信网 2022 应用层大作业

哈佛 CS50 课程——搜索模块

姓名：乔千
学号：19211213
班级：通信 1904

Abstract

本次大作业是基于哈佛 CS50 七个模块下的搜索模块项目进行制作的。项目涉及多种搜索算法，如宽度优先算法，深度优先算法，贪婪算法，Astar 算法，爬山算法和模拟退火算法等，分别解决了迷宫问题与八皇后问题，其中宽度优先算法，深度优先算法，贪婪算法，Astar 算法是基于哈佛项目已由的代码完善后所得，爬山算法，模拟退火算法是笔者自己编写所得。此外，该项目还完成了搜索模块中的井字棋代码补全项目，并成功实现简单的人机博弈。

Type of Text	Font Size	Style
paper title	15 pt	**bold**
author names	12 pt	**bold**
author affiliation	12 pt	
the word “Abstract”	12 pt	**bold**
section titles	12 pt	**bold**
document text	11 pt	
captions	10 pt	
abstract text	10 pt	
bibliography	10 pt	
footnotes	9 pt	

Table 1: Font guide.

1 Introduction

搜索问题所描述的是给定初始状态和目标状态，通过相关算法找到从初始状态到达目标状态的方法，并返回相关路径，以实现问题的解决。此外，还可以在搜索的过程中引入成本的概念，以实现不同方案处理结果的比较，从而实现找到多种处理方案中的最优方案，部分问题还可以给出最优解。

本次项目涉及到三个问题分别是迷宫问题，八皇后问题与井字棋问题，在之后部分里笔者将分别使用不同算法实现该问题的解决，其中第二部分将详细介绍迷宫问题，第三部分将介绍八皇后问题，第四部分将介绍井字棋问题，第五部分笔者会将这些不同算法的结果进行比较，最后会在第六部分给出总结。

2 Maze Problem

2.1 Describe

迷宫问题就是在一个平面上设置起始点 A 与目标点 B，其间由多条道路与障碍链接，我们需要从中找到 A 通往 B 的道路。对于这种搜索问题，需要从初始状态开始设置其为父节点，找到下一步的动作，到达下一个状态，若该状态是目标状态则返回路径，否则设置当前状态为父节点，继续重复上述步骤，进行节点扩展，直到到达目标状态或者所有节点均无法在扩展为止。

2.2 Depth-First Search

前面在讲述问题时有一件事没有提到，就是应该扩展哪个节点，这种选择会影响解决方案的质量与实现速度，其中有两种数据结构可以用于选择相应的节点，分别是堆栈与队列。本小节所讲述的深度优先算法就是基于堆栈表示的。深度优先算法在尝试另一个方向前，会遍历当前方向的所有状态，在这种情况下，可以用表示“后进先出”的堆栈存储待扩展节点，在添加新的节点后，首先要考虑与删除的是最后一个添加的节点，这导致该算法在一个方向上尽可能的深入，将其他方向留到之后扩展。部分相关代码如下：

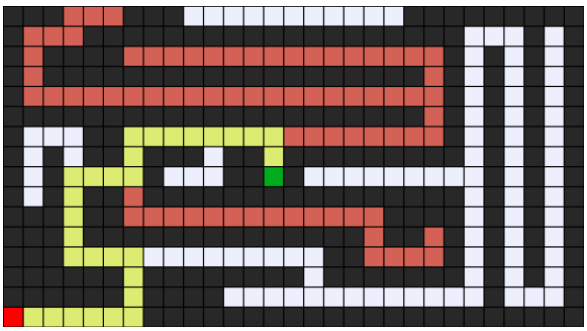
```
node = frontier.removedepth()
```

其中

```
def removedepth(self):
    if self.empty():
```

```
            raise Exception("empty frontier")
        else:
            node = self.frontier[-1]
            self.frontier = self.frontier[:-1]
        return node
```

