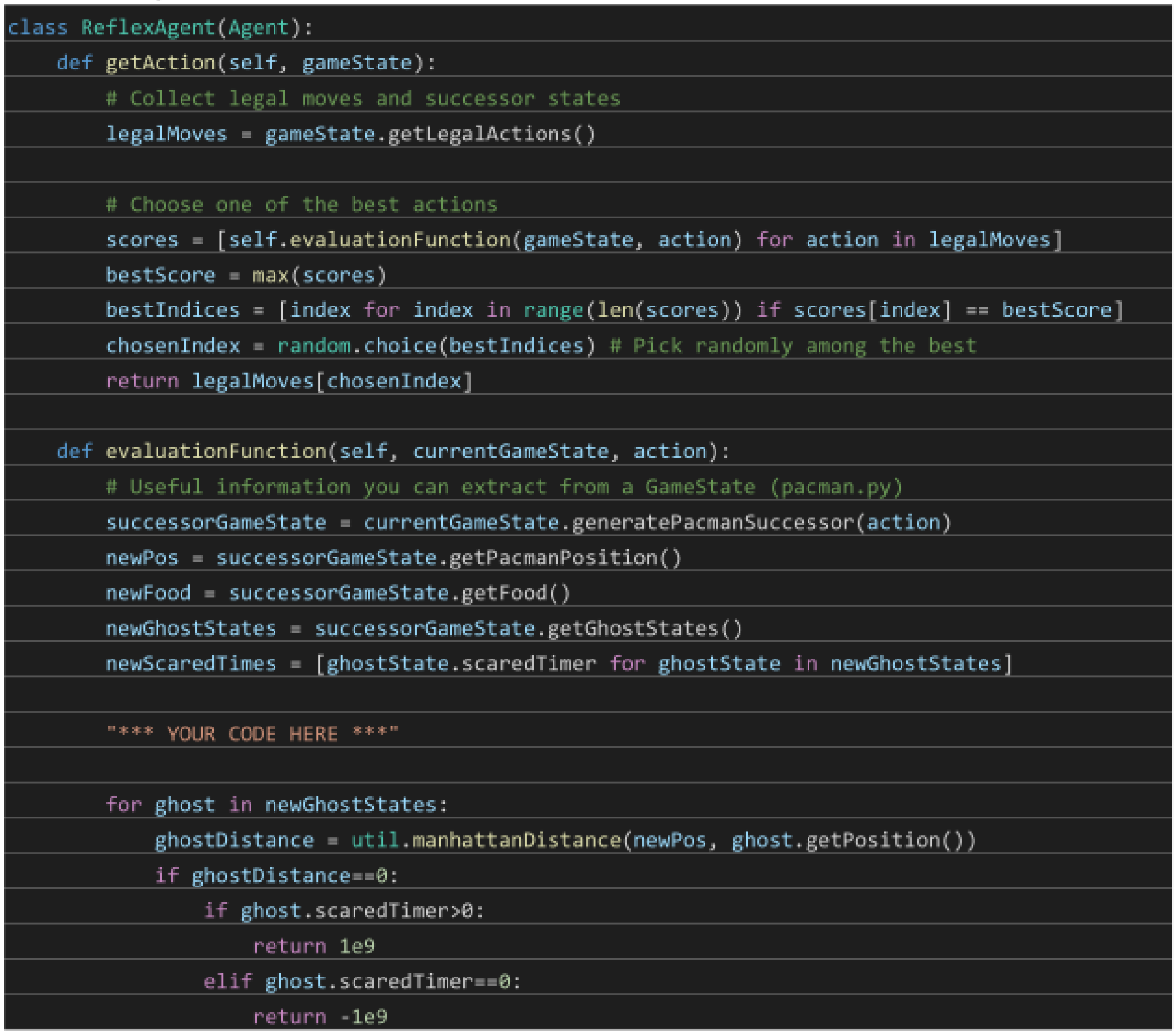
Lab 4

Tasnimul Hasnat
190041113

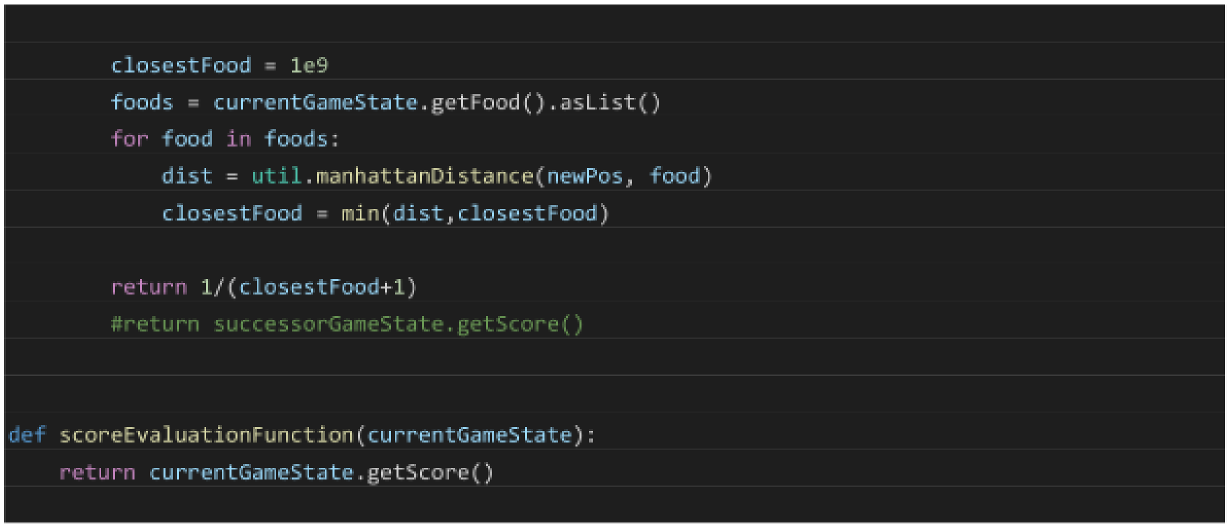
Saturday, Sep 30 2023

Task 1

This task required us to improve the ReflexAgent class so that the pacman plays respectably. For this I worked on the evaluationFunction() function as described below:
At first to deal with the ghosts I calculated the Manhattan distance from the Pacman to each of the ghosts. If it's equal to 0 (Pacman and ghost in the same position) then I checked if the scared time is more than 0 or not (scared time over or not). If it is not (scared time over), then Pacman will die and -1x10^9 will be returned else the score 1x10^9 will be returned. If none of these happens then I took the list of currently available foods. Then I used a loop to find the closest food distance from Pacman using Manhattan distance between each of the foods and Pacman and taking the minimum of them. The reciprocal of the closest food distance + 1 will be the returned score. 1 is added so that divide by 0 situation doesn't come.
The code is given below:

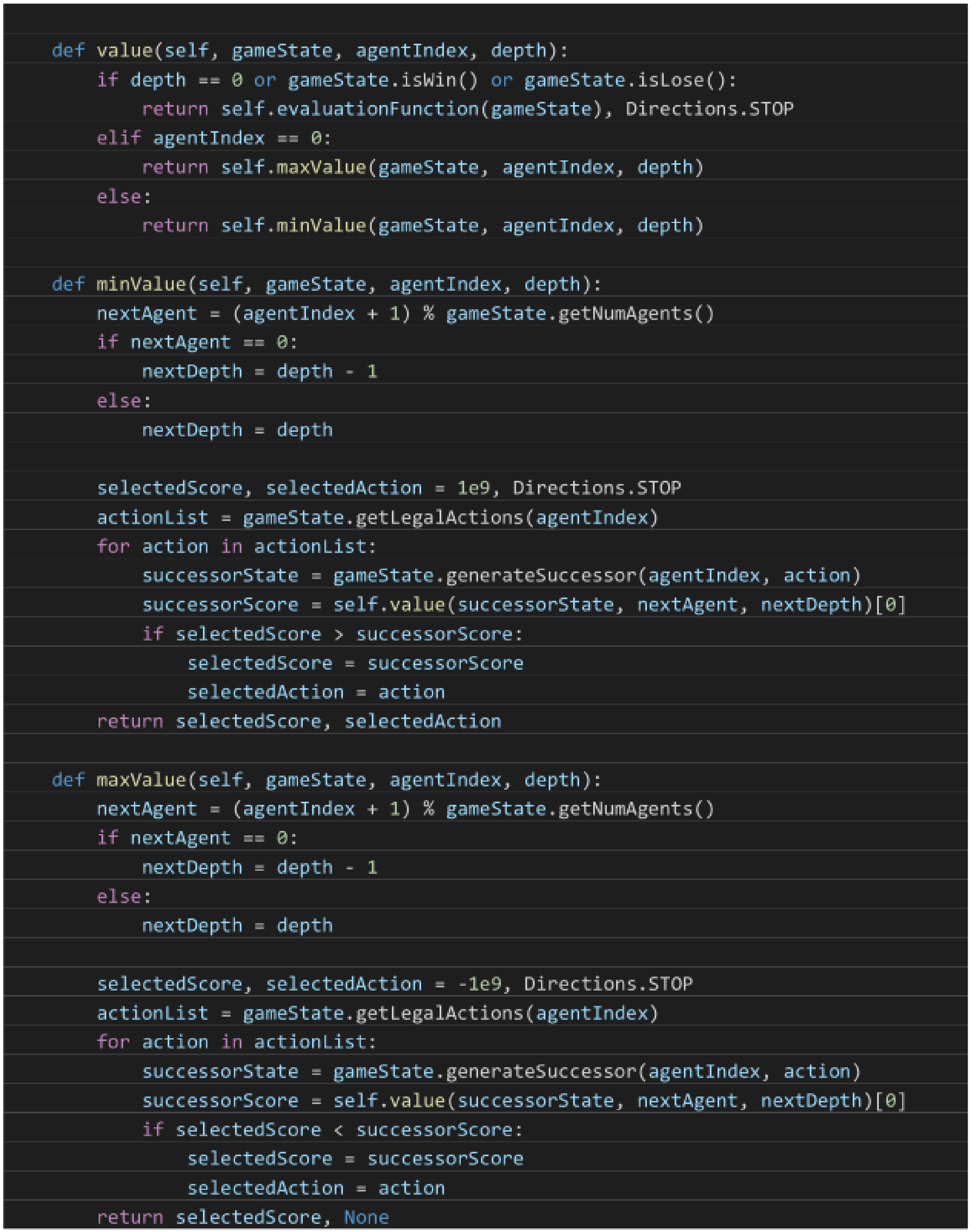


```python
class ReflexAgent(Agent):
    def getAction(self, gameState):
        # Collect legal moves and successor states
        legalMoves = gameState.getLegalActions()

        # Choose one of the best actions
        scores = [self.evaluationFunction(gameState, action) for action in legalMoves]
        bestScore = max(scores)
        bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
        chosenIndex = random.choice(bestIndices) # Pick randomly among the best
        return legalMoves[chosenIndex]

    def evaluationFunction(self, currentGameState, action):
        # 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]

        "*** YOUR CODE HERE ***"

        for ghost in newGhostStates:
            ghostDistance = util.manhattanDistance(newPos, ghost.getPosition())
            if ghostDistance==0:
                if ghost.scaredTimer>0:
                    return 1e9
                elif ghost.scaredTimer==0:
                    return -1e9
```


```
        closestFood = 1e9
        foods = currentGameState.getFood().asList()
        for food in foods:
            dist = util.manhattanDistance(newPos, food)
            closestFood = min(dist,closestFood)

        return 1/(closestFood+1)
        #return successorGameState.getScore()


def scoreEvaluationFunction(currentGameState):
    return currentGameState.getScore()
```

