人工智能基础 LAB1 实验

PB18111699 魏钊

BFS:

```
58  def myBreadthFirstSearch(problem):
59      # YOUR CODE HERE
60      visited = {}
61      frontier = util.Queue()
62
63      frontier.push((problem.getStartState(), None))
64
65      while not frontier.isEmpty():
66          state, prev_state = frontier.pop()
67
68          if problem.isGoalState(state):
69              solution = [state]
70              while prev_state != None:
71                  solution.append(prev_state)
72                  prev_state = visited[prev_state]
73              return solution[::-1]
74
75          if state not in visited:
76              visited[state] = prev_state
77
78              for next_state, step_cost in problem.getChildren(state):
79                  frontier.push((next_state, state))
80      #util.raiseNotDefined()
81      return []
```

只需修改深度优先搜索的代码即可，两者的区别在于，深度优先搜索使用栈存储，广度优先搜索使用队列存储。

AStar:

```
def myAStarSearch(problem, heuristic):
    # YOUR CODE HERE
    frontier = util.PriorityQueue()
    start = [problem.getStartState(), heuristic(problem.getStartState()), []]
    p = 0
    frontier.push(start, p)  # queue push at index_0
    closed = []
    while not frontier.isEmpty():
        [state, cost, path] = frontier.pop()
        # print(state)
        if problem.isGoalState(state):
            # print(path)
            return path+[state]  # here is a deep first algorithm in a sense
        if state not in closed:
            closed.append(state)
            for child_state, child_cost in problem.getChildren(state):
                new_cost = cost + child_cost
                new_path = path + [state]
                frontier.push([child_state, new_cost, new_path], new_cost + heuristic(child_state))
    #util.raiseNotDefined()
    return []
```

AStar 算法使用优先队列存储，start 为初始状态及信息。

将 start 优先度设为 0，压进队列，然后开始循环，判断是否为目标状态，若是，则返回路径；若不是，判断当前状态是否访问过，若没有访问过则存进 closed。

然后将当前状态的后代状态及相关信息压进队列，队列不为空则继续循环。

经过动画演示对比发现：深度优先搜索红色最少，Astar 搜索次之，广度优先搜索最多，即三者搜索过的路径数目。

Minmax：

```
def minimax(self, state, depth):
    if depth==0 or state.isTerminated():
        return None, state.evaluateScore()

    best_state, best_score = None, -float('inf') if state.isMe() else float('inf')

    def Max_s(a,b,c,d):
        if(a>c):
            return a,b
        else:
            return c,d

    def Min_s(a,b,c,d):
        if(a<c):
            return a,b
        else:
            return c,d
```

当前深度为 0 或已经为终止状态时返回。

定义了 Max_s 和 Min_s 用于比较分数大小，返回分数及相关状态。

```
for child in state.getChildren():
    # YOUR CODE HERE
    #util.raiseNotDefined()
    if state.isMe():
        ghost,min_score=self.minimax(child,depth)
        best_score,best_state=Max_s(best_score,best_state,min_score,child)
    elif child.isMe():
        agent,max_score=self.minimax(child,depth-1)
        best_score,best_state=Min_s(best_score,best_state,max_score,child)
    else:
        ghost,min_score=self.minimax(child,depth)
        best_score,best_state=Min_s(best_score,best_state,min_score,child)
return best_state, best_score
```

首先判断状态是否为 Agent，若是，则其孩子为 Ghost，需要从孩子分数（最小分）中选出最大的。

若当前状态不是 Agent（即 Ghost）需判断其孩子状态是否为 Agent,若是，则需要从孩

子分数（最大分）中选出最小。(此时需深度减一)

最后，当前状态和孩子状态均为 Ghost，需从最小分数中选出最小。

AlphaBeta：

```
def minimax(self, state, depth,a,b):
    if depth==0 or state.isTerminated():
        return None, state.evaluateScore()

    best_state, best_score = None, -float('inf') if state.isMe() else float('inf')

    def Max_s(a,b,c,d):
        if(a>c):
            return a,b
        else:
            return c,d

    def Min_s(a,b,c,d):
        if(a<c):
            return a,b
        else:
            return c,d
```

首先和 Minmax 类似的，定义函数并加上端点值 a,b。

```
for child in state.getChildren():
    # YOUR CODE HERE
    #util.raiseNotDefined()
    if state.isMe():
        ghost,min_score=self.minimax(child,depth,a,b)
        best_score,best_state=Max_s(best_score,best_state,min_score,child)
        if best_score > b:
            return best_state, best_score
        a = max(a, best_score)
    elif child.isMe():
        agent,max_score=self.minimax(child,depth-1,a,b)
        best_score,best_state=Min_s(best_score,best_state,max_score,child)
        if best_score < a:
            return best_state, best_score
        b = min(b, best_score)
    else:
        ghost,min_score=self.minimax(child,depth,a,b)
        best_score,best_state=Min_s(best_score,best_state,min_score,child)
        if best_score < a:
            return best_state, best_score
        b = min(b, best_score)
return best_state, best_score
```

大部分代码和 Minmax 相同，还有加上每次的剪枝操作，如上图有三种，当复合 if 的剪枝条件时，返回，否则修改端点值。