COEN 266 ARTIFICIAL INTELLIGENCE
GROUP ASSIGNMENT 2

NAME: PAURAVI NAGARKAR (1650209): 50%
NITYANAND PUJARI (1650422) : 50%

Problem : Implement Minimax Function: For Pacman and Ghosts

```
class MinimaxAgent(MultiAgentSearchAgent):
    """
    Your minimax agent (question 2)
    """

    def getAction(self, gameState):

        def max_Player(gameState, depth):
            currDepth = depth + 1
            if gameState.isWin() or gameState.isLose() or currDepth == self.depth:
                return self.evaluationFunction(gameState)
            maxvalue = -float('inf')
            actions = gameState.getLegalActions(0)
            for action in actions:
                nextState = gameState.generateSuccessor(0, action)
                maxvalue = max(maxvalue, min_Player(nextState, currDepth, 1))
            return maxvalue

        # For all ghosts.
        def min_Player(gameState, depth, agentIndex):
            minvalue = float('inf')
            if gameState.isWin() or gameState.isLose():
                return self.evaluationFunction(gameState)
            actions = gameState.getLegalActions(agentIndex)
            for action in actions:
                nextState = gameState.generateSuccessor(agentIndex, action)
                if agentIndex == (gameState.getNumAgents() - 1):
                    minvalue = min(minvalue, max_Player(nextState, depth))
                else:
                    minvalue = min(minvalue, min_Player(nextState, depth, agentIndex + 1))
            return minvalue

        # Root level action.
        actions = gameState.getLegalActions(0)
        root_node_value = -float('inf')
        best_Action = ''
        for action in actions:
            nextLevel = gameState.generateSuccessor(0, action)
            # Next level is a min level. Hence calling min for successors of the root.
            score = min_Player(nextLevel, 0, 1)
            # Choosing the action which is Maximum of the successors.
            if score > root_node_value: best_Action, root_node_value = action, score
            print(root_node_value)
        return best_Action
```

Implementation steps for Minimax Function:

1. Define the game state: Create a data structure to represent the game state, including possible moves and rules.
2. Implement an evaluation function: Create a function that assigns a score to each game state, indicating how good it is for the current player.
3. Implement the Minimax algorithm: Use a recursive algorithm to search through the game tree and find the best move for the current player. The algorithm should alternate between maximizing and minimizing the score until it reaches the maximum search depth or a terminal state.
4. Specify the search depth: Decide on the maximum number of levels the algorithm should search in the game tree. A larger search depth may result in better moves but may take longer to compute.
5. Select the first move: Once the Minimax algorithm has finished searching the game tree, choose the move with the best score for the current player and update the game state.
6. Repeat until the game is over: Continue repeating steps 2-5 until the game is over. If necessary, switch between players.
7. Handle edge cases: Ensure that the program can handle any edge cases that may arise, such as tie games or unsolvable game states. The program should also be able to handle any errors that may occur during gameplay.

Comment:

- The goal is to write code for an Adversarial Search Agent using the Minimax algorithm.
- In this game, Pacman(max_Player) acts as the Maximizing Agent, while the Ghosts(min_Player) act as the Minimizing agents.
- The functions provided include those to evaluate scores at terminal states, legal actions at any node, and various other utility functions. The Minimax algorithm is implemented using two functions - 'max_Player' and 'min_Player'.
- At any given depth level, Pacman's(max_Player) agent index is 0, while the Ghosts(min_Player) have indices greater than one.
- The best action at the root level, which is the action to be taken by Pacman(max_Player), is returned, if the Ghosts(min_Player) are playing optimally and based on the score or Node's Value at that level.

have indices greater than one.

- The best action at the root level, which is the action to be taken by Pacman(max_Player), is returned, if the Ghosts(min_Player) are playing optimally and based on the score or Node's Value at that level.

The code for the Pacman i.e., at the root level is:

```
actions = gameState.getLegalActions(0)
root_node_value = -float('inf')
best_Action = ''
for action in actions:
    nextLevel = gameState.generateSuccessor(0, action)
    score = min_Player(nextLevel, 0, 1)
       if score > root_node_value: best_Action, root_node_value = action, score
    print(root_node_value)
return best_Action
```

- The code starts by identifying the legal actions available for Pacman(max_Player) at the current gamestate.
- For each of these actions, the code finds the corresponding next_State nodes, which will be the Ghosts(min_Player) (i.e., Min nodes).
- The Minimax functions (min_Player() and max_Player()) are then called to calculate the score at each of these successor nodes.
- These scores are then used by Pacman at the root level to determine which action will result in the maximum score.
- The action that gives the maximum score is returned as the best_Action in the code.

The Minimizing agent's code for Ghosts (min_Ghosts):

```
def min_Player(gameState, depth, agentIndex):
  minvalue = float('inf')
  if gameState.isWin() or gameState.isLose():
     return self.evaluationFunction(gameState)
  actions = gameState.getLegalActions(agentIndex)
  for action in actions:
     nextState = gameState.generateSuccessor(agentIndex, action)
     if agentIndex == (gameState.getNumAgents() - 1):
        minvalue = min(minvalue, max_Player(nextState, depth))
     else:
        minvalue = min(minvalue, min_Player(nextState, depth, agentIndex + 1))
  return minvalue
```

- To check the terminal layer, the code uses the isWin() and isLose() functions. If the current layer is the terminal layer, the code returns the values using the self.evaluationFunctions().
- If the current layer is not the terminal layer, the code finds the legal actions for that node and generates the corresponding successors.
- If the layer below this is Pacman's layer, the code finds the minimum value of the various nodes using the max_Player(next_State, depth) function.
- If the next layer is the Ghost layer, the code finds the minimum value of the layer below it.

```
minvalue = min(minvalue,min_Player(next_State,depth,agentIndex+1))
```

Similarly for the Pacman node layer - the maximum value at its Next levels is found in the max_Pacman function.

```
maxvalue = max(maxvalue,min_Player(next_State,currDepth,1))
```

The following is then run in the command prompt to get the output as below.

```
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 --frameTime 0
```

Result:

Using MiniMax algorithms for Pacman was designed and implemented.