Task 2

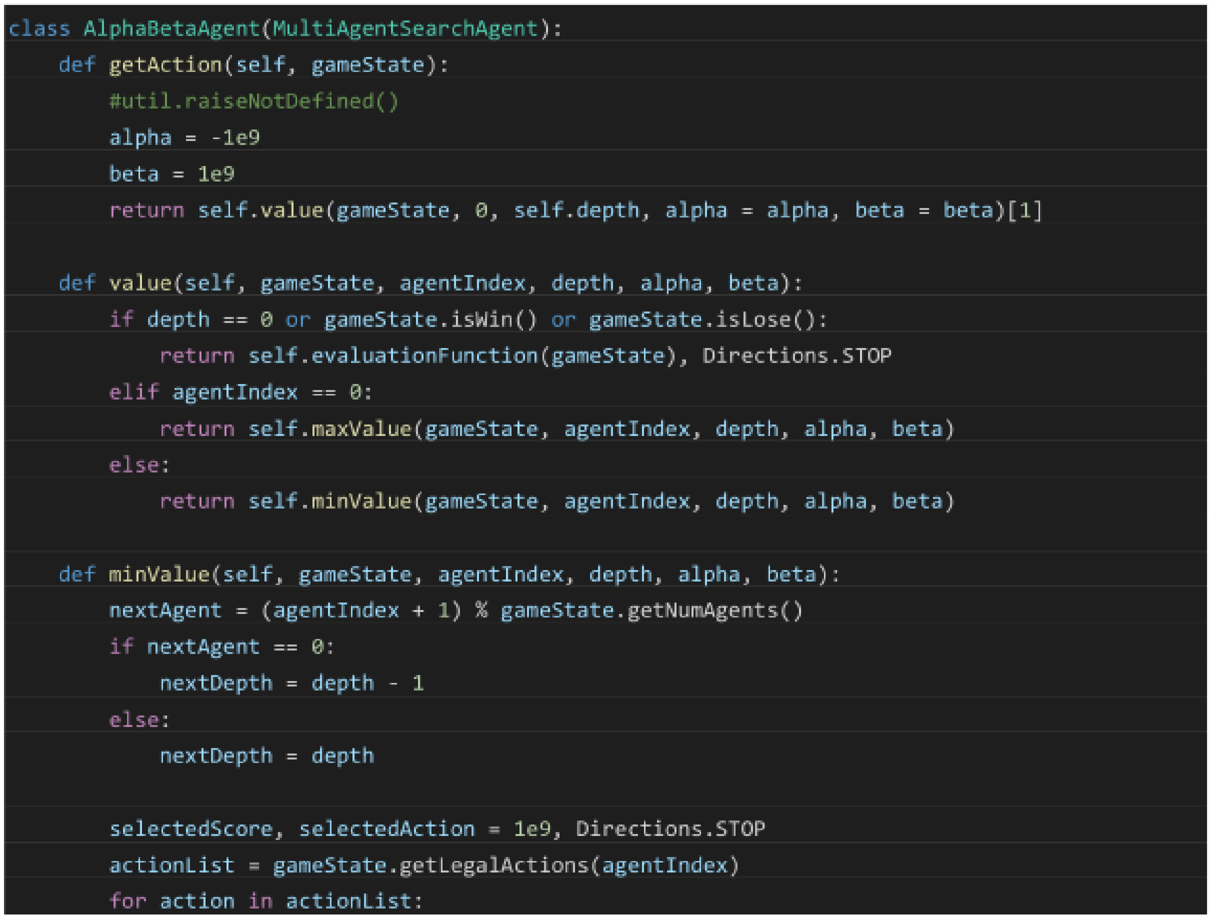
This task required us to implement minimax search for Pacman and the ghosts where the Pacman does the Max function and the ghosts do the Min function. This task was solved for us where the work was done in the MinimaxAgent class.
The main functions for minimax agent is minValue and maxValue. In both cases at first, if the next agent is Pacman then depth is decreased by 1 because it means actions for Pacman and all the ghosts are done for the previous depth level. Else the depth remains same for next ghost agent. Then for every action we collect the next successor state from the current state using given functions. Then we collect the scores for reaching theses successor states. Finally, for minValue function we take the minimum of the scores and return the score and the action that led to this minimum score. Similarly, for maxValue function we take the maximum of the scores and return the score and the action that led to this maximum score.
In the value function the gamestate, agentIndex and depth is given as parameter. Here at first it is checked if the depth limit has been reached or if the game has come to a decision of win or lose. If it is then the evaluationFunction is called to calculate the score and the game is stopped by giving direction STOP. Else if the agent is Pacman then maxValue function is called and if it's any of the ghosts then minValue function is called. This function returns the next score and action.
Finally, the getAction function calls the value function for Pacman from a game state and returns the action.
The code is given below:



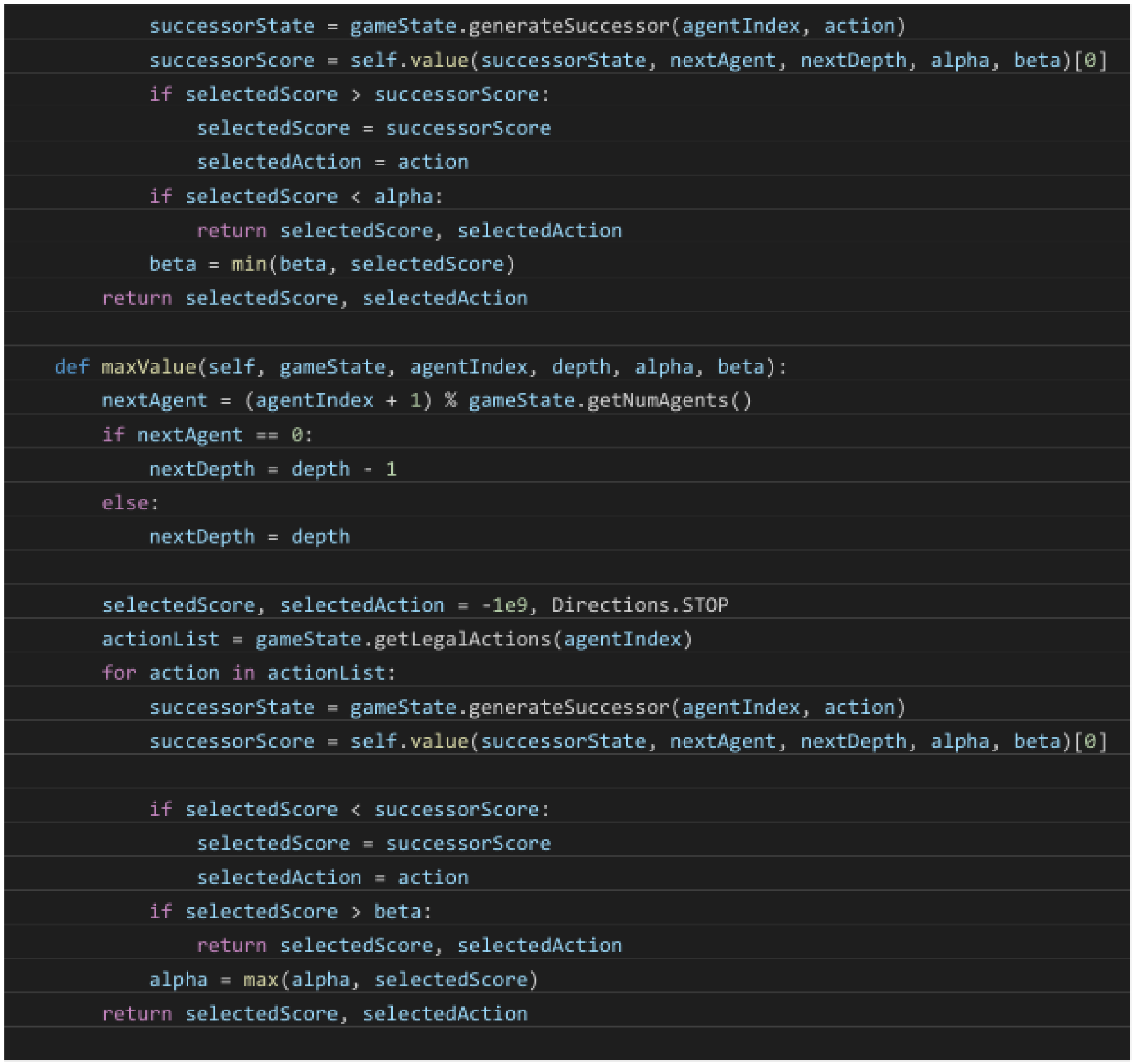
```python
    def value(self, gameState, agentIndex, depth):
        if depth == 0 or gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState), Directions.STOP
        elif agentIndex == 0:
            return self.maxValue(gameState, agentIndex, depth)
        else:
            return self.minValue(gameState, agentIndex, depth)

    def minValue(self, gameState, agentIndex, depth):
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        if nextAgent == 0:
            nextDepth = depth - 1
        else:
            nextDepth = depth

        selectedScore, selectedAction = 1e9, Directions.STOP
        actionList = gameState.getLegalActions(agentIndex)
        for action in actionList:
            successorState = gameState.generateSuccessor(agentIndex, action)
            successorScore = self.value(successorState, nextAgent, nextDepth)[0]
            if selectedScore > successorScore:
                selectedScore = successorScore
                selectedAction = action
        return selectedScore, selectedAction

    def maxValue(self, gameState, agentIndex, depth):
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        if nextAgent == 0:
            nextDepth = depth - 1
        else:
            nextDepth = depth

        selectedScore, selectedAction = -1e9, Directions.STOP
        actionList = gameState.getLegalActions(agentIndex)
        for action in actionList:
            successorState = gameState.generateSuccessor(agentIndex, action)
            successorScore = self.value(successorState, nextAgent, nextDepth)[0]
            if selectedScore < successorScore:
                selectedScore = successorScore
                selectedAction = action
        return selectedScore, None
```

