人 工 智 能 实 验 报 告

实验名称： 博弈搜索

学员姓名: 程景愉

学　号: 202302723005

实验日期: 2025.12.02

国防科技大学教育训练部制

《本科实验报告》填写说明

实验报告内容编排应符合以下要求：

（1）采用 A4（21cm×29.7cm）白色复印纸，单面黑字。上下左右各侧的页边距均为 3cm；缺省文档网格：字号为小 4 号，中文为宋体，英文和阿拉伯数字为 Times New Roman，每页 30 行，每行 36 字；页脚距边界为 2.5cm，页码置于页脚、居中，采用小 5 号阿拉伯数字从 1 开始连续编排，封面不编页码。

（2）报告正文最多可设四级标题，字体均为黑体，第一级标题字号为 4 号，其余各级标题为小 4 号；标题序号第一级用“一、”、“二、”……，第二级用“（一）”、“（二）”……，第三级用“1.”、“2.”……，第四级用“（1）”、“（2）”……，分别按序连续编排。

（3）正文插图、表格中的文字字号均为 5 号。

0 目录

1 实验介绍

本项目是UC Berkeley CS188人工智能课程的第二个项目，重点在于对抗搜索（Adversarial Search）。在此项目中，我们将编写控制Pacman的智能体，使其能够在不仅有静止障碍物和食物，还有主动追捕Pacman的幽灵（Ghosts）的环境中生存并获胜。实验涉及的核心算法包括Reflex Agent（反射智能体）、Minimax（极大极小算法）、Alpha-Beta剪枝以及Expectimax（期望最大算法）。此外，还需要设计高效的评估函数来指导智能体在有限深度的搜索中做出最优决策。通过本实验，我们将深入理解博弈树搜索、剪枝优化以及非完全理性对手建模等关键概念。

2 实验内容

本次实验内容涵盖了5个核心编程任务，具体如下：

1. **Q1: Reflex Agent**：实现一个能够根据当前状态特征（如食物距离、幽灵位置）做出即时反应的反射智能体。
2. **Q2: Minimax**：实现通用的Minimax算法，使Pacman能够假设幽灵采取最优策略（即尽力让Pacman得分最低）的情况下，规划出最优路径。
3. **Q3: Alpha-Beta Pruning**：在Minimax的基础上引入Alpha-Beta剪枝，以减少不必要的节点扩展，提高搜索效率，从而支持更深层次的搜索。
4. **Q4: Expectimax**：实现Expectimax算法，不再假设幽灵是最优对手，而是将其建模为随机行动的对手，通过计算期望得分来最大化收益。
5. **Q5: Evaluation Function**：设计一个更强大的评估函数，综合考虑多种游戏状态特征，使智能体在搜索深度有限时仍能准确评估局面好坏。

3 实验要求

本项目要求在`multiAgents.py`文件中根据指引补全代码，核心要求如下：

1. 文件修改：仅允许修改`multiAgents.py`，其他文件（如`pacman.py`, `game.py`）不得修改，以确保与自动评分器的兼容性。
2. 算法通用性：实现的Minimax和Alpha-Beta算法必须支持任意数量的幽灵（Min层）和任意指定的搜索深度。
3. 剪枝正确性：在Q3中，必须正确实现Alpha-Beta剪枝逻辑，不能改变节点的遍历顺序（必须按照`getLegalActions`返回的顺序），且不能进行基于等值的剪枝，以完全匹配自动评分器的扩展节点数检查。
4. 性能指标：对于Q1和Q5，设计的评估函数必须使Pacman在测试关卡中达到一定的胜率（如Q1需赢得5次以上，Q5需在smallClassic上赢得全部10次且均分超过1000），且运行时间需在规定范围内。

4 实验步骤与实现

4.1 Q1: Reflex Agent (反射智能体)

实现思路：Reflex Agent 不进行树搜索，而是基于当前状态的直接评估来选择动作。我们在 `evaluationFunction` 中设计了一个简单的线性组合评分系统。主要考虑两个因素：一是离食物越近越好（使用距离倒数作为激励）；二是如果幽灵太近且未受惊，则必须避开（给予极大的惩罚）。

