

# 期末大作业

小组成员：

队长：尤冠杰-20053068-智能科学学院

队员1：孙浏鑫-20053070-智能科学学院

队员2：邓慧明-20020022-计算机学院

## 0 目录

### 期末大作业

0 目录

1 问题描述

2 建模过程

2.1 准备工作

2.1.1 数据可视化

2.1.2 游戏框架搭建

2.1.3 问题分析

2.2 问题1

2.2.1 A\*算法：

传统的A\*算法

改进的A\*算法

2.2.2 启发式设计

2.3 问题2

启发式设计

2.4 问题3

启发式设计

3 结果分析

问题1

问题2

问题3

总结

4 心得体会

附录

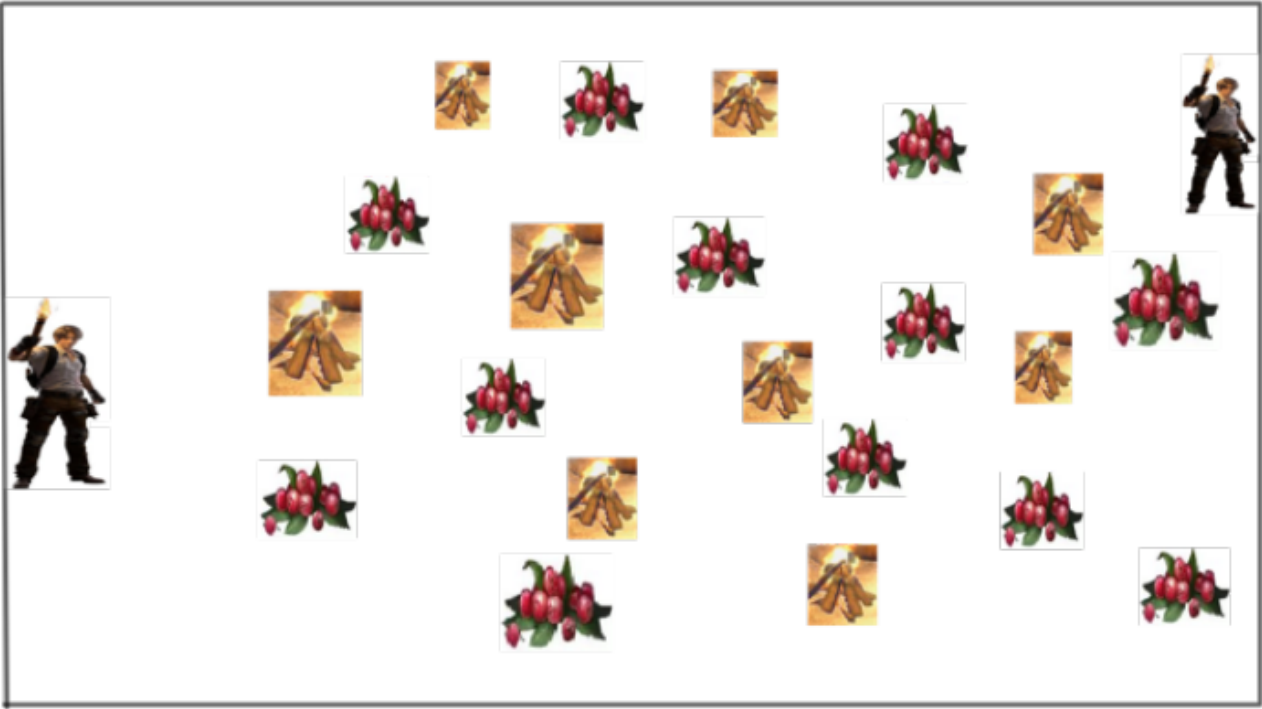
参考文献

## 1 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。

## 2 建模过程

### 2.1 准备工作

#### 2.1.1 数据可视化

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)，实现代码如下<sup>1</sup>：

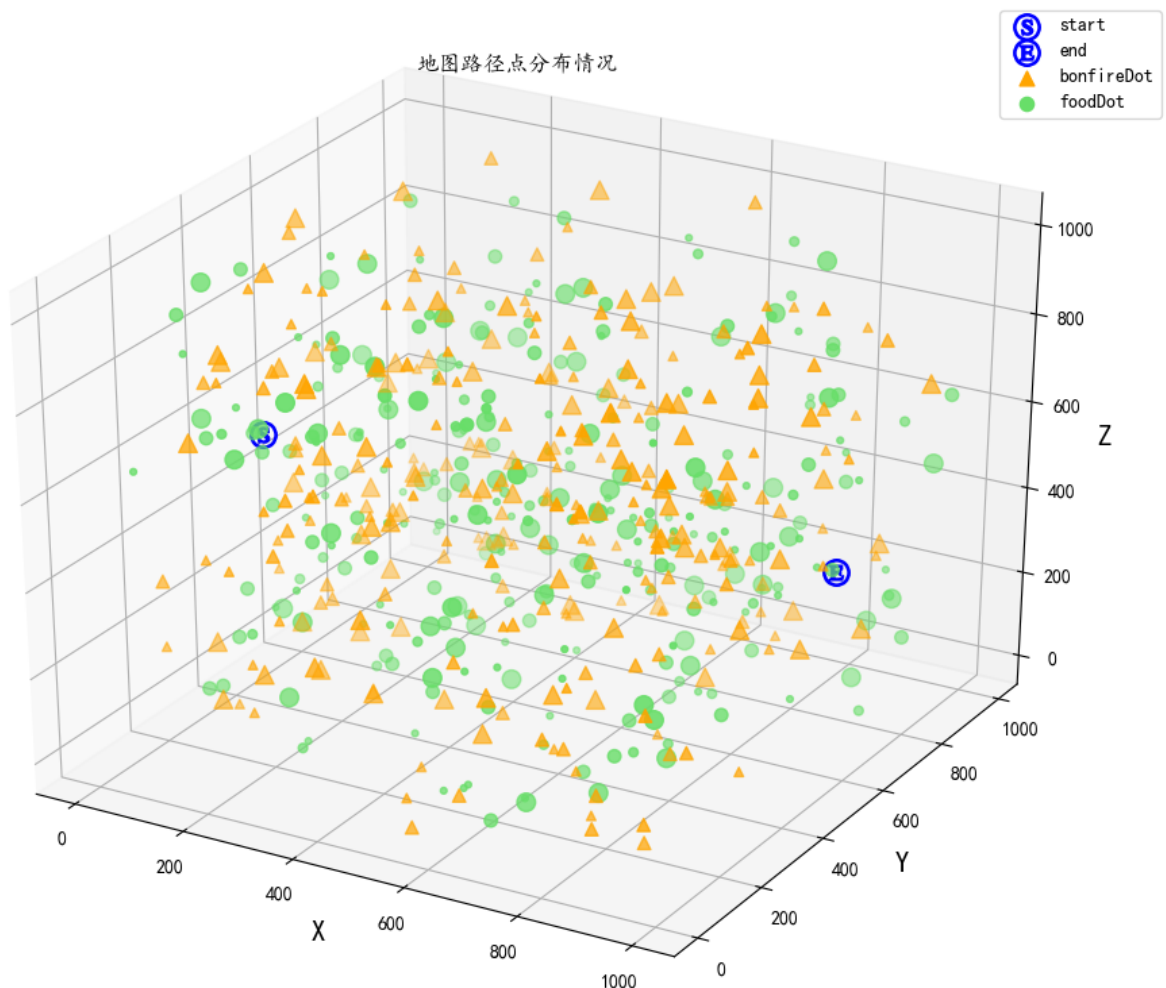
```
1  #!usr/bin/env python
2  # -*- coding: utf-8 -*-
3  # @author:ygj
4  # @file: 01-dataVisulation.py
5  # @time: 2021/04/23
6
7
8  import numpy as np
9  from matplotlib import pyplot as plt
10 from mpl_toolkits.mplot3d import Axes3D
11
12 if __name__ == "__main__":
13     # Load data
14     dots = np.loadtxt('./data.csv', delimiter=',')
15     routeDots = dots[1:-1, :]
16     bonfireIndex = [i for i, x in enumerate(routeDots[:, 4].tolist()) if x == 0]
17     foodIndex = [i for i, x in enumerate(routeDots[:, 4].tolist()) if x == 1]
18
19     startDot = dots[0, :] # 起点
20     endDot = dots[-1, :] # 终点
21     bonfireDots = routeDots[bonfireIndex, :] # 篝火点
22     foodDots = routeDots[foodIndex, :] # 食物点
23
24     '''
25     可视化
26     '''
27     #解决中文显示问题
28     plt.rcParams['font.sans-serif'] = ['KaiTi']
29     plt.rcParams['axes.unicode_minus'] = False
30
31     x1, y1, z1 = bonfireDots[:, 1], bonfireDots[:, 2], bonfireDots[:, 3]
32     x2, y2, z2 = foodDots[:, 1], foodDots[:, 2], foodDots[:, 3]
33
34     fig = plt.figure()
35     ax = Axes3D(fig)
36     # 图例设置
37     startMarker = '$\circledS$'
38     endMarker = '$\circledE$'
39
```

```

40 # 散点绘制
41 ax.scatter(startDot[1], startDot[2], startDot[3], c='b', s=200, marker=startMarker,
label='start')
42 ax.scatter(endDot[1], endDot[2], endDot[3], c='b', s=200, marker=endMarker, label='end')
43 ax.scatter(x1, y1, z1, c='#FFA500', s=np.exp2(bonfireDots[:, 5]) * 6, marker='^',
label='bonfireDot')
44 ax.scatter(x2, y2, z2, c='#68DE69', s=np.exp2(foodDots[:, 5]) * 3, label='foodDot')
45
46 # 添加坐标轴(顺序是Z, Y, X)
47 ax.set_zlabel('Z', fontdict={'size': 15, 'color': 'black'})
48 ax.set_ylabel('Y', fontdict={'size': 15, 'color': 'black'})
49 ax.set_xlabel('X', fontdict={'size': 15, 'color': 'black'})
50 # 添加图例
51 ax.legend(loc='best')
52
53 plt.title('地图路径点分布情况')
54 plt.show()
55 # 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

## 2.1.2 游戏框架搭建

为了更好的实现搜索策略，以及在改进算法过程中便于迭代代码，我们选择将路径点数据抽象为**路径点类**，并构建了一个**玩家类**，用来模拟玩家人物在路径点之间移动的行为。

**路径点类：**

主要包含路径点的序列、位置、补给量和补给类型。

实现代码如下：