通过深度优先算法输出的结果为：



其中红色代表起始点 A，绿色代表目标点 B，黄色表示算法返回的解，橘色表示算法运行所遍历的路径。

2.3 Breadth-First Search

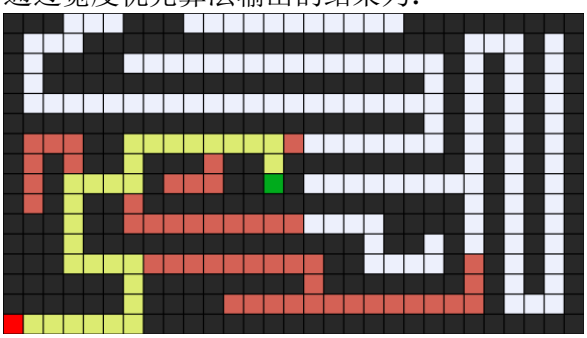
广度优先算法将同时跟踪多个方向，在每个可能的方向均扩展一步，然后再在每个方向上扩展第二步。在这种情况下，使用之前提到的另一种数据结构队列存储待扩展节点，这种情况下遵循“先进先出”原则，所有节点顺序相加，先添加的节点先考虑，这就使得该算法在每个方向均扩展一步后，才继续扩展第二步。相关代码如下：

```
            node = frontier.removebroadth()
```

其中

```
    def removebroadth(self):
        if self.empty():
            raise Exception("empty frontier")
        else:
            node = self.frontier[0]
            self.frontier = self.frontier[1:]
        return node
```

通过宽度优先算法输出的结果为：



图中表示形式与上一小结相同。

2.4 Greedy Best-First Search

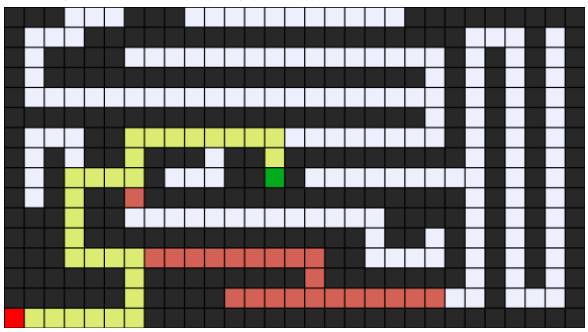
贪婪最佳优先算法扩展最接近目标的节点。由启发式函数 h(n)决定。顾名思义，该函数估计下一个节点离目标有多近，选择最近的节点扩展。在迷宫中，算法可以使用依赖于可能节点和迷宫目标节点之间的迈哈顿距离的启发式函数。曼哈顿距离忽略墙壁并且计算从一个位置到目标位置所需的向上与向下的步数。这个简单的估计，可以基于当前位置和目标位置的坐标得出。相关代码如下：

```
            min = 0
            num = 0
            length = 999
            for tmp in frontier.frontier:
                x1 = tmp.state[0]
                y1 = tmp.state[1]
                x2 = self.goal[0]
                y2 = self.goal[1]
                l = abs(x2 - x1) + abs(y2 - y1)
                if l <= length:
                    min = num
                    length = l
                num += 1
            node = frontier.remove(min)
```

其中

```
    def remove(self, i):
        node = self.frontier[i]
        self.frontier=self.frontier[:i] + \
                      self.frontier[i+1:]
    return node
```

通过贪婪最佳优先算法输出的结果为：



图中表示形式与上一小结相同。

2.5 Astar Search

Astar 算法不仅考虑从当前位置到到目标位置的估计成本，还要考虑当前位置所的累积成本，通过结合这两个值，该算法可以更准确的选择路径，并随时随地的优化其选择，该算法跟踪到现在为止的累积成本与到目标的估计成本，一旦他超过某个先前路径的累积成本与

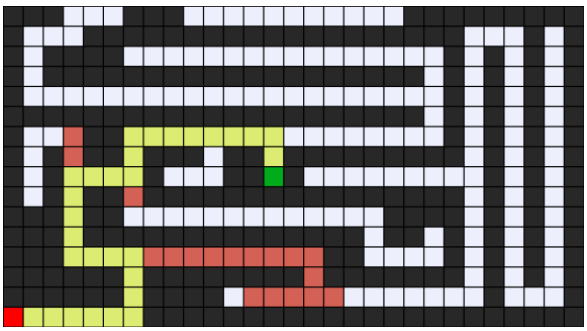
估计成本，算法将放弃当前路径并返回到先前路径，从而防止自己沿着一条更长的路径前进。相关代码如下：

```
    min = 0
    num = 0
    length = 999
    for tmp in frontier.frontier:
        x1 = tmp.state[0]
        y1 = tmp.state[1]
        x2 = self.goal[0]
        y2 = self.goal[1]
        l = abs(x2-x1) + abs(y2-y1) + \
            count[num]
        if l <= length:
            min = num
            length = l
        num += 1
    node = frontier.remove(min)
    way = count[min] + 1
    count = count[:min] + count[min + 1:]
```