Task 3

This task required us to implement alpha beta pruning for the minimax tree. For this I worked on the AlphaBetaAgent class. The code is pretty much same as the previous one with additions of some more lines of code.
Here in the minValue and maxValue function, after selecting the minimum and maximum score with the action that led to the score we compare the score with the best option on path to root. If the min/max score is worse than the best option, then we discard it. For minValue we check the score, if it's less than alpha (MAX's best option on path to root). If it is then we prune that is we return the score, action and don't calculate further. We also update the beta if the score is less than its present value, before returning. For maxValue we check the score, if its greater than beta (MIN's best option on path to root). If it is then we prune that is we return the score, action and don't calculate further. We also update the alpha if the score is greater than its present value before returning.
Additionally in value(), minValue(), maxValue() function we give alpha, beta as the parameters.
The code is given below:



```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    def getAction(self, gameState):
        #util.raiseNotDefined()
        alpha = -1e9
        beta = 1e9
        return self.value(gameState, 0, self.depth, alpha = alpha, beta = beta)[1]

    def value(self, gameState, agentIndex, depth, alpha, beta):
        if depth == 0 or gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState), Directions.STOP
        elif agentIndex == 0:
            return self.maxValue(gameState, agentIndex, depth, alpha, beta)
        else:
            return self.minValue(gameState, agentIndex, depth, alpha, beta)

    def minValue(self, gameState, agentIndex, depth, alpha, beta):
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        if nextAgent == 0:
            nextDepth = depth - 1
        else:
            nextDepth = depth

        selectedScore, selectedAction = 1e9, Directions.STOP
        actionList = gameState.getLegalActions(agentIndex)
        for action in actionList:
```