```
1 class Dot:
2     def __init__(self, index, position, food, supply):
3         """
4         路径点类
5         :param index: 序号
6         :param position: 位置
7         :param isFood: 是否为食物
8         :param isBonfire: 是否为篝火
9         :param isArrived: 是否到达过
10        :param supply: 补给数
11        :param x, y, z: 坐标
12        """
13        self.index = int(index)
14        self.position = position
15        self.x, self.y, self.z = self.position
16        self.isFood = bool(food)
17        self.isBonfire = not bool(food)
18        self.supply = supply
19
20    def __eq__(self, other):
21        """
22        =运算符重载
23        判断两点是否相同
24        """
25        return (self.index == other.index) and (self.x == other.x) and (self.y == other.y)
26        and (self.z == other.z)
27
28    def info(self):
29        """
30        查看当前路径点信息
31        """
32        if self.isFood:
33            print('Dot', self.index, ' Position: ', self.position, 'Food Supply: ',
34                  self.supply)
35        elif self.isBonfire:
36            print('Dot', self.index, ' Position: ', self.position, 'BonFire Supply: ',
37                  self.supply)
```

**玩家类：**

主要包含玩家起点、终点、初始饱食度、初始舒适度、当前位置和路径。

同时包含了根据路径更新当前状态、移动到目标点、判断目标点是否能够到达、计算到目标点的消耗值、获得当前状态等函数。

实现代码如下：

```
1 class Player:
2     def __init__(self, startDot, endDot, satiety, comfort):
```

```

3         """
4         玩家类
5         :param startDot:起点
6         :param endDot:终点
7         :param satiety:饱食度
8         :param comfort:舒适度
9         """
10        self.route = [startDot]
11        self.position = startDot.position
12        self.x, self.y, self.z = self.position
13        self.startDot = startDot
14        self.endDot = endDot
15        self.satiety, self.comfort = satiety, comfort
16        self.initSC = [satiety, comfort]
17
18    def update(self, route_set):
19        """
20        更新路径信息
21        :param route_set:路径
22        :return:
23        """
24        self.moveTo(self.startDot)
25        self.route.clear()
26        self.route.append(self.startDot)
27        self.satiety, self.comfort = self.initSC
28        for dot in route_set:
29            self.moveTo(dot)
30
31    def moveTo(self, targetDot):
32        """
33        移动到目标点
34        :param targetDot:目标点
35        :return:
36        """
37        currentDot = self.route[-1]
38        supply = targetDot.supply
39        # 到目标点要消耗的SC(饱食度和舒适度)值
40        loss = self.scCostCal(targetDot)
41        # 更新饱食度和舒适度
42        self.satiety = self.satiety + targetDot.isFood * targetDot.supply -
self.scCostCal(targetDot)
43        self.comfort = self.comfort + targetDot.isBonfire * targetDot.supply -
self.scCostCal(targetDot)
44        # 将目标点加入路径
45        self.route.append(targetDot)
46        # 更新玩家位置
47        self.position = targetDot.position
48        self.x, self.y, self.z = self.position
49        # 打印移动过程
50        # print('From', currentDot.index,
51        #       'to', targetDot.index,
52        #       currentDot.position, '--->', targetDot.position,
53        #       'S:', self.satiety,
54        #       'C:', self.comfort,
55        #       'Supply:', supply,
56        #       'Loss:', loss
57        #       )
58
59    def approachable(self, targetDot):
60        """
61        判断目标点是否能到达

```

```

62         :param targetDot:目标点
63         :return:
64         """
65         scCost = self.scCostCal(targetDot)
66         if targetDot == self.endDot:
67             return not ((self.satiety - scCost <= -3) or (self.comfort - scCost <= -3))
68         else:
69             return not ((self.satiety - scCost <= -5) or (self.comfort - scCost <= -5))
70
71     def getApproachableSet(self, dots):
72         """
73         返回当前状态下可到达的点的list
74         :param dots: 所有路径点
75         :return: 可到达路径点
76         """
77         approachable_set = []
78         for dot in dots:
79             if dot in self.route:
80                 continue
81             if player1.approachable(dot):
82                 approachable_set.append(dot)
83         return approachable_set
84
85     def scCostCal(self, targetDot):
86         """
87         计算到目标点消耗的舒适度和饱食度
88         :param targetDot:目标点
89         :return:消耗值
90         """
91         currentDot = self.route[-1]
92         if self.z > targetDot.z:
93             scCost = euclidDistance(currentDot.position, targetDot.position) * 4 / 100
94         elif self.z < targetDot.z:
95             scCost = euclidDistance(currentDot.position, targetDot.position) * 6 / 100
96         else:
97             scCost = euclidDistance(currentDot.position, targetDot.position) * 5 / 100
98         return scCost
99
100     def printInfo(self):
101         """
102         打印当前状态
103         :return:
104         """
105         print('Position: ', self.position, 'satiety: ', self.satiety, 'comfort: ',
106               self.comfort, 'step: ',
107               len(self.route) - 1)
108
109     def getInfo(self):
110         """
111         获得当前状态
112         :return: 当前状态
113         """
114         info = 'Position: ' + str(self.position) + ' Satiety: ' + str(self.satiety) + '
115         Comfort: ' + str(
116             self.comfort) + ' Step: ' + str(len(self.route) - 2)
117         return info
118
119     def printRoute(self):
120         """
121         打印当前整条路径
122         :return:

```

```

121         """
122         for i, dot in enumerate(self.route):
123             try:
124                 print('Step', i, ': ', self.route[i].index, '--->', self.route[i +
125 1].index, self.route[i].position,
126                       '--->',
127                       self.route[i + 1].position)
128             except:
129                 return
130
131     def getRoute(self):
132         """
133         返回玩家路径
134         """
135         x = []
136         y = []
137         z = []
138         for i, dot in enumerate(self.route):
139             x.append(dot.x)
140             y.append(dot.y)
141             z.append(dot.z)
142         return x, y, z
143
144     def getRouteLength(self):
145         """
146         返回路径长度
147         :return:
148         """
149         length = 0
150         for i, dot in enumerate(self.route):
151             if i < len(self.route) - 1:
152                 length += euclidDistance(self.route[i].position, self.route[i +
153 1].position)
154         return length

```

在路径点类和玩家类之外，还增添了计算两点位置之间欧氏距离的函数：

```

1 def euclidDistance(currentPos, targetPos):
2     """
3     求两个位置坐标的欧式距离
4     :param currentPos: 当前坐标
5     :param targetPos: 目标坐标
6     :return: 欧氏距离(float)
7     """
8     return np.sqrt(np.sum(np.square(currentPos - targetPos)))

```

### 2.1.3 问题分析

经过可视化分析和小组讨论，我们将本次任务题理解为路径规划问题。这类问题通常可以通过状态空间搜索的方法解决。

目前主流的路径规划方法按照搜索方式可以分为传统搜索算法、启发式搜索算法和智能算法<sup>2</sup>。

传统搜索算法主要包括深度优先搜索和广度优先搜索。深度优先搜索算法（简称DFS）是一种用于遍历或搜索树或图的算法。沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点的所在边都被探寻过，搜索将回溯到发现节点的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。广度优先搜索算法（简称BFS）又称为宽度优先搜索从起点开始，首先遍历起点周围邻近的点，然后再遍历已经遍历过的点邻近的点，逐步的向外扩散，直到找到终点。在执行算法的过程中，每个点需要记录达到该点的前一个点的位置——父节点。



点。这样做之后，一旦到达终点，便可以从终点开始，反过来顺着父节点的顺序找到起点，由此就构成了一条路径。由于DFS和BFS没有利用环境信息，不适合本题最短路径算法，因此这里不做过多拓展。

启发式算法最经典的当属A\*算法<sup>3</sup>。A\*算法吸取了Dijkstra算法中的当前代价，为每个边长设置权值，不停的计算每个顶点到起始顶点的距离，以获得最短路线，同时也汲取贪婪最佳优先搜索算法中不断向目标前进优势，并持续计算每个顶点到目标顶点的距离，以引导搜索队列不断想目标逼近，从而搜索更少的顶点，保持寻路的最优解。A\*算法在运算过程中，每次从优先队列中选取 $f(n)$ 值最小（优先级最高）的节点作为下一个待遍历的节点。A\*算法使用两个集合来表示待遍历的节点，与已经遍历过的节点，这通常称之为open\_set和close\_set。

A\*算法性能很大程度上依赖于启发函数 $f(n)$ 的设计。 $f(n) = g(n) + h(n)$ ， $f(n)$ 是节点 $n$ 的综合优先级。当我们选择下一个要遍历的节点时，我们总会选取综合优先级最高（值最小）的节点。 $g(n)$ 是节点 $n$ 距离起点的代价。 $h(n)$ 是节点 $n$ 距离终点的预计代价，这也就是A\*算法的启发函数。估价值由顶点到起始顶点的距离（代价）加上顶点到目标顶点的距离（启发函数）之和构成。过调节启发函数我们可以控制算法的速度和精确度。因为在一些情况，我们可能未必需要最短路径，而是希望能够尽快找到一个路径即可。

智能算法有很多，例如粒子群算法和蚁群算法。粒子群算法（Particle swarm optimization, PSO）是模拟群体智能所建立起来的一种优化算法，主要用于解决最优化问题（optimization problems）。1995年由Eberhart和Kennedy提出，是基于对鸟群觅食行为的研究和模拟而来的。蚁群算法是一种用来寻找优化路径的概率型算法。它由Marco Dorigo于1992年在他的博士论文中提出，其灵感来源于蚂蚁在寻找食物过程中发现路径的行为。这种算法具有分布计算、信息正反馈和启发式搜索的特征，本质上是进化算法中的一种启发式全局优化算法。

在本次小组作业前期的探索中，我和孙浏鑫同学兵分两路分别尝试了A\*算法和蚁群算法。

针对基于蚁群算法的思路，孙浏鑫同学已完成了三维TSP动态规划最小路径的程序实现工作。但是在添加饱食度和舒适度的限定条件时，遇到了较大困难，同时由于路径规划过程中涉及点数较多，更新速度也过慢，因此最终这条路径被我们放弃了。

最终我们选择了使用A\*算法作为本次建模的核心算法。

## 2.2 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。

### 2.2.1 A\*算法：

#### 传统的A\*算法

以下是传统的A\*算法的流程<sup>4</sup>：

