导致结点的后继空间发生变化，无法简单的用parent保存。

针对以上情况，我对A*算法进行了一些改进，以适应本题的要求：

改进的A*算法：

```
*初始化route_set，open_set和approachable_set
*将起点加入route_set中
*更新当前player状态
*更新当前approachable_set，若节点在route_set中，则不加入
*将approachable_set加入到open_set中
*如果open_set不为空且route_set长度小于25(剪枝常数)，则：
    *如果open_set[-1]不为空，则从open_set[-1]中根据启发式选取优先级最高的点n:
        *如果节点为终点，则：
            *将节点n从open_set[-1]中删除(open_set[-1].pop())，并加入route_set中
            *更新当前player状态
            *保存完整路径
            *continue
        *如果节点n不是终点，则：
            *将节点n从open_set[-1]中删除(open_set[-1].pop())，并加入route_set中
            *更新当前player状态
            *更新当前approachable_set，若节点在route_set中，则不加入
            *将approachable_set加入到open_set中
    *如果open_set[-1]为空，则：
        *open_set.pop()
        *route_set.pop()
        *更新当前player状态
```

1. 首先针对路径无记忆性的问题：

   通过增加一个route_set用以保存当前玩家的历史路径，并在每次搜索过程中使用route_set更新玩家舒适度饱食度状态，并根据当前状态计算出一个可到达的后继节点列表approachable_set，并将整个approachable_set加入到open_set中。route_set、open_set和approachable_set均为栈。

2. 针对节点后继问题：
   每次搜索过程中从open_set的最后一个元素中根据启发式函数弹出优先级最高的后继节点，并将其加入到route_set中去，重复更新玩家状态和approachable_set，并将approachable_set加入到open_set中。若open_set的最后一个元素为空，则说明当前路径下无法找到后继节点。这时我们将open_set和route_set[]中最后一个元素弹出，并让玩家退后一个节点，继续搜索。

改进的A*算法代码实现如下：

```
# searchResult用于保存搜索的可行路径结果
    searchResult = []
    # 初始化open_set, route_set和approachable_set
```

```
    route_set, open_set, approachable_set = [[], [], []]
    # 将起点加入route_set中
    route_set.append(dots[0])
    # 更新当前player状态
    player.update(route_set)
    # 更新当前approachable_set，若节点在route_set中，则不加入
    approachable_set = player.getApproachableSet(dots)
    # 将approachable_set加入到open_set中
    open_set.append(approachable_set[:])
    # 如果open_set不为空
    while open_set:
        # 如果open_set[-1]不为空
        if open_set[-1]:
            # 从open_set[-1]中根据*启发式*选取优先级最高的点n:
            i, bestDot = strategy(player, open_set[-1][:])
            # 如果节点为终点
            if bestDot == player.endDot:
                res += 1
                print(res)
                # 将节点n从open_set[-1]中删除(open_set[-1].pop())，并加入route_set中
                route_set.append(open_set[-1].pop(i))
                # 更新当前player状态
                player.update(route_set)
                if res <= epoch:  # epoch次搜索
                    # 进行下一次搜索
                    route_set.pop()
                    player.update(route_set)
                    continue
                else:
                    return searchResult
            # 如果节点n不是终点
            else:
                # 将节点n从open_set[-1]中删除(open_set[-1].pop())，并加入route_set中
                route_set.append(open_set[-1].pop(i))
                # 更新当前player状态
                player.update(route_set)
                # 更新当前approachable_set，若节点在route_set中，则不加入
                approachable_set.clear()
                approachable_set = player.getApproachableSet(dots)
                # 将approachable_set加入到open_set中
                open_set.append(approachable_set[:])
        # 如果open_set[-1]为空
        else:
            open_set.pop()
            route_set.pop()
            player.update(route_set)
```

