人 工 智 能 实 验 报 告

实验名称：Pacman 实验

姓　　名：	黄结伦	学　　号：	202002422003
指导教员：	李莎莎	职　　称：	研究员
实 验 室：	102-201	实验日期：	2019.11-2020.1

国防科学技术大学训练部制

Pacman 实验2

一.实验目的

1.通过实验理解零和游戏的博弈过程，学会简单的评估函数的设计

2.掌握极小极大法和alpha-beta剪枝

二.实验原理

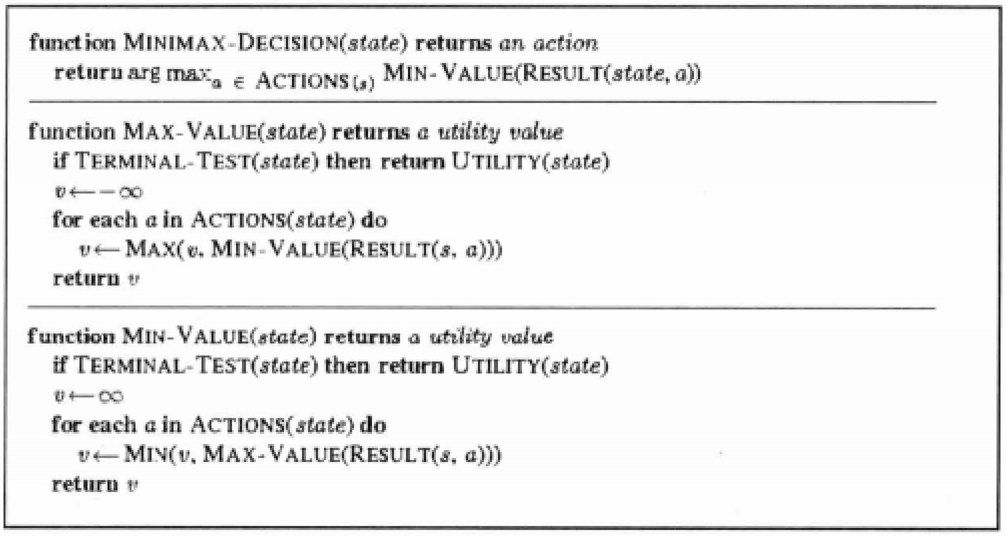
1.极小极大法

在零和博弈中，玩家均会在可选的选项中选择将其N步后优势最大化或者令对手优势最小化的选择。将双方决策过程视作一颗决策树，若决策树某一层均为己方决策依据状态（即接下来是己方进行动作），则己方必定会选择使得己方收益最大化的路径，将该层称为MAX层。若决策树某一层均为对手决策依据状态（即接下来是对手进行动作），则对手必定会选择使得己方收益最小化的路径，将该层成为MIN层。由此，一个极小极大决策树将包含max节点（MAX层中的节点）、min节点（MIN层中的节点）和终止节点（博弈终止状态节点或N步时的状态节点）。每个节点对应的预期收益成为该节点的minimax值。
对于终止结点， minimax值等于直接对局面的估值。对于max结点，由于max节点所选择的动作将会由己方给定，因此选择minimax值最大的子结点的值作为max结点的值。对于min结点，则选择minimax值最小的子结点的值作为min结点的值。
极小极大算法过程可描述如下：

构建决策树；
1.将评估函数应用于叶子结点；
2.自底向上计算每个结点的minimax值；
3.从根结点选择minimax值最大的分支，作为行动策略。
minimax计算流程如下：

1.如果节点是终止节点：应用估值函数求值；
2.如果节点是max节点：找到每个子节点的值，将其中最大的子节点值作为该节点的值；
3.如果节点时min节点：找到每个子节点的值，将其中最小的子节点值作为该节点的值。