其中

```
    def remove(self, i):
        node = self.frontier[i]
        self.frontier=self.frontier[:i] +\
                      self.frontier[i+1:]
        return node
```

通过 Astar 算法输出的结果为：



图中表示形式与上一小结相同。

3 Eight Queens Problem

3.1 Describe

所谓八皇后问题就是在 8×8 格的国际象棋上摆放 8 个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，求出皇后的摆法。这里我们可以先将问题抽象成模型，由于规定棋盘的同列只能出现一个皇后，因此每一个棋盘可用一个长度为 8 的序列表示，序列中的每一个数的范围是[1, 8]，第 k 个数字所代表的含义是第 k 列中皇后所在的行数，如[5, 6, 7, 4, 5, 6, 7, 6]代表棋盘上从第一列到第八列，皇后所摆放的行数分别为第 5, 6, 7, 4, 5, 6, 7, 6 行。将当前皇后移动到同列的其他 7 个格子便可以得到当前状态的后继状态。通过计算每个后继状态的相互攻击的皇后对的个数并从中选择合适的值保留，并如此往复便可以得到问题的解。

3.2 Hill Climbing

所谓爬山算法从当前的节点开始，和周围的邻居节点的值进行比较。如果当前节点是最大的，那么返回当前节点，作为最大值(既山峰最高点)，反之就用最高的邻居节点来，替换当前节点，从而实现向山峰的高处攀爬的目的。如此循环直到达到最高点。

对于八皇后问题，爬山算法计算每个后继状态的形成相互攻击的皇后对的个数，保留值最小的情况，该函数的全局最小值是零，仅在找到解时才会是这个值。

实验结果如下：

```
[4, 1, 7, 5, 2, 0, 3, 6] 1
对应棋盘如下:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
[3, 1, 7, 4, 0, 7, 2, 6] 2
对应棋盘如下:
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
[5, 7, 2, 0, 5, 1, 4, 6] 0
对应棋盘如下:
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 1 0 0
0 0 0 0 0 0 0 0
```

我们可以看出爬山算法不一定可以找到解。

3.3 Simulated Annealing Algorithm

模拟退火算法与爬山算法有一定相似度，但是爬山算法不“下山”，而模拟退火算法会依概率“下山”。内层循环与爬山类似，但是没有选择最佳移动，而是随机移动，如果该移动使情况改善，则该移动被接受，否则以某个小于 1 的概率接受该移动，这个概率随着温度下降而减小。容易推断出山峰越高被选择的概率越大，下山越困难，从整体上看，算法选择的点集中在更高的山峰，随着温度的下降选择逐渐稳定。

对于八皇后问题，模拟退火算法计算形成相互攻击的皇后对的个数，当小于现在状态时则保留，当大于现在状态时，会依概率保留，全局最小值同样是零，仅在找到解时才会是这个值。

实验结果如下：

```
对应棋盘如下：
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
成功
```

```
对应棋盘如下：
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
成功
```

4 Project——Adversarial Search

4.1 Describe

之前，我们已经讨论了需要寻找到问题答案的算法，然而我们常常会遇到双方相互博弈的问题，即双发试图实现对方相反的目标，这就是我们接下来要讨论的对抗搜索算法。通常在游戏中会遇到使用这种搜索算法的 AI，如井字棋游戏。

4.2 Minimax

Minimax 是一种对抗搜索算法，它将获胜条件表示为一方为（-1），另一方为（+1）。下一步的行动将由这些条件决定，最小化方试图获得最低分数，最大化方试图获得最高分数。