改进：当搜索路径长度大于50时剪枝（实际搜索过程中发现没有必要）

启发式设计

要使经过的点最少，就是要尽可能**每一步都走消耗最小且饱食度和舒适度最高的点**。其实就是让[目标点的能量值-到目标点消耗的能量]尽可能大。以此作为搜索算法的损失函数。

经过思考实际上应该是使[当前点到终点消耗值-目标点到终点消耗值+目标点的能量值-到目标点消耗的能量]尽可能大且大于0。

启发式函数如下：

$$max(\text{当前点到终点消耗能量} - \text{目标点到终点消耗能量} + \text{目标点的能量} - \text{到目标点消耗的能量})$$

代码实现如下：

```
def leastDotsStrategy(player, approachable_set):
    """
    第一问策略：
    MAX[当前点到终点消耗值 - 目标点到终点消耗值 + 目标点的能量值 - 到目标点消耗的能量]
    """
    currentDot = player.route[-1]
    endDot = player.endDot

    bestDotIndex = None
    bestDot = None
    cost = -float('inf')
    for i, dot in enumerate(approachable_set):
        if dot == player.endDot:
            bestDotIndex, bestDot = i, dot
            break
        else:
            tmp = scCostCal(currentDot, endDot) - scCostCal(dot, endDot) +
dot.supply - scCostCal(currentDot, dot)
            if tmp > cost:
                bestDotIndex, bestDot = i, dot
                cost = tmp
    return bestDotIndex, bestDot
```

问题2

> 在第一问的基础上，进一步考虑人物行走的路径尽可能短。

启发式设计

与问题一相似，其实只要保证让[目标点的能量值-到目标点消耗的能量]尽可能大的同时**尽可能接近终点。**

启发式函数如下：

$$min(\text{目标点到终点的距离})$$

代码实现如下：

```
def leastDistanceStrategy(player, approachable_set):
    """
    第二问策略
    使[目标点到终点距离]尽可能小
    """
    endDot = player.endDot

    bestDotIndex = None
    bestDot = None

    cost = float('inf')
```

```
    for i, dot in enumerate(approachable_set):
        if dot == player.endDot:
            bestDotIndex, bestDot = i, dot
            break
        else:
            tmp = euclidDistance(dot.position, endDot.position)
            if tmp < cost:
                bestDotIndex, bestDot = i, dot
                cost = tmp
    return bestDotIndex, bestDot
```