```
1  *初始化open_set和close_set;
2  *将起点加入open_set中，并设置优先级为0（优先级最高）；
3  *如果open_set不为空，则从open_set中选取优先级最高的节点n:
4      *如果节点n为终点，则：
5          *从终点开始逐步追踪parent节点，一直达到起点；
6          *返回找到的结果路径，算法结束；
7      *如果节点n不是终点，则：
8          *将节点n从open_set中删除，并加入close_set中；
9          *遍历节点n所有的邻近节点：
10             *如果邻近节点m在close_set中，则：
11                 *跳过，选取下一个邻近节点
12             *如果邻近节点m不在open_set中，则：
13                 *设置节点m的parent为节点n
14                 *计算节点m的优先级
15                 *将节点m加入open_set中
```

传统的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

改进后的A\*算法流程如下：

```
1  *初始化route_set, open_set和approachable_set
2  *将起点加入route_set中
3  *更新当前player状态
4  *更新当前approachable_set, 若节点在route_set中, 则不加入
5  *将approachable_set加入到open_set中
6  *如果open_set不为空且route_set长度小于25(剪枝常数), 则:
7      *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:
8          *如果节点为终点, 则:
9              *将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
10             *更新当前player状态
11             *保存完整路径
12             *continue
13          *如果节点n不是终点, 则:
14              *将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
15              *更新当前player状态
16              *更新当前approachable_set, 若节点在route_set中, 则不加入
17              *将approachable_set加入到open_set中
18      *如果open_set[-1]为空, 则:
19          *open_set.pop()
20          *route_set.pop()
21          *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。

改进后的A\*算法最核心之处在于：

1. 每次搜索的后继节点作为一个整体（approachable\_set）被加入到open\_set中。每次执行下一次搜索时，先判断open\_set的最后一个元素列表是否为空。为空就表明这条路径已经不存在可行的后继结点，并将open\_set的最后一个元素弹出。
2. 使用route\_set替代传统A\*算法的close\_set，相当于每次搜索都记录了历史路径，而不是将所有搜索过的节点无序保存。完美避免了使用parent节点进行回溯导致无法更新玩家历史状态的问题。

改进的A\*算法代码实现如下：

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

```

4 route_set, open_set, approachable_set = [], [], []
5 # 将起点加入route_set中
6 route_set.append(dots[0])
7 # 更新当前player状态
8 player.update(route_set)
9 # 更新当前approachable_set, 若节点在route_set中, 则不加入
10 approachable_set = player.getApproachableSet(dots)
11 # 将approachable_set加入到open_set中
12 open_set.append(approachable_set[:])
13 # 如果open_set不为空
14 while open_set:
15     # 如果open_set[-1]不为空
16     if open_set[-1]:
17         # 从open_set[-1]中根据*启发式*选取优先级最高的点n:
18         i, bestDot = strategy(player, open_set[-1][:])
19         # 如果节点为终点
20         if bestDot == player.endDot:
21             res += 1
22             print(res)
23             # 将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
24             route_set.append(open_set[-1].pop(i))
25             # 更新当前player状态
26             player.update(route_set)
27             if res <= epoch: # epoch次搜索
28                 # 进行下一次搜索
29                 route_set.pop()
30                 player.update(route_set)
31                 continue
32             else:
33                 return searchResult
34         # 如果节点n不是终点
35         else:
36             # 将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
37             route_set.append(open_set[-1].pop(i))
38             # 更新当前player状态
39             player.update(route_set)
40             # 更新当前approachable_set, 若节点在route_set中, 则不加入
41             approachable_set.clear()
42             approachable_set = player.getApproachableSet(dots)
43             # 将approachable_set加入到open_set中
44             open_set.append(approachable_set[:])
45         # 如果open_set[-1]为空
46         else:
47             open_set.pop()
48             route_set.pop()
49             player.update(route_set)

```

还尝试了当搜索路径长度大于50时剪枝, 但在实际搜索过程中发现没有必要, 因此不在此赘述。

## 2.2.2 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:

$$max(\text{当前点到终点消耗能量} - \text{目标点到终点消耗能量} + \text{目标点的能量} - \text{到目标点消耗的能量}) \quad (1)$$

代码实现如下：

```
1 def leastDotsStrategy(player, approachable_set):
2     """
3     第一问策略：
4     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]
5     """
6     currentDot = player.route[-1]
7     endDot = player.endDot
8
9     bestDotIndex = None
10    bestDot = None
11    cost = -float('inf')
12    for i, dot in enumerate(approachable_set):
13        if dot == player.endDot:
14            bestDotIndex, bestDot = i, dot
15            break
16        else:
17            tmp = scCostCal(currentDot, endDot) - scCostCal(dot, endDot) + dot.supply -
scCostCal(currentDot, dot)
18            if tmp > cost:
19                bestDotIndex, bestDot = i, dot
20                cost = tmp
21    return bestDotIndex, bestDot
```

## 2.3 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：

$$\min(\text{目标点到终点的距离}) \quad (2)$$

代码实现如下：

```
1 def leastDistanceStrategy(player, approachable_set):
2     """
3     第二问策略
4     使[目标点到终点距离]尽可能小
5     """
6     endDot = player.endDot
7
8     bestDotIndex = None
9     bestDot = None
10
11    cost = float('inf')
12
13    for i, dot in enumerate(approachable_set):
14        if dot == player.endDot:
15            bestDotIndex, bestDot = i, dot
16            break
17        else:
18            tmp = euclidDistance(dot.position, endDot.position)
```

```

19         if tmp < cost:
20             bestDotIndex, bestDot = i, dot
21             cost = tmp
22     return bestDotIndex, bestDot

```

## 2.4 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。

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**。

在搜索过程中增加制作火把策略：当路径点为篝火点时，若当前饱食度高于舒适度0.5以上，则进行篝火制作。

## 启发式设计

与前两问相似，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。

$$\max(\text{目标点的能量} - \text{到目标点消耗的能量}) \quad (3)$$

代码实现如下：

```

1  def maxSCStrategy(player, approachable_set):
2      """
3      第三问策略
4      max[目标点的能量-到目标点消耗的能量]
5      """
6      currentDot = player.route[-1]
7      endDot = player.endDot
8
9      bestDotIndex = None
10     bestDot = None
11     cost = -float('inf')
12     for i, dot in enumerate(approachable_set):
13         if dot == player.endDot:
14             bestDotIndex, bestDot = i, dot
15             break
16         else:
17             tmp = dot.supply - scCostCal(currentDot, dot)
18             if tmp > cost:
19                 bestDotIndex, bestDot = i, dot
20                 cost = tmp
21     return bestDotIndex, bestDot

