COEN 166 Artificial Intelligence

Lab 4

Divya Syal: 1594122, Marianne Yamazaki Dorr: 1606975: , Suvass Ravala: 1593088

Lab 4 Contributions of each group member (in percentage): Each did 33%

Task: Minimax Agent

paste your code:

```
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 ***"
```

```
"""
this function will
• recursively call a “max” function and a “min” function.
• propagate the “leaf” node values to upper layers, until the root node is reached.
• finally, return the best action for player-max (Pacman) at the root node.
"""
if (self.depth == 0) or (gameState.isWin()) or (gameState.isLose()):
    return self.scoreEvaluationFunction(gameState)

agentIndex = 0 # start with pacman at root first always
initialDepth = 1 # for first 1-ply game



# index should be 0 for pacman/root
def max_value (currentState, index, depth):
    # store all possible game states
    valueList=[]
    # grab all possible pacman actions
    actionList=currentState.getLegalActions(index)

    # if empty list of actions
    if not actionList:
        # no more actions, at terminal state, return payoff
        return scoreEvaluationFunction(currentState)

    # for each action (if they exist)
    for action in actionList:
        # get each next state
        nextState = currentState.generateSuccessor(0, action)
        # find minimum value for next state and append to list
        # next index will always be 1 from ghost, but may have multiple pacmans
        value = min_value(nextState, index+1, depth)
        valueList.append(value)

    # return score when at depth 1/root/first pacman
    if depth == 1:
        # return action with highest value
        bestValueIndex = valueList.index(max(valueList))
        bestAction = actionList[bestValueIndex]
        return bestAction
```

```
            # if not at depth 1, return max (going to another pacman)
            # if at second pacman
            else:
                return max(valueList)

        # index: 0 is pacman, where 1 to n is ghost
        # depth is current level, go until it = self.depth
        def min_value(state, index, depth):

            valueList = []
            actionList = state.getLegalActions(index)

            if not actionList:
                return scoreEvaluationFunction(state)

            if (index == (state.getNumAgents()-1)):
                #ghost 3 or last one's turn
                for action in actionList:
                    nextState = state.generateSuccessor(index, action)

                    # if at a leaf/terminal state
                    if depth == self.depth:
                        # ghost 3
                        value = scoreEvaluationFunction(nextState)
                        valueList.append(value)

                    # otherwise, there must be another pacman/ply underneath repeat call
max
                    else:

                        value = max_value(nextState, 0, depth+1)
                        valueList.append(value)
            else:
                # ghost 1 or 2 or n-1
                # generate successors and pick min recursively
                for action in actionList:
                    nextState = state.generateSuccessor(index, action)
                    value = min_value(nextState, index+1, depth)
                    valueList.append(value)

            return min(valueList)


        # pacman plays first, so find max
        return max_value(gameState, agentIndex, initialDepth)
```

```
class MinimaxAgent(MultiAgentSearchAgent):
        def getAction(self, gameState):
                …
                # inside the getAction function, you will define the following two functions:
                def maxValue_fun(state,PlayerIndex, other arguments if needed):
                …


                def minValue_fun(state, PlayerIndex, other arguments if needed):
                ...
                # commands in the getAction function will call the maxValue_fun function,
        which
                # will then return the best action for the root node
                ...
```

Function 1: max_value

paste your code (any other function that you defined or modified): def function_name(arguments):

```
# index should be 0 for pacman/root
      def max_value (currentState, index, depth):
         # store all possible game states
         valueList=[]
         # grab all possible pacman actions
         actionList=currentState.getLegalActions(index)

         # if empty list of actions
         if not actionList:
            # no more actions, at terminal state, return payoff
            return scoreEvaluationFunction(currentState)

         # for each action (if they exist)
         for action in actionList:
            # get each next state
            nextState = currentState.generateSuccessor(0, action)
            # find minimum value for next state and append to list
            # next index will always be 1 from ghost, but may have multiple pacmans
            value = min_value(nextState, index+1, depth)
            valueList.append(value)

         # return score when at depth 1/root/first pacman
         if depth == 1:
            # return action with highest value
            bestValueIndex = valueList.index(max(valueList))
            bestAction = actionList[bestValueIndex]
            return bestAction


         # if not at depth 1, return max (going to another pacman)
         # if at second pacman
         else:
            return max(valueList)
```

...