问题3

> 进一步考虑人物可制作火把携带的方案，使得人物到达终点后的饱食度和舒适度尽可能高。

启发式设计

在篝火点，游戏人物可以制作火把携带，制作火把将使得饱食度降低0.5个单位，但携带的火把能支持人物行走20米而不降低舒适度。

本质上就是在每一个篝火点增加一个可选的项：用0.5个饱食度换取1个舒适度。

这一问的逻辑还没想好，但初步的想法是只有当舒适度低于饱食度时才选择制作火把。

结果分析

为了更好对比不同启发式对算法的影响，我们以搜索到的前500条可行路径为结果进行分析。

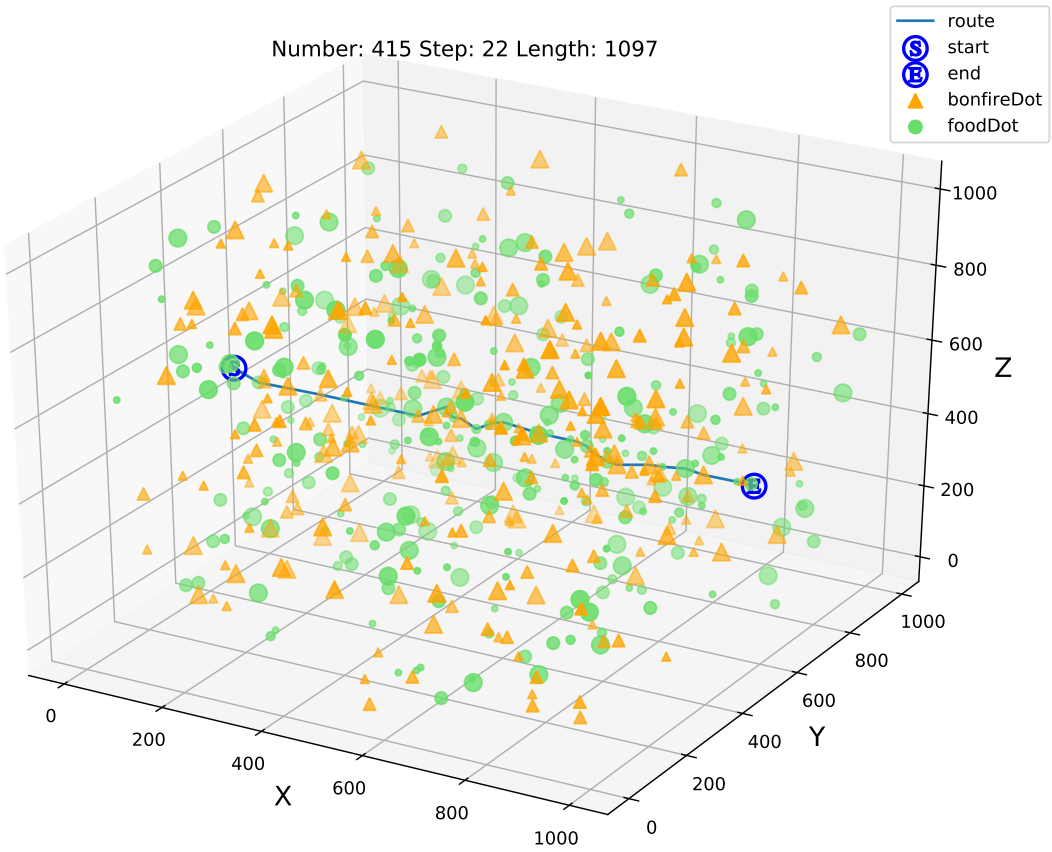
问题1

在第415次搜索中得到经过最少补给次数的路径，为22次，耗时69秒。

路径点序列为：

[0, 25, 91, 200, 232, 230, 243, 254, 270, 296, 307, 338, 387, 402, 411, 434, 452, 479, 501, 518, 537, 569, 587]

到终点时剩余的补给：饱食度-1.86 舒适度-2.86。



具体信息如下:

```
Position: [1000.     555.67   442.022]
Satiety: -1.8662648167212543 Comfort: -2.8662648167212534 Step: 22
time_cost:69.64824748039246
[0, 25, 91, 200, 232, 230, 243, 254, 270, 296, 307, 338, 387, 402, 411, 434, 452,
479, 501, 518, 537, 569, 587, ]
Step0: 0--->0[  0. 500. 500.]--->[  0. 500. 500.]
Step1: 0--->25[  0. 500. 500.]--->[ 49.2701 501.378  474.378 ]
Step2: 25--->91[ 49.2701 501.378  474.378 ]--->[143.808 552.57  442.582]
Step3: 91--->200[143.808 552.57  442.582]--->[341.06  554.258 440.06 ]
Step4: 200--->232[341.06  554.258 440.06 ]--->[397.14  556.215 479.556]
Step5: 232--->230[397.14  556.215 479.556]--->[396.166 557.686 463.285]
Step6: 230--->243[396.166 557.686 463.285]--->[414.847 550.557 458.839]
Step7: 243--->254[414.847 550.557 458.839]--->[432.744 544.693 464.194]
Step8: 254--->270[432.744 544.693 464.194]--->[466.047 540.082 447.639]
Step9: 270--->296[466.047 540.082 447.639]--->[497.857 539.374 473.068]
Step10: 296--->307[497.857 539.374 473.068]--->[518.037 544.067 478.518]
Step11: 307--->338[518.037 544.067 478.518]--->[594.039 532.547 476.9  ]
Step12: 338--->387[594.039 532.547 476.9  ]--->[668.799 533.409 475.24 ]
Step13: 387--->402[668.799 533.409 475.24 ]--->[686.951 534.936 468.021]
Step14: 402--->411[686.951 534.936 468.021]--->[703.063 520.512 467.263]
Step15: 411--->434[703.063 520.512 467.263]--->[739.278 513.132 452.663]
Step16: 434--->452[739.278 513.132 452.663]--->[758.889 512.304 450.352]
Step17: 452--->479[758.889 512.304 450.352]--->[803.157 544.765 446.707]
Step18: 479--->501[803.157 544.765 446.707]--->[827.67  547.777 447.158]
Step19: 501--->518[827.67  547.777 447.158]--->[866.403 557.739 448.524]
Step20: 518--->537[866.403 557.739 448.524]--->[904.524 552.094 445.804]
Step21: 537--->569[904.524 552.094 445.804]--->[965.271 559.438 441.67 ]
Step22: 569--->587[965.271 559.438 441.67 ]--->[1000.     555.67   442.022]
```