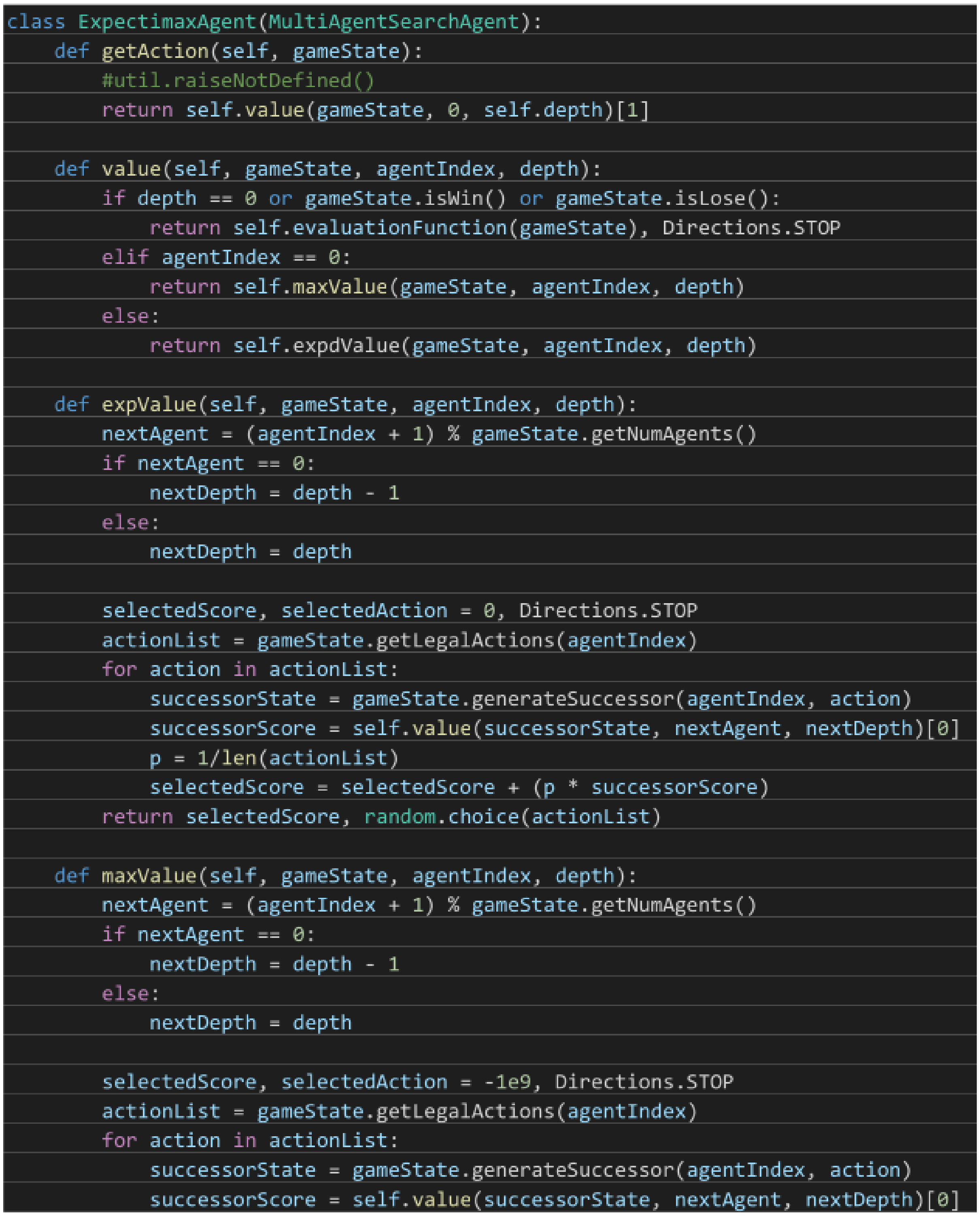


```
            successorState = gameState.generateSuccessor(agentIndex, action)
            successorScore = self.value(successorState, nextAgent, nextDepth, alpha, beta)[0]
            if selectedScore > successorScore:
                selectedScore = successorScore
                selectedAction = action
            if selectedScore < alpha:
                return selectedScore, selectedAction
            beta = min(beta, selectedScore)
        return selectedScore, selectedAction

    def maxValue(self, gameState, agentIndex, depth, alpha, beta):
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        if nextAgent == 0:
            nextDepth = depth - 1
        else:
            nextDepth = depth

        selectedScore, selectedAction = -1e9, Directions.STOP
        actionList = gameState.getLegalActions(agentIndex)
        for action in actionList:
            successorState = gameState.generateSuccessor(agentIndex, action)
            successorScore = self.value(successorState, nextAgent, nextDepth, alpha, beta)[0]

            if selectedScore < successorScore:
                selectedScore = successorScore
                selectedAction = action
            if selectedScore > beta:
                return selectedScore, selectedAction
            alpha = max(alpha, selectedScore)
        return selectedScore, selectedAction
```

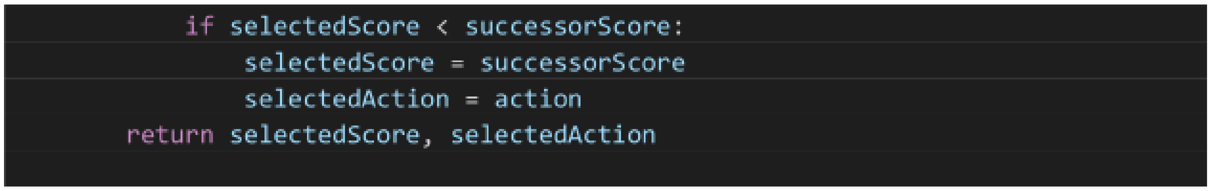
Task 4

This task required us to implement expectimax. For this I worked on the ExpectimaxAgent class. The code is pretty much same as the minimax one with some modifications.
So here the only change is that, instead of using minValue function we used expValue function to calculate the expected value. All other functionalities are exactly same as the minimax one. In the expValue function after getting score for each of the states that each of the actions led to, we calculate the expected value of the scores. Where uniform distribution is assumed among the states. Thus the probability p=1/len(actionList) is multiplied with each of the scores and they are