核心代码 (`multiAgents.py`):

```
def evaluationFunction(self, currentGameState: GameState, action):
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

    # 计算到最近食物的距离
    foodList = newFood.asList()
    if not foodList:
        return 999999
    minFoodDist = min([util.manhattanDistance(newPos, food) for food in
foodList])

    # 避免危险幽灵
    for i, ghostState in enumerate(newGhostStates):
        dist = util.manhattanDistance(newPos, ghostState.getPosition())
        if dist < 2 and newScaredTimes[i] == 0:
            return -999999

    # 综合评分：基础分 + 食物激励
    return successorGameState.getScore() + 1.0 / (minFoodDist + 1)
```

测试指令:

```
python autograder.py -q q1
python pacman.py -p ReflexAgent -l testClassic
```

4.2 Q2: Minimax (极大极小算法)

实现思路：Minimax 算法构建一棵博弈树，Pacman 作为 MAX 层试图最大化得分，而所有幽灵作为 MIN 层试图最小化 Pacman 的得分。每一层递归代表一个 Agent 的行动。当轮到 Pacman（Agent 0）时，深度加 1（实际上是所有 Agent 行动一轮算一层深度，这里实现时通常在回到 Agent 0 时增加深度计数）。

核心代码 (`multiAgents.py`):

```
def minimax(agentIndex, depth, gameState):
    if gameState.isWin() or gameState.isLose() or depth == self.depth:
        return self.evaluationFunction(gameState)

    # MAX Agent (Pacman)
    if agentIndex == 0:
        bestValue = float("-inf")
```

```
        for action in gameState.getLegalActions(agentIndex):
            succ = gameState.generateSuccessor(agentIndex, action)
            bestValue = max(bestValue, minimax(agentIndex + 1, depth, succ))
        return bestValue
    # MIN Agents (Ghosts)
    else:
        bestValue = float("inf")
        for action in gameState.getLegalActions(agentIndex):
            succ = gameState.generateSuccessor(agentIndex, action)
            if agentIndex == gameState.getNumAgents() - 1:
                bestValue = min(bestValue, minimax(0, depth + 1, succ))
            else:
                bestValue = min(bestValue, minimax(agentIndex + 1, depth, succ))
        return bestValue
```

测试指令:

```
python autograder.py -q q2
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
```

4.3 Q3: Alpha-Beta Pruning (Alpha-Beta 剪枝)

实现思路: 在 Minimax 的基础上维护两个变量 α (MAX 节点的下界) 和 β (MIN 节点的上界)。在 MAX 节点，如果发现某分支的值大于 $b\eta$，则剪枝（因为 MIN 父节点不会选它）；在 MIN 节点，如果某分支值小于 α，则剪枝（MAX 父节点不会选它）。

核心代码 (`multiAgents.py`):

```
def maxValue(agentIndex, depth, gameState, alpha, beta):
    v = float("-inf")
    for action in gameState.getLegalActions(agentIndex):
        succ = gameState.generateSuccessor(agentIndex, action)
        v = max(v, alphaBeta(agentIndex + 1, depth, succ, alpha, beta))
        if v > beta: return v # Pruning
        alpha = max(alpha, v)
    return v

def minValue(agentIndex, depth, gameState, alpha, beta):
    v = float("inf")
    for action in gameState.getLegalActions(agentIndex):
        succ = gameState.generateSuccessor(agentIndex, action)
        if agentIndex == gameState.getNumAgents() - 1:
            v = min(v, alphaBeta(0, depth + 1, succ, alpha, beta))
        else:
            v = min(v, alphaBeta(agentIndex + 1, depth, succ, alpha, beta))
        if v < alpha: return v # Pruning
        beta = min(beta, v)
    return v
```

测试指令:

```
python autograder.py -q q3
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

4.4 Q4: Expectimax (期望最大算法)

实现思路：Expectimax 不再假设幽灵是最优对手，而是假设它们随机行动（Uniform Random）。因此，Min 层变成了 Chance 层，计算所有后续状态得分的平均值（期望值）。这使得 Pacman 在面对非最优幽灵时敢于承担一定风险去获取更高收益。

核心代码 (multiAgents.py):