4.3 Representing a Tic-Tac-Toe AI

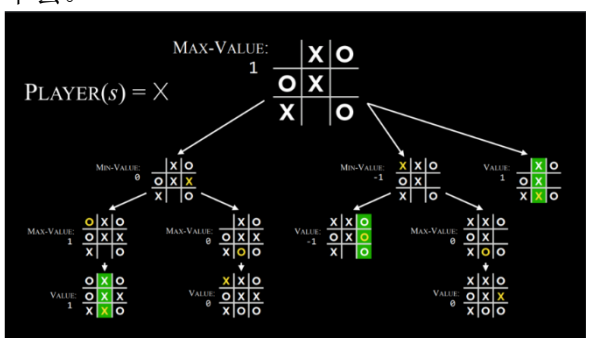
设计模型如下：

- S_0: 初始状态（本例中，是一个空的 3X3 棋盘）
- Player（s）：一个函数，给定一个状态 s，返回该轮到哪个玩家（X 或 O）。
- Actions（s）：一个函数，给定一个状态 s，返回该状态下的所有可能移动（棋盘上有哪些位置是空闲的）。
- Result（s，a）：给定状态 s 和动作 Actions，返回新状态的函数。这是返回在状态 s 上执行动作（在游戏中移动）所产生的棋盘。
- Terminal（s）：一个函数，在给定状态 s 的情况下，检查这是否是游戏中的最后一步，即是否有人赢了或打成平局。如果游戏结束，则返回 True，否则返回 False。
- Utility（s）：给定 Terminal 状态，返回状态的值：-1、0 或 1。

4.4 How the algorithm works

该算法通过递归的方法模拟所有可能的游戏情况。这些游戏从当前状态开始，直到达到终端状态，每个终端状态的值为（-1）、0 或（+1）。

根据已知将会轮到谁，该算法可以知道当前玩家希望得到的值更大还是更小。通过这种方式，算法在最小化和最大化之间交替，算法为每个可能的操作产生的状态赋值。对于最大化玩家，他需要思考最小化玩家下一步的操作，那么最小化玩家一定会选择一个值最小的情况，因此最大化玩家便会从这些最小值中情况中选择一个最大的值以阻止最小化玩家获胜。同理，最小化玩家也会预测最大化玩家下一步操作，并从最大值中选择一个最小的值以阻止最大化玩家获胜。之后以此类推，便可以将游戏进行下去。



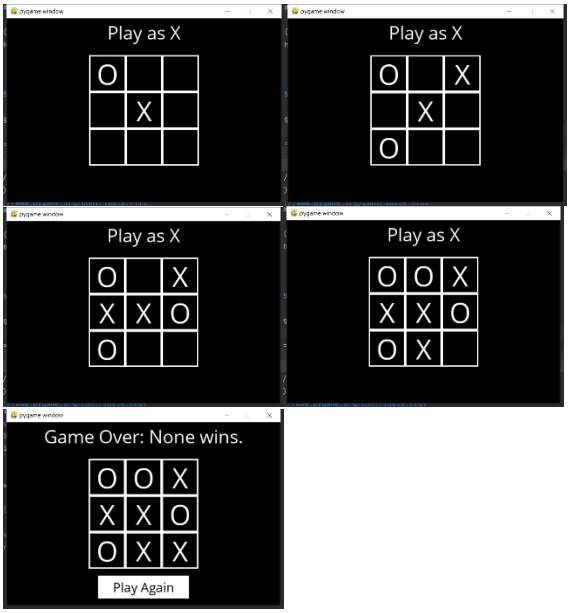
相关代码如下：

```
def minvalue(board):
    if terminal(board):
        return utility(board)
    v = 255
```

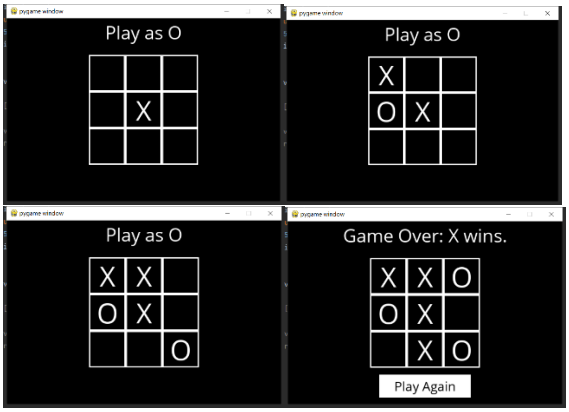
```
        for action in actions(board):
            b = copy.deepcopy(board)
            v = min(v, (maxvalue(result(b,
action))))
        return v

    def maxvalue(board):
        if terminal(board):
            return utility(board)
        v = -255
        for action in actions(board):
            b = copy.deepcopy(board)
            v = max(v, (minvalue(result(b,
action))))
        return v
```