all added to get the expected value. This expected value and a randomly chosen action is returned from the function.
The code is given below:



```
class ExpectimaxAgent(MultiAgentSearchAgent):
    def getAction(self, gameState):
        #util.raiseNotDefined()
        return self.value(gameState, 0, self.depth)[1]

    def value(self, gameState, agentIndex, depth):
        if depth == 0 or gameState.isWin() or gameState.isLose():
            return self.evaluationFunction(gameState), Directions.STOP
        elif agentIndex == 0:
            return self.maxValue(gameState, agentIndex, depth)
        else:
            return self.expdValue(gameState, agentIndex, depth)

    def expValue(self, gameState, agentIndex, depth):
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        if nextAgent == 0:
            nextDepth = depth - 1
        else:
            nextDepth = depth

        selectedScore, selectedAction = 0, Directions.STOP
        actionList = gameState.getLegalActions(agentIndex)
        for action in actionList:
            successorState = gameState.generateSuccessor(agentIndex, action)
            successorScore = self.value(successorState, nextAgent, nextDepth)[0]
            p = 1/len(actionList)
            selectedScore = selectedScore + (p * successorScore)
        return selectedScore, random.choice(actionList)

    def maxValue(self, gameState, agentIndex, depth):
        nextAgent = (agentIndex + 1) % gameState.getNumAgents()
        if nextAgent == 0:
            nextDepth = depth - 1
        else:
            nextDepth = depth

        selectedScore, selectedAction = -1e9, Directions.STOP
        actionList = gameState.getLegalActions(agentIndex)
        for action in actionList:
            successorState = gameState.generateSuccessor(agentIndex, action)
            successorScore = self.value(successorState, nextAgent, nextDepth)[0]
```

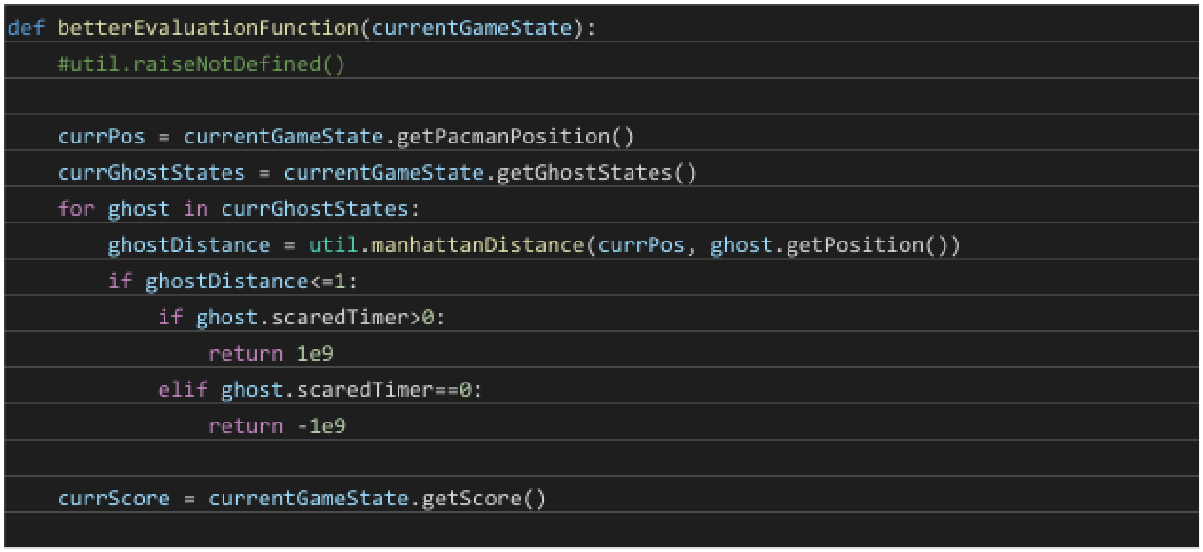


```
            if selectedScore < successorScore:
                selectedScore = successorScore
                selectedAction = action
        return selectedScore, selectedAction
```