```

## 3 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。

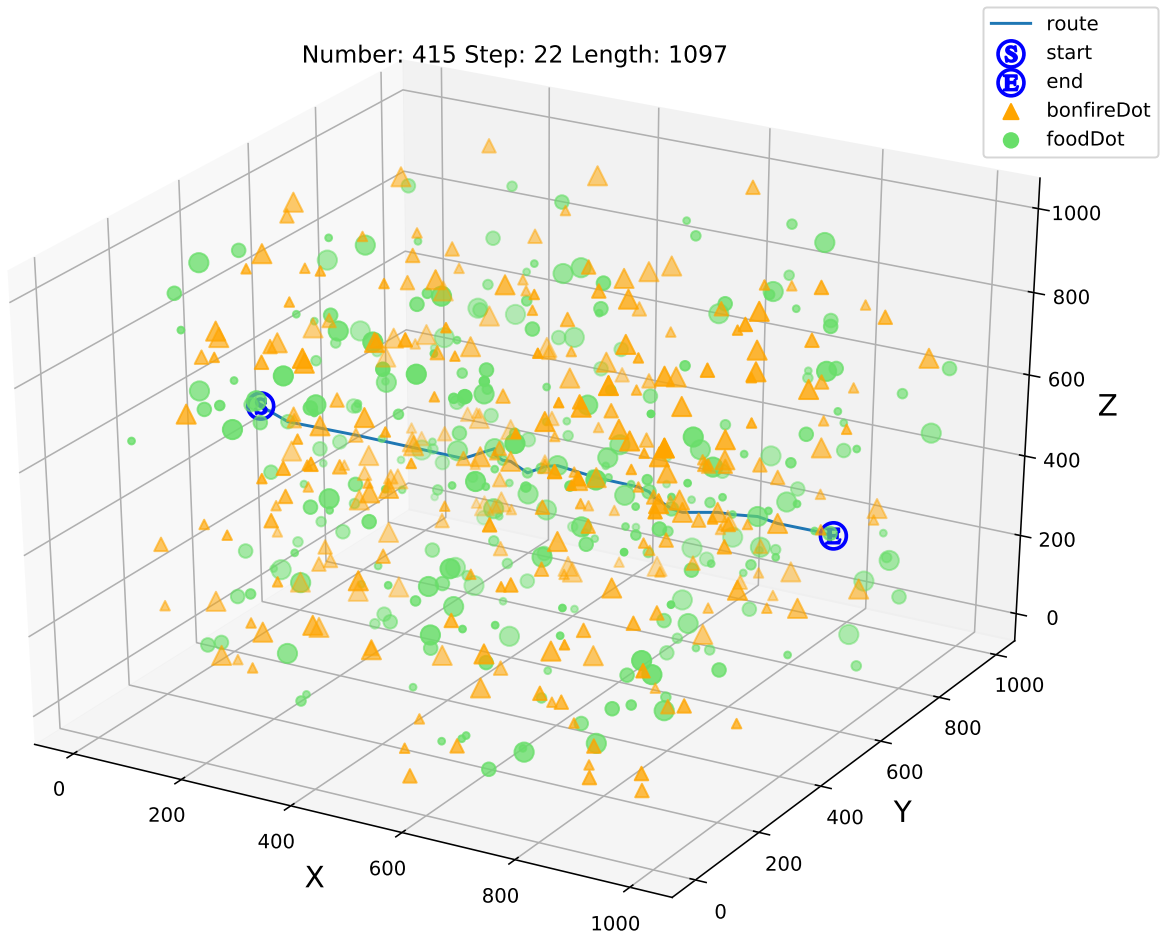
## 问题1

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。



详细信息如下：

```
1 Position: [1000.    555.67  442.022]
2 Satiety: -1.8662648167212543 Comfort: -2.8662648167212534 Step: 22
  time_cost:69.64824748039246
```

