COEN 266 Artificial Intelligence

Homework #5: Pacman-II

Name:	ID:
Benjamin Wang	1179478
Christopher Tam	1102598
Himanshu Dahiya	1629215

Contents

Task: Minimax Agent (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
    """

    def minimax_value_func(self, gameState, playerIndex: int, currentDepth: int):
      """
      Return the best_score (min/max) for the given gameState and playerIndex

      Args:
        gameState (gameState): Pacman Game gameState
        playerIndex (int): Pacman is always agent index 0, Ghosts are >= 1.
        currentDepth (int): Current depth of the search tree.

      Returns:
        best_score (int): The min/max value of the successor gameStates for playerIndex
      """
      # Exit Conditions:
      if gameState.isWin() or gameState.isLose() or currentDepth >= self.depth:
        # If node is leaf node (terminal state), return score without updating the gameState
        return self.evaluationFunction(gameState)
```

Figure 1 - Part 1 of 2 of Code

```
        # Collect legal moves and successor states
        legalMoves = gameState.getLegalActions(playerIndex)
        succPlayerIndex = (playerIndex + 1) % gameState.getNumAgents()
        succDepth = (currentDepth + 1) if succPlayerIndex == 0 else currentDepth

        # Recursively get the score of the successor states
        scores = [
            minimax_value_func(
                self,
                gameState.generateSuccessor(playerIndex, action),
                succPlayerIndex,
                succDepth,
            )
            for action in legalMoves
        ]

        # Choose one of the best actions
        best_score = max(scores) if playerIndex == 0 else min(scores)
        # # Update the actual gameState
        # gameState = gameState.generateSuccessor(playerIndex, action)
        if currentDepth == 0 and playerIndex == 0:
            bestIndices = [
                index for index in range(len(scores)) if scores[index] == best_score
            ]
            # Pick randomly among the same best scores
            chosenIndex = random.choice(bestIndices)
            best_action = legalMoves[chosenIndex]
            return best_score, best_action
        return best_score

    # Driver code:
    playerIndex = 0  # Always start at Pacman
    currentDepth = 0
    root_node_value, best_action = minimax_value_func(
        self, gameState, playerIndex=playerIndex, currentDepth=currentDepth
    )
    print(root_node_value)
    return best_action
    # util.raiseNotDefined()
```

Figure 2 - Part 2 of 2 of Code

Comment:

```
    def minimax_value_func(self, gameState, playerIndex: int, currentDepth: int):
```

This minimax_value_func() corresponds to both the maxValue_func() and minValue_func() example sub-functions in the HW5_submission_sample PDF as it does the function of both.

```
# Exit Conditions:
if gameState.isWin() or gameState.isLose() or currentDepth >= self.depth:
    # If node is leaf node (terminal state), return score without updating the gameState
    return self.evaluationFunction(gameState)
```

Since we're recursively calling minimax_value_func(), we define the exit conditions at the beginning of this sub-function as when the node is a leaf aka terminal state or has reached the set depth just return the value.

```
# Collect legal moves and successor states
legalMoves = gameState.getLegalActions(playerIndex)
succPlayerIndex = (playerIndex + 1) % gameState.getNumAgents()
succDepth = (currentDepth + 1) if succPlayerIndex == 0 else currentDepth
```

This section defines the available branches of a node, the successor's agent Index (playerIndex), and keep track of the successor's depth. The logic for successor's depth is that if its playerIndex resets to 0, it means we have gone through 1 ply/depth so add 1 to the tracker (currentDepth), otherwise stay same.

```
# Recursively get the score of the successor states
scores = [
    minimax_value_func(
        self,
        gameState.generateSuccessor(playerIndex, action),
        succPlayerIndex,
        succDepth,
    )
    for action in legalMoves
]
```

This is where the recursive call to itself takes place with each call returning only the best value with the exception for the first call from player 0 (Pacman) at depth 0 as will be explained later. This code snippet and the previous code snippet can be condensed into a one-liner if the previous code snippet's three variables (legalMoves, succPlayerIndex, and succDepth) are inserted into where they are used in this code snippet, but we chose to not do this for better readability.

```
# Choose one of the best actions
best_score = max(scores) if playerIndex == 0 else min(scores)
```

Best value or best score is defined as the maximum and minimum of recursively returned values if the agent is Pacman (agent index == 0) and if the agent is Ghost (agent index >= 1), respectively. This is the literal only difference between a maxValue_func() and minValue_func() and why they shouldn't be split into two functions.

```
    if currentDepth == 0 and playerIndex == 0:
        bestIndices = [
            index for index in range(len(scores)) if scores[index] == best_score
        ]
        # Pick randomly among the same best scores
        chosenIndex = random.choice(bestIndices)
        best_action = legalMoves[chosenIndex]
        return best_score, best_action
    return best_score
```

Ignoring the if-case for a moment, all but one special case calls for the minimax_value_func() to return only the value. If and only if the current depth is at 0 and the agent is Pacman, meaning we're at the very start/top of the game tree, then we want to also return the best action. We first get the indices of the best scoring branches, of which there might be several with exact same scores, then pick one among them fairly by random choice, then finally use that index to map it to the corresponding legal move as best action.

```
    # Driver code:
    playerIndex = 0  # Always start at Pacman
    currentDepth = 0
    root_node_value, best_action = minimax_value_func(
        self, gameState, playerIndex=playerIndex, currentDepth=currentDepth
    )
    print(root_node_value)
    return best_action
```

This is the part of the getAction() that initiate the recursive call to minimax_value_func(). We know that we always start at Pacman, aka agent index 0, and we start the tracker for depth at 0. Root_node_value or the best_score is returned so we can print it out, and we return the calculated best_action without updating the gameState because this is just returning what action to take, not taking the action itself.

Contribution Percentage

Benjamin Wang 40%

Christopher Tam 30%

Himanshu Dahiya..... 30%