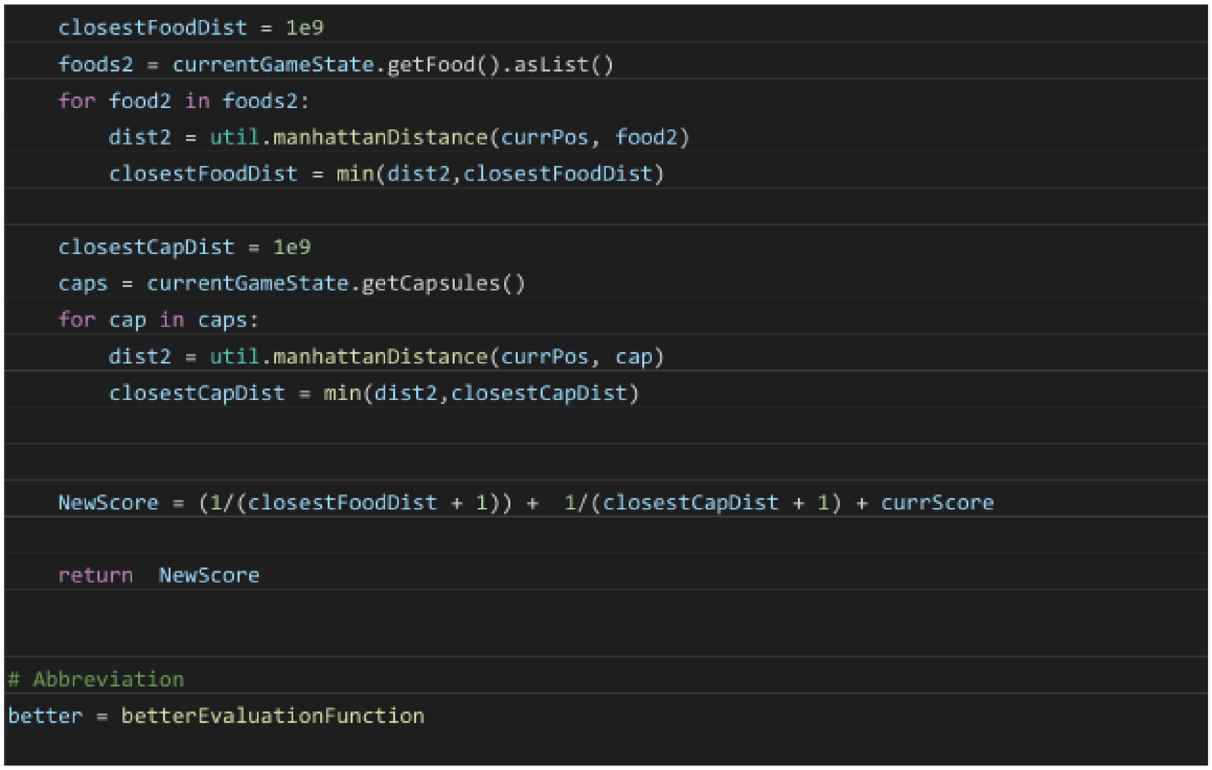
Please Turn Over

Task 5

This task required us to implement a better evaluation function than what it really is. For this I worked on the betterEvaluationFunction() function. The task is almost similar to task 1 with some additions and modifications.
At first to deal with the ghosts I calculated the Manhattan distance from the Pacman to each of the ghosts. If it's less or equal to 1 (Pacman and ghost may land in the same position after the next move or they are already in the same position). If it is then I checked if the scared time is more than 0 or not (scared time over or not). If it is not (scared time over), then Pacman will die and -1x10^9 will be returned else the score 1x10^9 will be returned. If none of these happens then the next part is executed. Here I used a loop to find the closest food distance from Pacman using Manhattan distance between each of the foods and Pacman and taking the minimum of them. The extra part that I did here than task1 is I calculated the distance to closest capsule or power pallet from Pacman. I did this by using a loop and finding the Manhattan distance from Pacman to each of the capsules and taking the minimum of them. Finally, the new score is calculated as the summation of 1/ (closest food distance +1), 1/ (closest capsule distance +1) and the current score. 1 is added so that divide by 0 situation doesn't come. This new score is retuned.
The code is given below:



```
def betterEvaluationFunction(currentGameState):
    #util.raiseNotDefined()

    currPos = currentGameState.getPacmanPosition()
    currGhostStates = currentGameState.getGhostStates()
    for ghost in currGhostStates:
        ghostDistance = util.manhattanDistance(currPos, ghost.getPosition())
        if ghostDistance<=1:
            if ghost.scaredTimer>0:
                return 1e9
            elif ghost.scaredTimer==0:
                return -1e9

    currScore = currentGameState.getScore()

```



```
    closestFoodDist = 1e9
    foods2 = currentGameState.getFood().asList()
    for food2 in foods2:
        dist2 = util.manhattanDistance(currPos, food2)
        closestFoodDist = min(dist2,closestFoodDist)

    closestCapDist = 1e9
    caps = currentGameState.getCapsules()
    for cap in caps:
        dist2 = util.manhattanDistance(currPos, cap)
        closestCapDist = min(dist2,closestCapDist)


    NewScore = (1/(closestFoodDist + 1)) +  1/(closestCapDist + 1) + currScore

    return  NewScore


# Abbreviation
better = betterEvaluationFunction
```

—XXX—