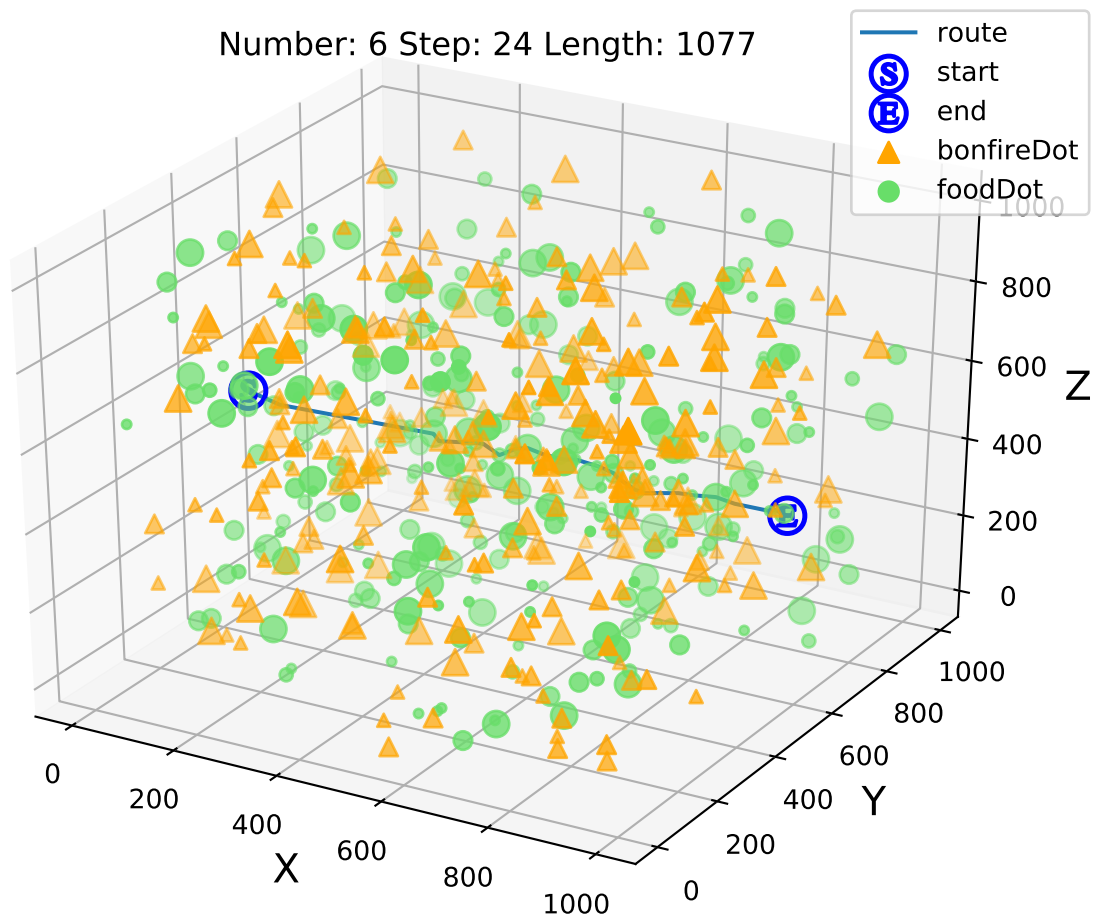
## 问题2

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。

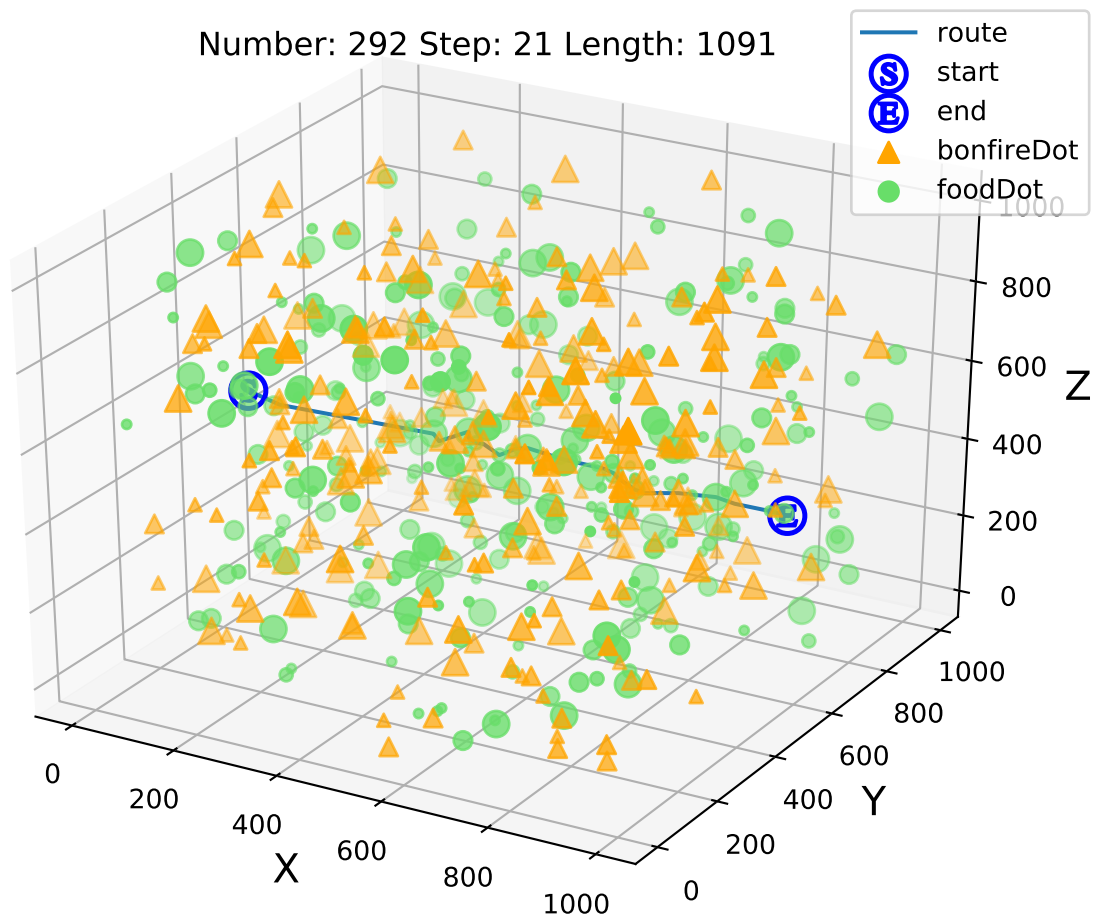


值得注意的是，在第292次搜索中得到了比第一问更好的结果，一条21个点的路径：

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。



### 问题3

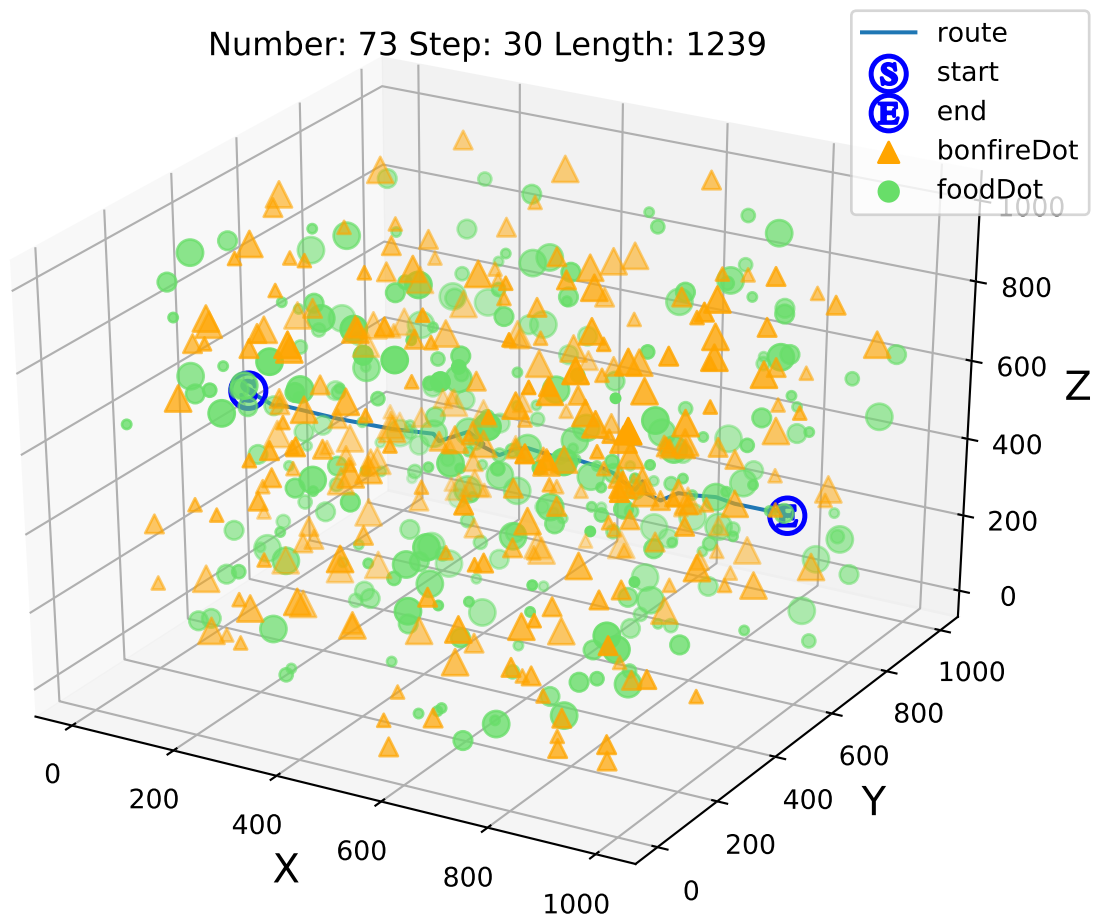
在第73次搜索中得到终点处饱食度和舒适度最高的路径，饱食度为0.95，舒适度为0.946，耗时8秒。

路径点序列为：

[0, 25, 33, 91, 133, 191, 200, 232, 254, 243, 270, 296, 307, 338, 345, 374, 387, 402, 411, 434, 446, 452, 463, 479, 501, 518, 537, 569, 576, 586, 587]

相应的移动距离和经过的补给点数量显著增加：经过30个路径点，路径长度为1239。



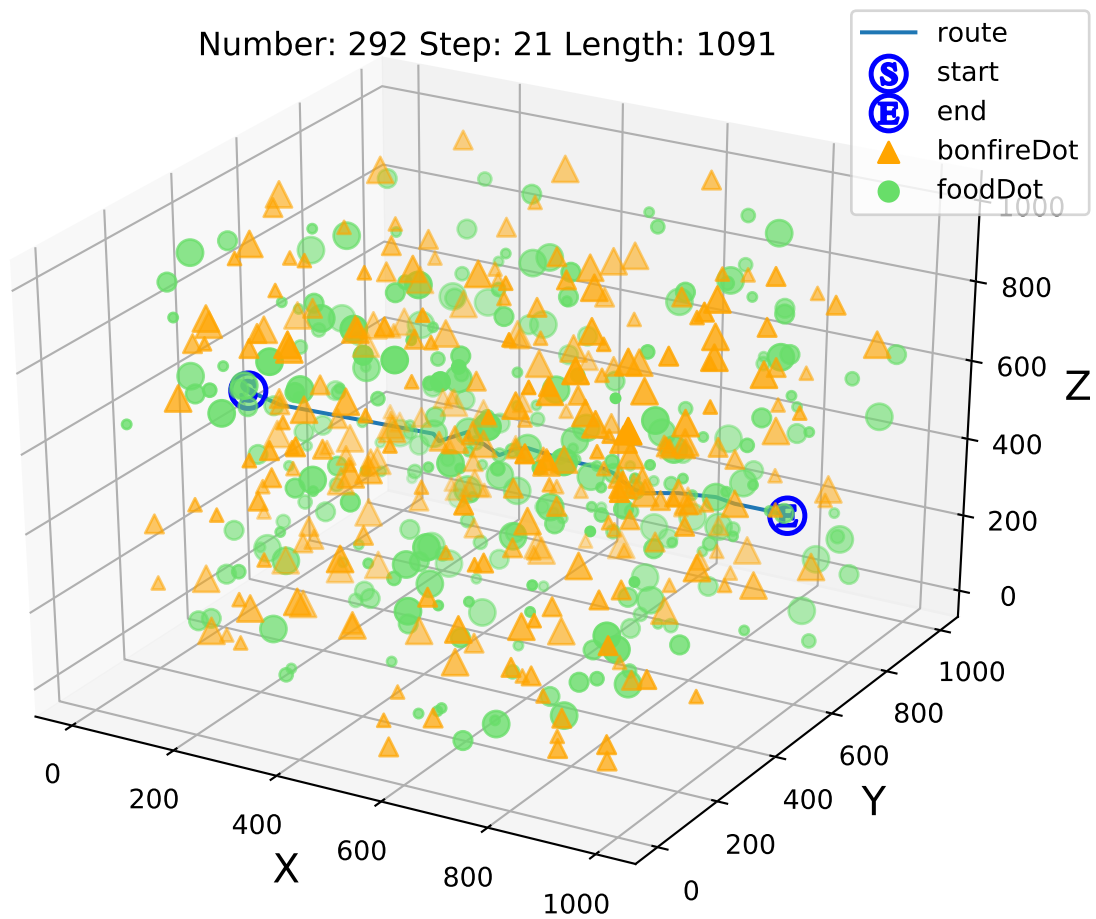


## 总结

问题一最优解:

21次补给

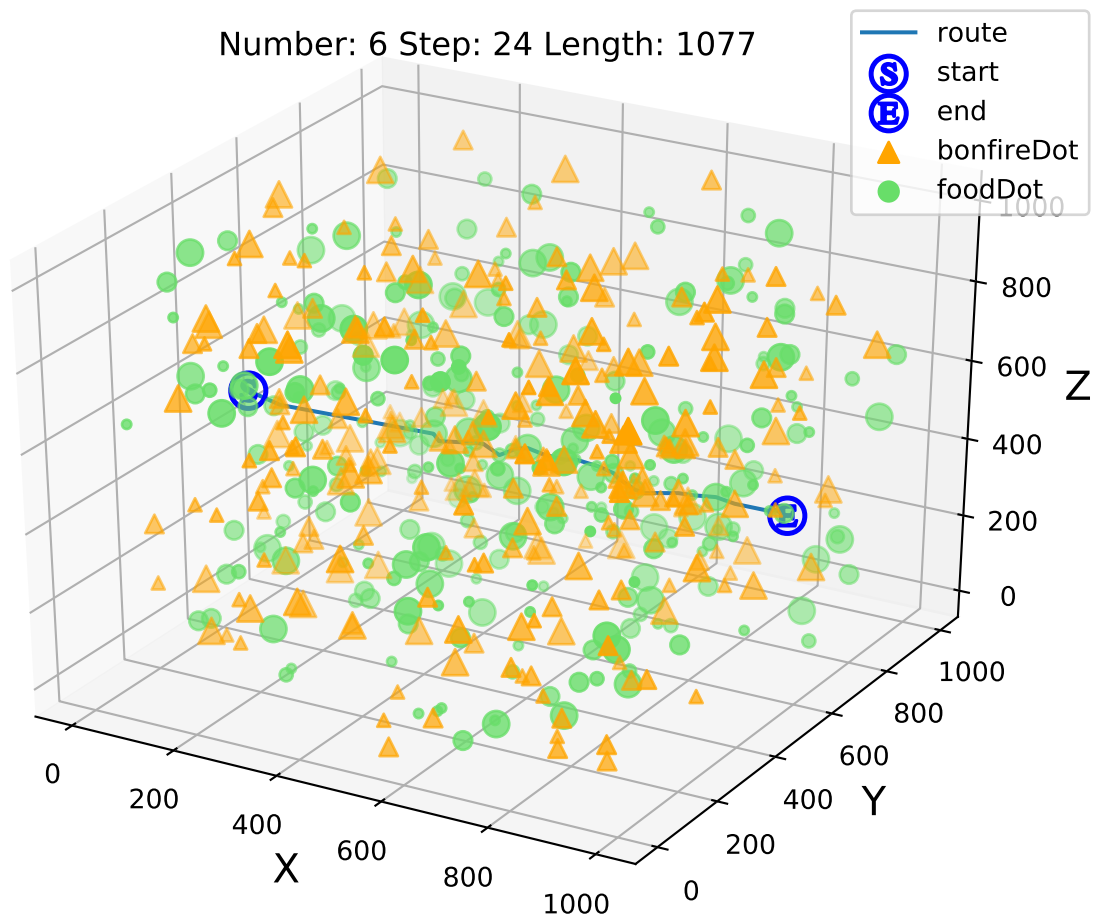
路径: [0, 33, 191, 200, 232, 230, 243, 254, 270, 296, 307, 338, 387, 402, 411, 434, 452, 479, 518, 537, 569, 587]