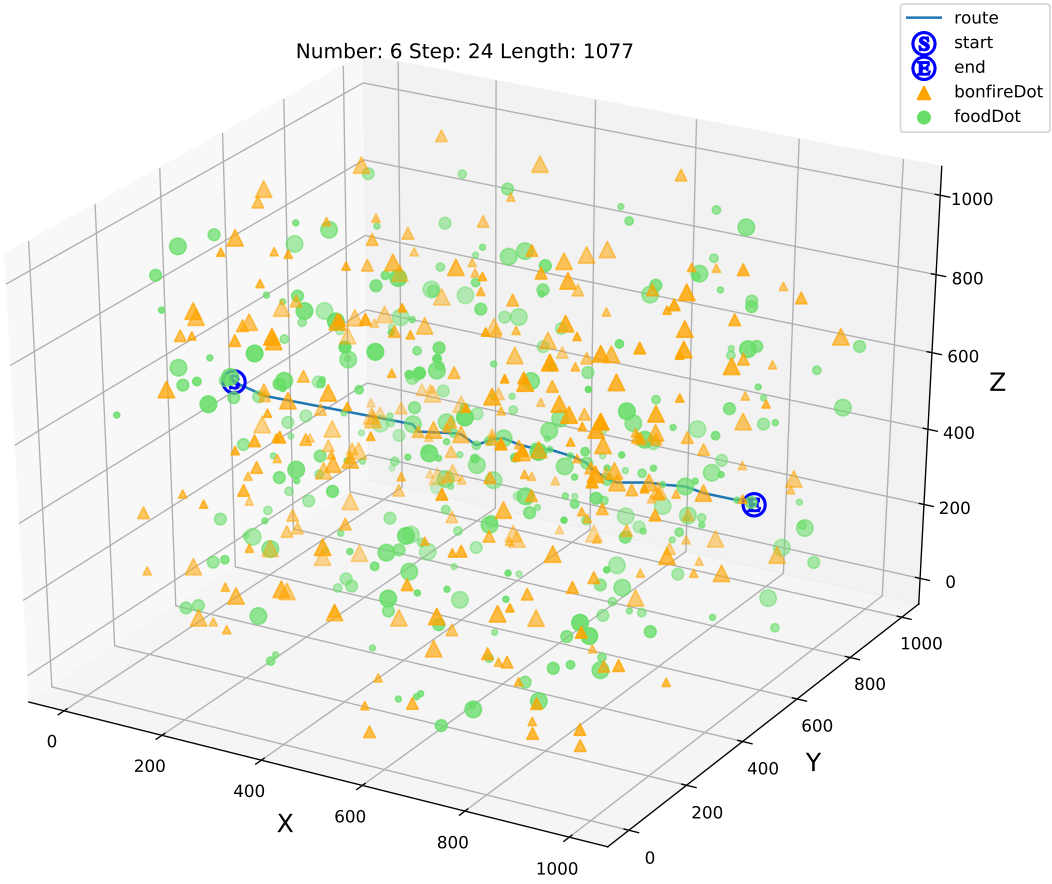
问题2

在第6次搜索中得到经过距离最短的路径，为1077米，耗时5秒。

路径点序列为：

[0, 33, 191, 200, 243, 254, 270, 296, 307, 326, 338, 345, 374, 387, 402, 411, 434, 452, 479, 501, 518, 537, 569, 576, 587]