其伪代码如下，



```
function MINIMAX-DECISION(state) returns an action
    return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a)))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

2.alpah-beta剪枝

举个简单的例子说明**alpha-beta剪枝**的过程

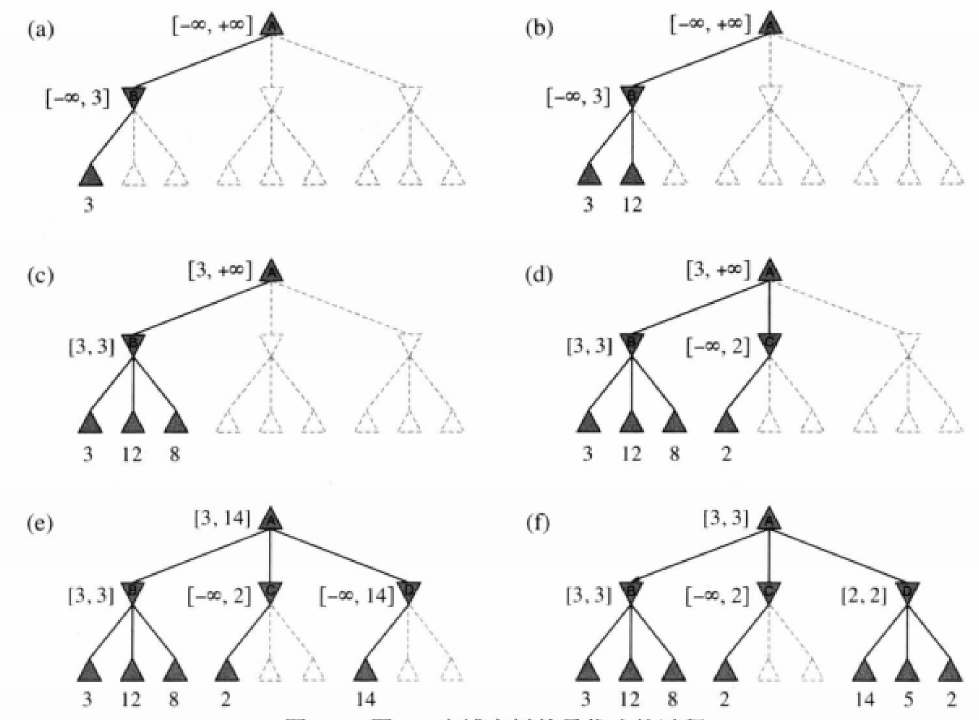
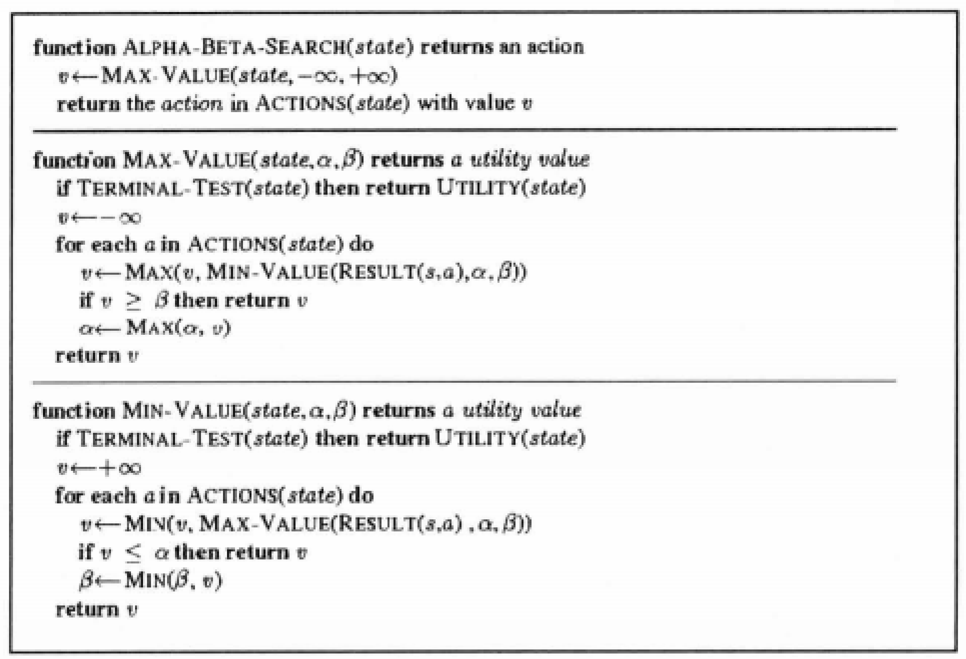


图 5.5　图 5.2 中博弈树的最优决策过程

每一结点上标出了可能的取值范围。（a）*B* 下面的第一个叶结点值为 3。因此作为 MIN 结点的 *B* 值至多为 3。（b）*B* 下面的第二个叶结点值为 12。MIN 不会用这招，所以 *B* 的值仍然至多为 3。（c）*B* 下面的第三个叶子值为 8；此时已经观察了 *B* 的所有后继，所以 *B* 的值就是 3。现在可以推断根结点的值至少为 3，因为 MAX 在根结点有值为 3 的后继。（d）*C* 下面的第一个叶结点值为 2。因此 *C* 这个 MIN 结点的值至多为 2。不过已经知道 *B* 的值是 3，所以 MAX 不会选择 *C*。这时再考察 *C* 的其他后继已经没有意义了。这就是α-β剪枝的实例。（e）*D* 下面的第一个叶结点值为 14，所以 *D* 的值至多为 14。这比 MAX 的最佳选择（即 3）要大，所以继续探索 *D* 的其他后继。还要注意现在知道根的取值范围，根结点的值至多为 14。（f）*D* 的第二个后继值为 5，所以我们又必须继续探索。第三个后继值为 2，所以 *D* 的值就是 2 了。最终 MAX 在根结点的决策是走到值为 3 的 *B* 结点

其伪代码如下，