问题二最优解:

路径1077米

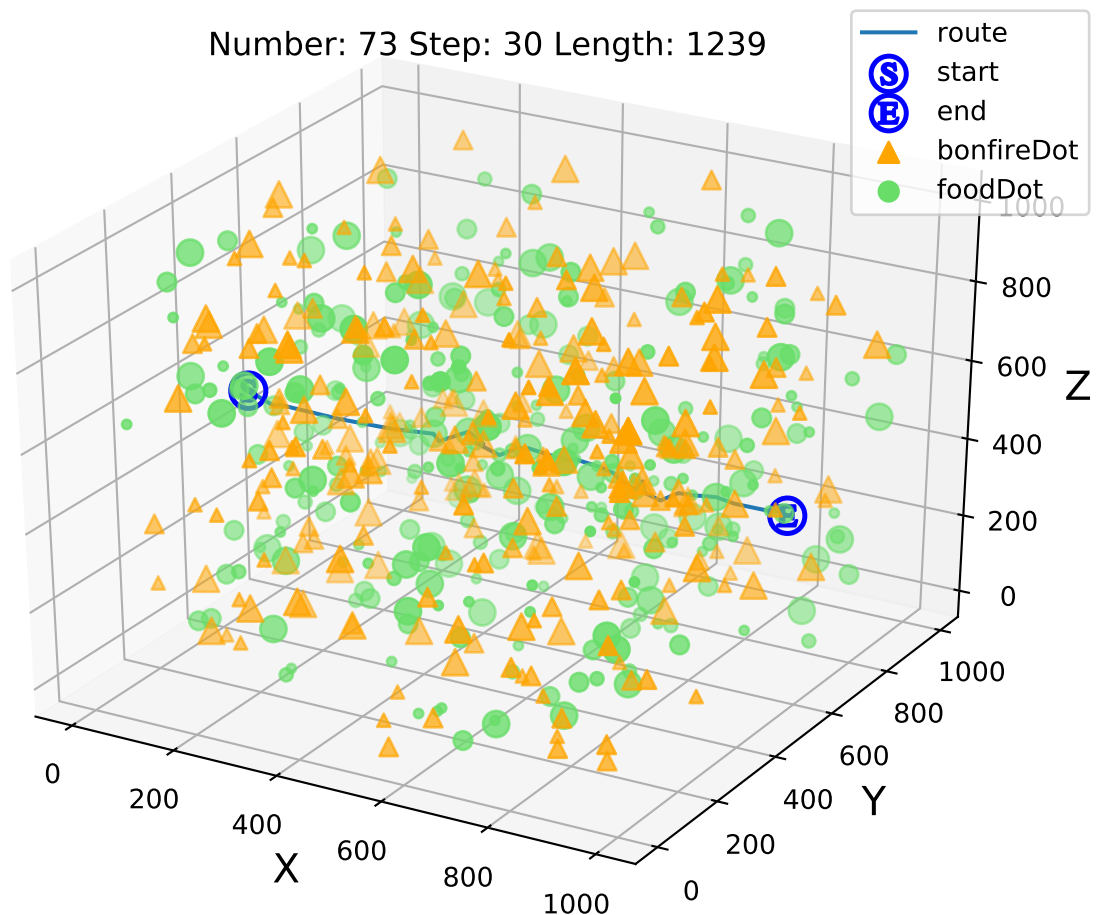
路径: [0, 33, 191, 200, 243, 254, 270, 296, 307, 326, 338, 345, 374, 387, 402, 411, 434, 452, 479, 501, 518, 537, 569, 576, 587]



问题三最优解:

饱食度为0.95, 舒适度为0.946

路径: [0, 25, 33, 91, 133, 191, 200, 232, 254, 243, 270, 296, 307, 338, 345, 374, 387, 402, 411, 434, 446, 452, 463, 479, 501, 518, 537, 569, 576, 586, 587]



综合来看，经过恰当的设定 $A^*$ 算法的启发式，搜索算法的倾向性策略有明显的区分，并很好的完成了三个问题的路径搜索。实际上，若不限制搜索结果数量为500，而是无限制的搜索，不难发现可行路径数量是无限的（至少在有限时间内无法结束算法）。因此使用普通的深度优先搜索是无法在有限时间内逼近最优结果的，这也是 $A^*$ 算法的价值所在。

实际搜索过程中问题一的启发式表现不如问题二的启发式，个人认为这是由于本题中路径点补给数量都分布在1-5，差距较小，导致问题二的启发式更直接的同时不会损失很多路径信息，从而达到了一举两得的作用。

## 4 心得体会

本次作业与期中的图像分析的很大不同之处在于：图像处理和分析任务通常有大量已有的库供我们使用，因此只要选择好技术路径，直接当掉包侠就好；但这次的任务虽然看起来简单，但需要我们自己从头搭建整个游戏模型和模拟玩家游戏过程，因此对我们的编码能力提出了一定挑战。经过这次任务，我学到了很多python编码的实用技巧，同时首次实用面向对象的方法实现了游戏环境的模拟，同时还学会了实用axes3D进行3d图像的绘制；在算法方面，由于传统的 $A^*$ 算法无法简单的套用到我们的任务当中，因此在如何改进优化 $A^*$ 算法上我走了许多弯路，包括尝试将问题转化为图论问题、尝试使用强化学习让玩家自己行走、使用蚁群算法等等，但都由于时间关系未能达到很好的效果。在经过大量的资料搜索后，我最终摸索到了通过增加route\_set解决 $A^*$ 算法路径记忆问题的方法，并最终实现了三个问题的搜索算法，较好的完成了任务。

本次作业分工情况：

作为组长，尤冠杰同学在前期进行了问题分析和技术路线的调研，并协调组员进行任务分工。

孙浏鑫同学之前有使用智能搜索算法的经验，因此主要负责在前期进行智能搜索算法的尝试，以及论文资料搜集。尤冠杰同学在上个学期选修了《人工智能》课程并接触了 $A^*$ 算法，因此主要负责 $A^*$ 算法及其改进模型的搭建、数据可视化模块以及最后论文的撰写。邓慧明同学主要负责数据的整理和论文资料搜集。本次建模过程中，我们小组的三人分工协作，发挥各自特长，以较高的效率完成了建模任务。

最后感谢王丹老师一直以来以来的精彩授课和课下答疑，让我们能顺利的完成本次作业！

# 附录

0. 完整代码运行需要将数据文件处理为csv格式并命名为'data.csv', 格式如下:

0	0	500	500	0	0
1	3.02638	656.271	618.879	0	4
2	4.12313	605.713	947.624	0	4
3	7.92612	841.923	473.195	1	3
4	10.5955	571.063	355.66	0	4
5	15.9735	179.247	170.539	0	2

1. 改进的A\*算法

```
1 def ygjAStar(player, dots, strategy, epoch=50):
2     """
3     改进的A*算法
4     :param player: 玩家
5     :param dots: 路径点
6     :param strategy: 启发式策略
7     :return:
8     """
9     def saveRoute(filename, player, time_cost, loop):
10         """
11         保存路径到txt文件
12         :param filename:
13         :param player:
14         :param time_cost:
15         :param loop:
16         :return:
17         """
18         with open(filename, 'a') as f:
19             f.write(player.getInfo())
20             f.write(' time_cost:' + str(time_cost) + '\n')
21             line = '['
22             for i, dot in enumerate(player.route):
23                 line = line + str(dot.index) + ', '
24             line = line + ']\n'
25             f.write(line)
26             for i, dot in enumerate(player.route):
27                 try:
28                     line = 'Step' + str(i) + ': ' + str(player.route[i].index) + '--->'
29                     + str(
30                         player.route[i + 1].index) + str(player.route[i].position) + '--
31                     ->' + str(
32                         player.route[i + 1].position)
33                     f.write(line)
34                     f.write('\n')
35                 except:
36                     break
37
38     time_start = time.time()
39     res = 0
40     minRouteLength = float('inf')
```

```

40 # searchResult用于保存搜索的可行路径结果
41 searchResult = []
42 # 初始化open_set, route_set和approachable_set
43 route_set, open_set, approachable_set = [], [], []
44 # 将起点加入route_set中
45 route_set.append(dots[0])
46 # 更新当前player状态
47 player.update(route_set)
48 # 更新当前approachable_set, 若节点在route_set中, 则不加入
49 approachable_set = player.getApproachableSet(dots)
50 # 将approachable_set加入到open_set中
51 open_set.append(approachable_set[:])
52 # 如果open_set不为空
53 while open_set:
54     # 如果open_set[-1]不为空
55     if open_set[-1]:
56         # 从open_set[-1]中根据*启发式*选取优先级最高的点n:
57         i, bestDot = strategy(player, open_set[-1][:])
58         # 如果节点为终点
59         if bestDot == player.endDot:
60             res += 1
61             print(res)
62             # 将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
63             route_set.append(open_set[-1].pop(i))
64             # 更新当前player状态
65             player.update(route_set)
66             # 记录route_set(不小于历史最优的路径不予以保存)
67             if True:
68                 minRouteLength = len(route_set) - 1
69                 time_end = time.time()
70                 time_cost = time_end - time_start
71                 filename = './route/' + str(res) + '-' + str(len(route_set) - 1) +
'.txt'
72                 saveRoute(filename, player1, time_cost, res)
73                 searchResult.append([res, player.getRouteLength(),
player.getRoute()])
74                 if res <= epoch: # 50次搜索
75                     # 进行下一次搜索
76                     route_set.pop()
77                     player.update(route_set)
78                     continue
79                 else:
80                     return searchResult
81             # 如果节点n不是终点
82             else:
83                 # 将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
84                 route_set.append(open_set[-1].pop(i))
85                 # 更新当前player状态
86                 player.update(route_set)
87                 # 更新当前approachable_set, 若节点在route_set中, 则不加入
88                 approachable_set.clear()
89                 approachable_set = player.getApproachableSet(dots)
90                 # 将approachable_set加入到open_set中
91                 open_set.append(approachable_set[:])
92             # 如果open_set[-1]为空
93             else:
94                 # print('back!')
95                 open_set.pop()
96                 route_set.pop()
97                 player.update(route_set)
98             # 显示部分