Comment: explain how you modified/developed the above function/code... …

In this function, we fully implemented the max value function, first we created a valueList array that would store all the possible states, then we would get all the possible actions that PacMan would take. Then we would check if it's the final state and return the score associated with it. If not, then we run through all the actions and get the nextState, then call the min_value function and append it. Since we would need to return the value if it were at depth 1, we would check for it and return the bestAction.

Function 2: min_value

paste your code (any other function that you defined or modified): def function_name(arguments):

```
# index: 0 is pacman, where 1 to n is ghost
      # depth is current level, go until it = self.depth
      def min_value(state, index, depth):

         valueList = []
         actionList = state.getLegalActions(index)

         if not actionList:
            return scoreEvaluationFunction(state)

         if (index == (state.getNumAgents()-1)):
            #ghost 3 or last one's turn
            for action in actionList:
               nextState = state.generateSuccessor(index, action)

               # if at a leaf/terminal state
               if depth == self.depth:
                  # ghost 3
                  value = scoreEvaluationFunction(nextState)
                  valueList.append(value)

               # otherwise, there must be another pacman/ply underneath repeat call
max
               else:

                  value = max_value(nextState, 0, depth+1)
                  valueList.append(value)
         else:
            # ghost 1 or 2 or n-1
            # generate successors and pick min recursively
            for action in actionList:
               nextState = state.generateSuccessor(index, action)
               value = min_value(nextState, index+1, depth)
               valueList.append(value)

         return min(valueList)
```

...

Comment: explain how you modified/developed the above function/code... …

In this function, we would start off by doing similar commands as the max value by creating a valueList and return the evaluation if there aren't any actions. Then we would check if we were at the last ghost, if we were then we would get the nextState and check if it's a leaf node and get the score. If it's not then we would run the max function and append the value to it. If it's the first 2 ghosts or n-1, then we would generate the nextState and find the min value.

Questions

1. How does the “max” function work?
    a. The “max” function will first create a list to store all the possible game states in valueList. Next, it will grab all the possible actions that Pacman can take. If the list of actions is empty, the function will simply return the payoff at the current state. Otherwise, the function will iterate through the action list and for each action, it will generate the next state based on the current state and action. It will then call the “min” function by passing the next state to the ghosts and store the value returned from ghosts’ actions. It will then store that value into valueList for each action.
    Next, the “max” function will check if we have reached the last depth level we want to reach, if we have, then the function will return the action associated with the highest value since we want to maximize Pacman’s score. Otherwise, we are not at the last depth and are going to another Pacman and so we return the max element of the valueList which will be returned to the min_value function.

2. How does the “min” function work?
    a. The “min” function will also first create a list to store all possible game states in valueList. It will then get all the possible actions of the current ghost. If the list of actions is empty, then the function will simply return the payoff at the current state.
    Otherwise, the function will then check if we’re at the last ghost. If we are, then the function will generate the next states for every action. It will also check if we are at the last depth level we want to reach, if so, then the function will add the payoff of the next state to the valueList. Otherwise, it will call the “max” function to go another depth level down into the next Pacman. It will append the value received from the “max” function to the valueList.
    If we are not at the last ghost, then the function will iterate through the possible actions and generate the next state for each action. It will then call the next ghost by recursively calling the “min” function and pass it the next state. It will then store the value returned from the recursive call to the valueList.
    Finally, the function will return the minimum element from the valueList since the ghosts seek to minimize the payoff.

3. How to recursively call the “max” function and “min” function?
    a. Until the desired depth has been reached, the max function will call the min function as ghost 1 which in turn will call itself twice for ghost 2 and 3

respectively. If we're not at the final depth, then ghost 3 will call the max function to go another depth lower.

4. How is depth=4 reached?
    a. 1st-ply to 2nd-ply to 3rd-ply to four-ply where each ply consists of pacman -> ghost 1 -> ghost 2 -> ghost 3

5. How does the root node return the best action?
    a. To get the action with the highest value, first find the index of valueList that contains the largest element. Then navigate to the same index of actionList and extract that element; that element is the best action

6. How do you set the PlayerIndex for Pacman and for three ghosts?
    a. Index = 0 for pacman
    b. Index = 1 for ghost 1
    c. Index = 2 for ghost 2
    d. Index = 3 for ghost 3 (terminal state)

7. When depth<4, how to go from the 3rd ghost to the next player-MAX, and at the same time increase
   the depth by 1?
    a. Since depth and the agent index are passed to the "max" function as arguments, when the 3rd ghost calls the "max" function, it will pass the next state, the index 0, which represents Pacman, and depth+1, which increments the depth by 1. This is the only place where depth is incremented.