```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

三.实验内容

1.完成multiagent文件夹中的evaluationFunction，MinimaxAgent，AlphaBetaAgent函数。

在编写完代码后，可在终端中的multiagent目录下执行`python2 .\autograder.py`命令，即可查看是否通过！

四.实验结果

博弈算法中的三个函数（类）的具体实现如下

1.evaluationFunction

```
    def evaluationFunction(self, currentGameState, action):
        """
        Design a better evaluation function here.

        The evaluation function takes in the current and proposed successor
        GameStates (pacman.py) and returns a number, where higher numbers are
better.

        The code below extracts some useful information from the state, like the
        remaining food (newFood) and Pacman position after moving (newPos).
        newScaredTimes holds the number of moves that each ghost will remain
        scared because of Pacman having eaten a power pellet.

        Print out these variables to see what you're getting, then combine them
        to create a masterful evaluation function.
        """
        # Useful information you can extract from a GameState (pacman.py)
        successorGameState = currentGameState.generatePacmanSuccessor(action)
        newPos = successorGameState.getPacmanPosition()
        newFood = successorGameState.getFood()
        newGhostStates = successorGameState.getGhostStates()
        newScaredTimes = [ghostState.scaredTimer for ghostState in
newGhostStates]
        #many ghosts
        "*** YOUR CODE HERE ***"
        GhostPos = successorGameState.getGhostPositions()
        x_pacman,y_pacman = newPos
        failedDist = min([(abs(each[0]- x_pacman) + abs(each[1]-y_pacman)) for
each in GhostPos])
        if failedDist != 0 and failedDist < 4:
            ghostScore = -11 / failedDist
        else :
            ghostScore = 0
        nearestFood = float('inf')
        width = newFood.width
        height = newFood.height
        if failedDist >= 2:
            dx = [1,0,-1,0]
            dy = [0,1,0,-1]
            List = []
            d = {}
            List.append(newPos)
            d.update({(x_pacman,y_pacman) : 1})
            while List:
                tempPos = List[0]
                List.pop(0)
                temp_x,temp_y = tempPos
                if newFood[temp_x][temp_y]:
                    nearestFood = min(nearestFood,(abs(temp_x - x_pacman) +
abs(temp_y - y_pacman)))
                    break
                for i in range(len(dx)):
                    x = temp_x + dx[i]
                    y = temp_y + dy[i]
                    if 0 <= x < width and 0 <= y < height:
                        tempPos =(x,y)
                        if tempPos not in d:
                            d[tempPos] = 1
                            List.append(tempPos)
        if nearestFood != float('inf'):
            foodScore = 10 / nearestFood
        else :
            foodScore = 0
        return successorGameState.getScore() + foodScore + ghostScore
```