```

```

99         # i = os.system("cls")
100        # print('-----')
101        # print(len(open_set))
102        # player.printInfo()
103        # print(len(player.getApproachableSet(dots)))
104        # player.printRoute()
105        # print('-----')

```

## 2. 问题一启发式策略

```

1  def leastDotsStrategy(player, approachable_set):
2      """
3      第一问策略:
4      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
5      """
6      currentDot = player.route[-1]
7      endDot = player.endDot
8
9      bestDotIndex = None
10     bestDot = None
11     cost = -float('inf')
12     for i, dot in enumerate(approachable_set):
13         if dot == player.endDot:
14             bestDotIndex, bestDot = i, dot
15             break
16         else:
17             tmp = scCostCal(currentDot, endDot) - scCostCal(dot, endDot) + dot.supply -
scCostCal(currentDot, dot)
18             if tmp > cost:
19                 bestDotIndex, bestDot = i, dot
20                 cost = tmp
21     return bestDotIndex, bestDot

```

## 3. 问题二启发式策略

```

1  def leastDistanceStrategy(player, approachable_set):
2      """
3      第二问策略
4      使[目标点到终点距离]尽可能小
5      """
6      endDot = player.endDot
7
8      bestDotIndex = None
9      bestDot = None
10
11     cost = float('inf')
12
13     for i, dot in enumerate(approachable_set):
14         if dot == player.endDot:
15             bestDotIndex, bestDot = i, dot
16             break
17         else:
18             tmp = euclidDistance(dot.position, endDot.position)
19             if tmp < cost:
20                 bestDotIndex, bestDot = i, dot
21                 cost = tmp
22     return bestDotIndex, bestDot

```

#### 4. 问题三启发式策略

```
1 def maxSCStrategy(player, approachable_set):
2     """
3     第三问策略
4     """
5     currentDot = player.route[-1]
6     endDot = player.endDot
7
8     bestDotIndex = None
9     bestDot = None
10    cost = -float('inf')
11    for i, dot in enumerate(approachable_set):
12        if dot == player.endDot:
13            bestDotIndex, bestDot = i, dot
14            break
15        else:
16            tmp = dot.supply - scCostCal(currentDot, dot)
17            if tmp > cost:
18                bestDotIndex, bestDot = i, dot
19                cost = tmp
20    return bestDotIndex, bestDot
```

#### 5. 完整代码（需要在main进程中的调整策略以选择不同启发式）

```
1 #!usr/bin/env python
2 # -*- coding: utf-8 -*-
3 # @author:ygj
4 # @file: Game.py
5
6
7 import numpy as np
8 from matplotlib import pyplot as plt
9 from mpl_toolkits.mplot3d import Axes3D
10 import os
11 import time
12
13 '''
14 Search:ygjAStar
15 Q1:leastDotsStrategy
16 Q2:leastDistanceStrategy
17 Q3:maxSCStrategy
18 '''
19
20
21 class Dot:
22     def __init__(self, index, position, food, supply):
23         """
24         路径点类
25         :param index: 序号
26         :param position: 位置
27         :param isFood: 是否为食物
28         :param isBonfire: 是否为篝火
29         :param isArrived: 是否到达过
30         :param supply: 补给数
31         :param x, y, z: 坐标
32         """
33         self.index = int(index)
34         self.position = position
35         self.x, self.y, self.z = self.position
```



```

36         self.isFood = bool(food)
37         self.isBonfire = not bool(food)
38         self.supply = supply
39
40     def __eq__(self, other):
41         """
42         =运算符重载
43         判断两点是否相同
44         """
45         return (self.index == other.index) and (self.x == other.x) and (self.y ==
other.y) and (self.z == other.z)
46
47     def info(self):
48         """
49         查看当前路径点信息
50         """
51         if self.isFood:
52             print('Dot', self.index, ' Position: ', self.position, 'Food Supply: ',
self.supply)
53         elif self.isBonfire:
54             print('Dot', self.index, ' Position: ', self.position, 'BonFire Supply: ',
self.supply)
55
56
57 class Player:
58     def __init__(self, startDot, endDot, satiety, comfort):
59         """
60         玩家类
61         :param startDot:起点
62         :param endDot:终点
63         :param satiety:饱食度
64         :param comfort:舒适度
65         """
66         self.route = [startDot]
67         self.position = startDot.position
68         self.x, self.y, self.z = self.position
69         self.startDot = startDot
70         self.endDot = endDot
71         self.satiety, self.comfort = satiety, comfort
72         self.initSC = [satiety, comfort]
73
74     def update(self, route_set):
75         """
76         更新路径信息
77         :param route_set:路径
78         :return:
79         """
80         self.moveTo(self.startDot)
81         self.route.clear()
82         self.route.append(self.startDot)
83         self.satiety, self.comfort = self.initSC
84         for dot in route_set:
85             self.moveTo(dot)
86
87     def moveTo(self, targetDot):
88         """
89         移动到目标点
90         :param targetDot:目标点
91         :return:
92         """
93         currentDot = self.route[-1]

```

```

94         supply = targetDot.supply
95         # 到目标点要消耗的SC(饱食度和舒适度)值
96         loss = self.scCostCal(targetDot)
97         # 更新饱食度和舒适度
98         self.satiety = self.satiety + targetDot.isFood * targetDot.supply -
self.scCostCal(targetDot)
99         self.comfort = self.comfort + targetDot.isBonfire * targetDot.supply -
self.scCostCal(targetDot)
100         # 将目标点加入路径
101         self.route.append(targetDot)
102         # 更新玩家位置
103         self.position = targetDot.position
104         self.x, self.y, self.z = self.position
105         # 打印移动过程
106         # print('From', currentDot.index,
107         # 'to', targetDot.index,
108         # currentDot.position, '--->', targetDot.position,
109         # 'S:', self.satiety,
110         # 'C:', self.comfort,
111         # 'Supply:', supply,
112         # 'Loss:', loss
113         # )
114
115     def approachable(self, targetDot):
116         """
117         判断目标点是否能到达
118         :param targetDot: 目标点
119         :return:
120         """
121         scCost = self.scCostCal(targetDot)
122         if targetDot == self.endDot:
123             return not ((self.satiety - scCost <= -3) or (self.comfort - scCost <= -3))
124         else:
125             return not ((self.satiety - scCost <= -5) or (self.comfort - scCost <= -5))
126
127     def getApproachableSet(self, dots):
128         """
129         返回当前状态下可到达的点的list
130         :param dots: 所有路径点
131         :return: 可到达路径点
132         """
133         approachable_set = []
134         for dot in dots:
135             if dot in self.route:
136                 continue
137             if player1.approachable(dot):
138                 approachable_set.append(dot)
139         return approachable_set
140
141     def scCostCal(self, targetDot):
142         """
143         计算到目标点消耗的舒适度和饱食度
144         :param targetDot: 目标点
145         :return: 消耗值
146         """
147         currentDot = self.route[-1]
148         if self.z > targetDot.z:
149             scCost = euclidDistance(currentDot.position, targetDot.position) * 4 / 100
150         elif self.z < targetDot.z:
151             scCost = euclidDistance(currentDot.position, targetDot.position) * 6 / 100
152         else:

```

```

153         scCost = euclidDistance(currentDot.position, targetDot.position) * 5 / 100
154     return scCost
155
156     def printInfo(self):
157         """
158         打印当前状态
159         :return:
160         """
161         print('Position: ', self.position, 'satiety: ', self.satiety, 'comfort: ',
162 self.comfort, 'step: ',
163             len(self.route) - 1)
164
165     def getInfo(self):
166         """
167         获得当前状态
168         :return: 当前状态
169         """
170         info = 'Position: ' + str(self.position) + ' Satiety: ' + str(self.satiety) + '
171 Comfort: ' + str(
172             self.comfort) + ' Step: ' + str(len(self.route) - 2)
173         return info
174
175     def printRoute(self):
176         """
177         打印当前整条路径
178         :return:
179         """
180         for i, dot in enumerate(self.route):
181             try:
182                 print('Step', i, ': ', self.route[i].index, '--->', self.route[i +
183 1].index, self.route[i].position,
184                     '--->',
185                     self.route[i + 1].position)
186             except:
187                 return
188
189     def getRoute(self):
190         """
191         返回玩家路径
192         """
193         x = []
194         y = []
195         z = []
196         for i, dot in enumerate(self.route):
197             x.append(dot.x)
198             y.append(dot.y)
199             z.append(dot.z)
200         return x, y, z
201
202     def getRouteLength(self):
203         """
204         返回路径长度
205         :return:
206         """
207         length = 0
208         for i, dot in enumerate(self.route):
209             if i < len(self.route) - 1:
210                 length += euclidDistance(self.route[i].position, self.route[i +