```
def expValue(agentIndex, depth, gameState):
    v = 0
    actions = gameState.getLegalActions(agentIndex)
    prob = 1.0 / len(actions)
    for action in actions:
        succ = gameState.generateSuccessor(agentIndex, action)
        if agentIndex == gameState.getNumAgents() - 1:
            v += prob * expectimax(0, depth + 1, succ)
        else:
            v += prob * expectimax(agentIndex + 1, depth, succ)
    return v
```

测试指令:

```
python autograder.py -q q4
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3
```

4.5 Q5: Evaluation Function (评估函数)

实现思路：为了在有限深度搜索中获得更好表现，betterEvaluationFunction 考虑了更多特征。除了基础分数，我们增加了对最近食物的奖励（权重+10），对胶囊的奖励（权重+20），以及对受惊幽灵的追捕奖励（权重+100）。同时，对靠近非受惊幽灵给予重罚（权重-1000），并根据剩余食物和胶囊的数量给予惩罚，以激励 Pacman 尽快清空地图。

核心代码 (multiAgents.py):

```
def betterEvaluationFunction(currentGameState: GameState):
    pos = currentGameState.getPacmanPosition()
    score = currentGameState.getScore()

    # 距离特征
    foodDist = min([util.manhattanDistance(pos, f) for f in foodList]) if
foodList else 0
    score += 10.0 / (foodDist + 1)

    # 幽灵特征
    for i, ghost in enumerate(ghostStates):
        dist = util.manhattanDistance(pos, ghost.getPosition())
        if scaredTimes[i] > 0:
            score += 100.0 / (dist + 1) # 鼓励吃受惊幽灵
        elif dist < 2:
            score -= 1000.0 # 极力避免接触

    # 数量特征（越少越好，故减去）
    score -= len(foodList) * 4
    score -= len(capsules) * 20
```

```
    return score
```

测试指令:

```
python autograder.py -q q5
python autograder.py -q q5 --no-graphics
```

5 实验结果

本项目的所有 5 个任务均已成功实现, 并通过了自动评分器（autograder）的所有测试用例, 最终取得了 **25/25** 的满分成绩。

```
Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 5/5
Question q5: 6/6
------------------
Total: 25/25
```

主要测试结果分析:

1. **Q1 (Reflex)**: 智能体能够有效避开幽灵并吃到食物，在 `testClassic` 上表现稳定，平均分超过 1000。
2. **Q2 (Minimax)**: 成功通过了所有深度和幽灵数量的博弈树测试，扩展节点数与标准答案完全一致，证明了逻辑的正确性。
3. **Q3 (Alpha-Beta)**: 在保持结果与 Minimax 一致的前提下，显著减少了扩展的节点数量。在 `smallClassic` 上，深度为 3 的搜索能够在极短时间内完成。
4. **Q5 (Eval)**: 设计的评估函数表现优异，Pacman 在 `smallClassic` 的 10 次随机测试中全部获胜，且平均得分远超 1000 分的要求，证明了特征选择和权重分配的合理性。

6 实验总结

通过本次实验，我深入掌握了对抗搜索算法的核心思想及其在多智能体环境中的应用。

首先，我实现了 Minimax 算法，理解了零和博弈中 MAX 与 MIN 节点的相互制约关系。随后，通过实现 Alpha-Beta 剪枝，我深刻体会到了剪枝技术对于提升搜索效率的重要性——它能够在不改变最终决策的前提下，指数级地减少搜索空间，使得更深层的搜索成为可能。

其次，Expectimax 算法的实现让我认识到，在面对随机或非最优对手时，概率模型往往比悲观的 Minimax 模型更具优势。最后，也是最具挑战性的部分，是设计评估函数。我学会了如何将游戏状态抽象为特征向量（如食物距离、幽灵状态等），并通过线性组合这些特征来量化状态的优劣。这不仅考验了代码实现能力，更考验了对问题本质的理解和特征工程的能力。