到终点时剩余的补给：饱食度-2.58 舒适度-1.58。

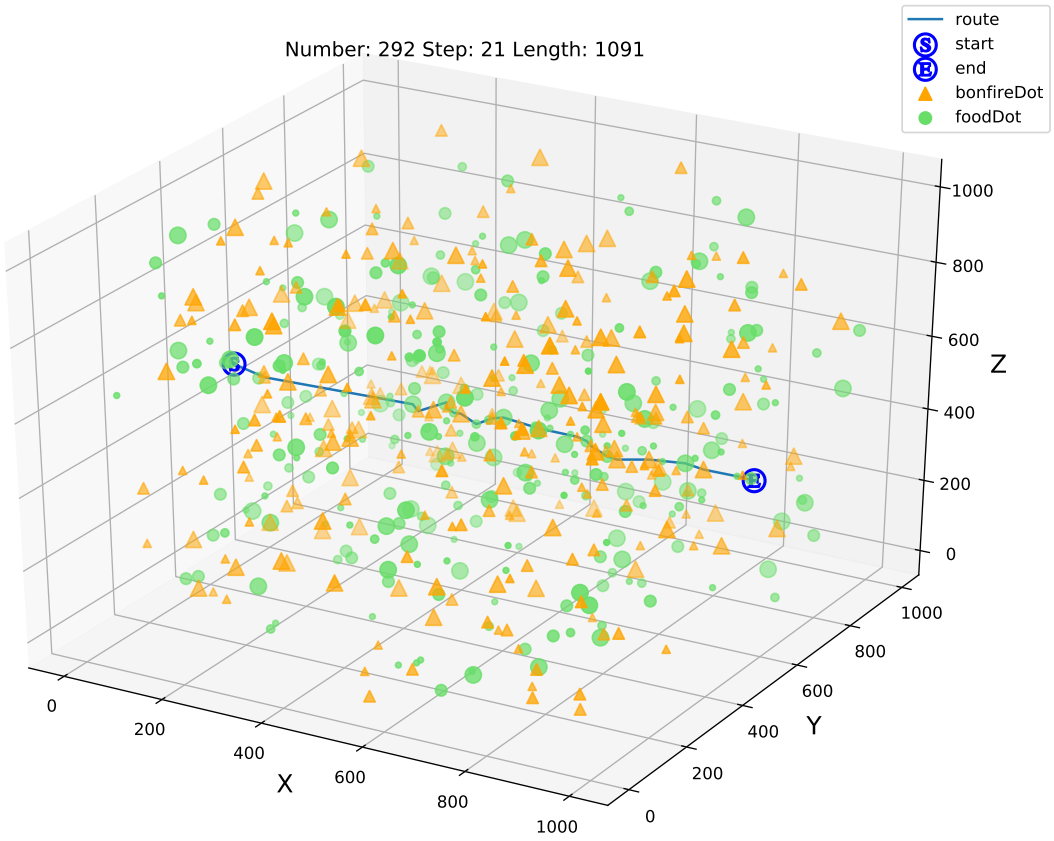


值得注意的是，在第292次搜索中得到了比第一问更好的结果，一条21个点的路径：

路径点序列为：

[0, 33, 191, 200, 232, 230, 243, 254, 270, 296, 307, 338, 387, 402, 411, 434, 452, 479, 518, 537, 569, 587]

到终点时剩余的补给：饱食度-2.63 舒适度-2.63。


Number: 292 Step: 21 Length: 1091
route
start
end
bonfireDot
foodDot
X
Y
Z
0
200
400
600
800
1000