```

```

210
211 def euclidDistance(currentPos, targetPos):
212     """
213     求两个位置坐标的欧式距离
214     :param currentPos: 当前坐标
215     :param targetPos: 目标坐标
216     :return: 欧氏距离(float)
217     """
218     return np.sqrt(np.sum(np.square(currentPos - targetPos)))
219
220
221 def scCostCal(startDot, targetDot):
222     """
223     求两个点间的饱食度舒适度消耗
224     :param startDot: 起点
225     :param targetDot: 终点
226     :return: 消耗值
227     """
228     if startDot.z > targetDot.z:
229         scCost = euclidDistance(startDot.position, targetDot.position) * 4 / 100
230     elif startDot.z < targetDot.z:
231         scCost = euclidDistance(startDot.position, targetDot.position) * 6 / 100
232     else:
233         scCost = euclidDistance(startDot.position, targetDot.position) * 5 / 100
234     return scCost
235
236
237 def routeVisual(searchResult):
238     # 路径信息: 第n条, 长度l, 路径点
239     n, l, route = searchResult
240     x, y, z = route[0], route[1], route[2]
241     # Load data
242     dots = np.loadtxt('./data.csv', delimiter=',')
243     routeDots = dots[1:-1, :]
244     bonfireIndex = [i for i, x in enumerate(routeDots[:, 4].tolist()) if x == 0]
245     foodIndex = [i for i, x in enumerate(routeDots[:, 4].tolist()) if x == 1]
246     startDot = dots[0, :] # 起点
247     endDot = dots[-1, :] # 终点
248     bonfireDots = routeDots[bonfireIndex, :] # 篝火点
249     foodDots = routeDots[foodIndex, :] # 食物点
250     x1, y1, z1 = bonfireDots[:, 1], bonfireDots[:, 2], bonfireDots[:, 3]
251     x2, y2, z2 = foodDots[:, 1], foodDots[:, 2], foodDots[:, 3]
252
253     # 绘图
254     fig = plt.figure()
255     ax = Axes3D(fig)
256     # 图例设置
257     startMarker = '$\circledS$'
258     endMarker = '$\circledE$'
259
260     # 散点绘制
261     ax.scatter(startDot[1], startDot[2], startDot[3], c='b', s=200, marker=startMarker,
262 label='start')
263     ax.scatter(endDot[1], endDot[2], endDot[3], c='b', s=200, marker=endMarker,
264 label='end')
265     ax.scatter(x1, y1, z1, c='#FFA500', s=np.exp2(bonfireDots[:, 5]) * 6, marker='^',
266 label='bonfireDot')
267     ax.scatter(x2, y2, z2, c='#68DE69', s=np.exp2(foodDots[:, 5]) * 3, label='foodDot')
268
269     # 路径绘制
270     ax.plot(x, y, z, label='route')

```

```

268
269     # 添加坐标轴(顺序是Z, Y, X)
270     ax.set_zlabel('Z', fontdict={'size': 15, 'color': 'black'})
271     ax.set_ylabel('Y', fontdict={'size': 15, 'color': 'black'})
272     ax.set_xlabel('X', fontdict={'size': 15, 'color': 'black'})
273     # 添加图例
274     ax.legend(loc='best')
275
276     line = 'Number: ' + str(n) + ' Step: ' + str(len(route[0]) - 2) + ' Length: ' +
str(int(round(l)))
277     plt.title(line)
278     # plt.show()
279     savepath = str(n) + '-' + str(len(route[0]) - 2) + '-' + str(round(l)) + '.svg'
280     fig.savefig(savepath, dpi=600)
281     plt.close()
282
283
284 def leastDotsStrategy(player, approachable_set):
285     """
286     第一问策略:
287     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
288     """
289     currentDot = player.route[-1]
290     endDot = player.endDot
291
292     bestDotIndex = None
293     bestDot = None
294     cost = -float('inf')
295     for i, dot in enumerate(approachable_set):
296         if dot == player.endDot:
297             bestDotIndex, bestDot = i, dot
298             break
299         else:
300             tmp = scCostCal(currentDot, endDot) - scCostCal(dot, endDot) + dot.supply -
scCostCal(currentDot, dot)
301             if tmp > cost:
302                 bestDotIndex, bestDot = i, dot
303                 cost = tmp
304     return bestDotIndex, bestDot
305
306
307 def leastDistanceStrategy(player, approachable_set):
308     """
309     第二问策略
310     使[目标点到终点距离]尽可能小
311     """
312     endDot = player.endDot
313
314     bestDotIndex = None
315     bestDot = None
316
317     cost = float('inf')
318
319     for i, dot in enumerate(approachable_set):
320         if dot == player.endDot:
321             bestDotIndex, bestDot = i, dot
322             break
323         else:
324             tmp = euclidDistance(dot.position, endDot.position)
325             if tmp < cost:

```

```

326         bestDotIndex, bestDot = i, dot
327         cost = tmp
328     return bestDotIndex, bestDot
329
330
331 def maxSCStrategy(player, approachable_set):
332     """
333     第三问策略
334     """
335     currentDot = player.route[-1]
336     endDot = player.endDot
337
338     bestDotIndex = None
339     bestDot = None
340     cost = -float('inf')
341     for i, dot in enumerate(approachable_set):
342         if dot == player.endDot:
343             bestDotIndex, bestDot = i, dot
344             break
345         else:
346             tmp = dot.supply - scCostCal(currentDot, dot)
347             if tmp > cost:
348                 bestDotIndex, bestDot = i, dot
349                 cost = tmp
350     return bestDotIndex, bestDot
351
352
353 def ygjAStar(player, dots, strategy, epoch=50):
354     """
355     改进的A*算法
356     :param player: 玩家
357     :param dots: 路径点
358     :param strategy: 启发式策略
359     :return:
360     """
361     def saveRoute(filename, player, time_cost, loop):
362         """
363         保存路径到txt文件
364         :param filename:
365         :param player:
366         :param time_cost:
367         :param loop:
368         :return:
369         """
370         with open(filename, 'a') as f:
371             f.write(player.getInfo())
372             f.write(' time_cost:' + str(time_cost) + '\n')
373             line = '['
374             for i, dot in enumerate(player.route):
375                 line = line + str(dot.index) + ', '
376             line = line + ']\n'
377             f.write(line)
378             for i, dot in enumerate(player.route):
379                 try:
380                     line = 'Step' + str(i) + ': ' + str(player.route[i].index) + '--->'
381 + str(
382                             player.route[i + 1].index) + str(player.route[i].position) + '--
383 ->' + str(
384                             player.route[i + 1].position)
385                     f.write(line)
386                     f.write('\n')

```

```

385         except:
386             break
387
388     time_start = time.time()
389     res = 0
390     minRouteLength = float('inf')
391
392     # searchResult用于保存搜索的可行路径结果
393     searchResult = []
394     # 初始化open_set, route_set和approachable_set
395     route_set, open_set, approachable_set = [], [], []
396     # 将起点加入route_set中
397     route_set.append(dots[0])
398     # 更新当前player状态
399     player.update(route_set)
400     # 更新当前approachable_set, 若节点在route_set中, 则不加入
401     approachable_set = player.getApproachableSet(dots)
402     # 将approachable_set加入到open_set中
403     open_set.append(approachable_set[:])
404     # 如果open_set不为空
405     while open_set:
406         # 如果open_set[-1]不为空
407         if open_set[-1]:
408             # 从open_set[-1]中根据*启发式*选取优先级最高的点n:
409             i, bestDot = strategy(player, open_set[-1][:])
410             # 如果节点为终点
411             if bestDot == player.endDot:
412                 res += 1
413                 print(res)
414                 # 将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
415                 route_set.append(open_set[-1].pop(i))
416                 # 更新当前player状态
417                 player.update(route_set)
418                 # 记录route_set(不小于历史最优的路径不予以保存)
419                 if True:
420                     minRouteLength = len(route_set) - 1
421                     time_end = time.time()
422                     time_cost = time_end - time_start
423                     filename = './route/' + str(res) + '-' + str(len(route_set) - 1) +
424                         '.txt'
425                     saveRoute(filename, player1, time_cost, res)
426                     searchResult.append([res, player.getRouteLength(),
427 player.getRoute()])
428                 if res <= epoch: # 50次搜索
429                     # 进行下一次搜索
430                     route_set.pop()
431                     player.update(route_set)
432                     continue
433                 else:
434                     return searchResult
435             # 如果节点n不是终点
436             else:
437                 # 将节点n从open_set[-1]中删除(open_set[-1].pop()), 并加入route_set中
438                 route_set.append(open_set[-1].pop(i))
439                 # 更新当前player状态
440                 player.update(route_set)
441                 # 更新当前approachable_set, 若节点在route_set中, 则不加入
442                 approachable_set.clear()
443                 approachable_set = player.getApproachableSet(dots)
444                 # 将approachable_set加入到open_set中
445                 open_set.append(approachable_set[:])

```

```

444         # 如果open_set[-1]为空
445         else:
446             # print('back!')
447             open_set.pop()
448             route_set.pop()
449             player.update(route_set)
450         # 显示部分
451         # i = os.system("cls")
452         # print('-----')
453         # print(len(open_set))
454         # player.printInfo()
455         # print(len(player.getApproachableSet(dots)))
456         # player.printRoute()
457         # print('-----')
458
459
460 if __name__ == "__main__":
461     # Load data
462     data = np.loadtxt('./data.csv', delimiter=',')
463     dots = []
464     for dotData in data:
465         dots.append(Dot(dotData[0], dotData[1:4], dotData[4], dotData[5]))
466     # Init player
467     player1 = Player(dots[0], dots[-1], 10, 10)
468     # A* Search
469     searchResults = ygjAStar(player1, dots, maxSCStrategy, epoch=500)
470     # Visualiation
471     for searchResult in searchResults:
472         routeVisual(searchResult)

```

## 参考文献

1. python绘制3D散点图 [↩](#)
2. 智能算法综述 [↩](#)
3. Introduction to the A\* Algorithm [↩](#)
4. 路径规划之 A\* 算法 [↩](#)