玩家持 X 机器持 O 的实验结果：



玩家持 O 机器持 X 的实验结果：

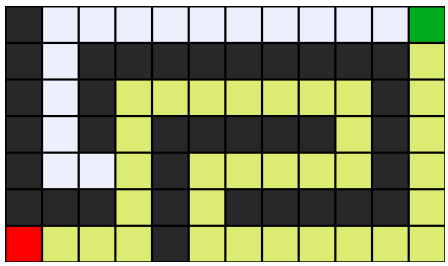


5 Analyse

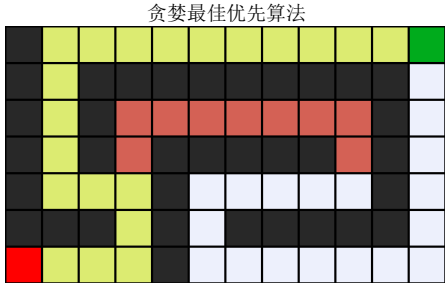
5.1 How the algorithm works

对比之前提到的四种搜索算法，对于深度优先算法而言，如果他比较幸运那么他找到解的速度会是最快的，然而通常都不会那么幸运，这会使得该算法花费很长时间，同时得到的解也未必是最优解，对于广度优先算法而言该算法可以保证结果一定是最优解，但是其运行时间比较长，而对于后两种算法而言，虽然结果不一定是最优解但是运行速度较快，同时得到的解也是比较优越的解。比如，对于之前迷宫的实验结果深度优先算法、广度优先算法、贪婪最佳优先算法、Astar 算法所运行的步骤分别为 106、78、53、49 由此可以验证上述说法。

而对于贪婪最佳优先算法、Astar 算法而言，我们还可以做以下比较：



贪婪最佳优先算法

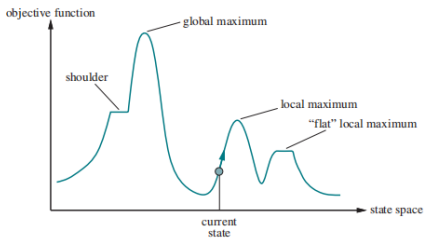


Astar 算法

我们可以发现对于贪婪最佳优先算法而言，可能会出现进入长路径而无法跳出的情况，而 Astar 算法通过引入累积成本则很好的解决了这个问题。

5.2 How the algorithm works

对于爬山算法而言，如果遇到局部最大值、山脊、高原等情况就会陷入局部最优的情况从而无法找到全局最优解。



而模拟退火算法很好的解决了这个问题，因为他不只是去寻找最优解，在一些情况下还会下山去搜索其他情况。

5.3 How the algorithm works

理论上如果可以完美预测对手之后的所有步骤，机器便不会输，由于井字棋是过程较为简单，因此存在必胜法，比如先手开始落子在中间，那么只要正常下棋，基本上只会出现胜利或者和棋的情况，而如果在先手落子在中间的情况下，后手只要落子在棋盘的四个角上，就基本上会出现和棋的情况。因此对于该算法而言，机器只会获胜或者平局，目前还未出现失败的情况。

6 Conclusion

基础：笔者在完成这次大作业之前自学过少量 python 相关知识。

收获：虽然曾经自学过一部分 python，但是苦于没找到合适的项目练手，而通过这次大作业笔者对 python 有了进一步的了解将所学的知识运用于实际，加深了对这些知识的理解，比如变量传参，类等知识。此外通过这次学习笔者还加深了对人工智能的理解，培养了对人工智能的兴趣，学会了部分人工智能相关知识，使笔者受益匪浅。最后在这次研讨中，我还学会了自己探索问题，比如在八皇后问题中，我学会了如何深入了解一个问题，查阅资料，并处理这问题，同时这也是笔者第一次尝试阅读英文文献，这使我收获颇多。

大作业视频：

- **迷宫问题：**
  https://www.bilibili.com/video/BV1bg411d7ti/
- **八皇后问题：**
  https://www.bilibili.com/video/BV1u3411P7Dg/
- **井字棋：**
  https://www.bilibili.com/video/BV1yY4y1z7Xc/

大作业代码：

https://github.com/LinYinRin/-.git

研讨视频：

- https://www.bilibili.com/video/BV1oU4y1m7E9/

References

哈佛大学 CS50 课程《基于 Python 的 AI 入门》课程材料。链接：https://cs50.harvard.edu/ai/2020/

美国，罗素（Stuart J.Russell）、美国，诺维格（Peter Norvig）《人工智能：一种现代的方法》