2.MinimaxAgent

```
class MinimaxAgent(MultiAgentSearchAgent):
    """
    Your minimax agent (question 2)
    """
    def getAction(self, gameState):
        """
        Returns the minimax action from the current gameState using self.depth
        and self.evaluationFunction.

        Here are some method calls that might be useful when implementing
minimax.

        gameState.getLegalActions(agentIndex):
        Returns a list of legal actions for an agent
        agentIndex=0 means Pacman, ghosts are >= 1

        gameState.generateSuccessor(agentIndex, action):
        Returns the successor game state after an agent takes an action

        gameState.getNumAgents():
        Returns the total number of agents in the game

        gameState.isWin():
        Returns whether or not the game state is a winning state

        gameState.isLose():
        Returns whether or not the game state is a losing state
        """
        "*** YOUR CODE HERE ***"

        def gameOver(gameState):
            return gameState.isWin() or gameState.isLose()
        #Be different with me
        def min_value(gameState, depth, ghost):
            value = float('inf')
            if gameOver(gameState):
                return self.evaluationFunction(gameState)
            for action in gameState.getLegalActions(ghost):
                if ghost == gameState.getNumAgents() - 1:
                    value = min(value,
max_value(gameState.generateSuccessor(ghost, action), depth))
                else:
                    value = min(value,
min_value(gameState.generateSuccessor(ghost, action), depth,
                                                 ghost + 1))
            return value


        def max_value(gameState, depth):
            value = float('-inf')
            depth = depth + 1
            #Be different with me
            if depth == self.depth or gameOver(gameState):
                return self.evaluationFunction(gameState)
            for action in gameState.getLegalActions(0):
                value = max(value, min_value(gameState.generateSuccessor(0,
action), depth, 1))
            return value
        nextAction = gameState.getLegalActions(0)
        Max = float('-inf')
        Result = None

        for action in nextAction:
            if (action != "stop"):
                depth = 0
                value = min_value(gameState.generateSuccessor(0, action), depth,
1)
                if (value > Max):
                    Max = value
                    Result = action
        return Result
        util.raiseNotDefined()
```

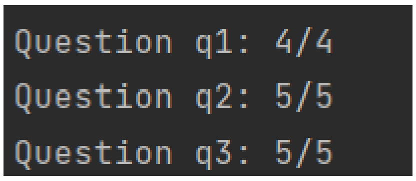
3.AlphaBetaAgent

```
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
      Your minimax agent with alpha-beta pruning (question 3)
    """

    def getAction(self, gameState):
        """
          Returns the minimax action using self.depth and
self.evaluationFunction
        """
        "*** YOUR CODE HERE ***"
        ghostIndex = [i for i in range(1,gameState.getNumAgents())]
        def gameOver(state,depth):
            return state.isWin() or state.isLose() or depth == self.depth
        def min_value(state,depth,ghost,alpha,beta):
            if gameOver(state,depth):
                return self.evaluationFunction(state)
            value = float('inf')
            for action in state.getLegalActions(ghost):
                if ghost == ghostIndex[-1]:
                    value = 
min(value,max_value(state.generateSuccessor(ghost,action),depth+1,alpha,beta))
                else:
                    value = 
min(value,min_value(state.generateSuccessor(ghost,action),depth,ghost+1,alpha,be
ta))
                if value < alpha:
                    return value
                beta = min(beta,value)
            return value
        def max_value(state,depth,alpha,beta):
            if gameOver(state,depth):
                return self.evaluationFunction(state)
            value = float('-inf')
            for action in state.getLegalActions(0):
                if action == 'stop':
                    continue
                value = 
max(value,min_value(state.generateSuccessor(0,action),depth,1,alpha,beta))
                if value > beta:
                    return value
                alpha = max(value,alpha)
            return value
        def function(state):
            value = float('-inf')
            actions = None
            alpha = float('-inf')
            beta = float('inf')
            for action in state.getLegalActions(0):
                if action == 'stop':
                    continue
                tmpValue = 
min_value(state.generateSuccessor(0,action),0,1,alpha,beta)
                if value < tmpValue:
                    value = tmpValue
                    actions = action
                alpha = max(value,alpha)
            return actions
        return function(gameState)
        util.raiseNotDefined()
```

测试过程与结果如下

```
PS D:\AI\Subject\Pacman\multiagent\multiagent> python2 .\autograder.py
Starting on 10-22 at 15:37:54
```



```
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
```

五.实验总结

博弈算法在AI课程中第一次接触，虽然博弈过程是一个十分玄妙的过程，但是计算机人用其大智慧，将这个过程转化为一个可以用数学公式描述的过程，而这种抽象化的思维能力是一个及其重要的能力。通过Pacman的实验，加深了我对alpha-beta剪枝的原理与过程。