Diego Sáez Mulero U137519
Amanda Pintado Lineros U137702

Project 2- Pacman Multiplayer Search

The objective of this lab is to implement and get familiar with the multiplayer game for Pacman.

Introduction

We can find a Pacman Project with all the specifications and the files we need to complete. We had to fill some methods for the classes in multiAgent.py that will provide us information to create paths for the pacman.

evaluationFunction de la clase ReflexAgent

```
        #First, the food-------------
        #List for food, minimum distance and actual position of the food
        foodDistance = []
        minFoodDist = 0
        foodPos = newFood.asList()

        #We will pass trhought all the food and get their distance wiht pacman
        for food in foodPos:
            distances = util.manhattanDistance(newPos, food)
            foodDistance.append(distances)

        #if the list is empty, then we add the lower distance from foodDistance
        if len(foodDistance) > 0:
            minFoodDist = min(foodDistance)

        #For the ghosts--------
        #List of distances with the ghosts, minimun distance and actual position of ghosts
        ghostsDistance = []
        minGhostDist = 0
        ghostPos = currentGameState.getGhostPositions()

        #We will pass trhought all the ghost and get their distance wiht pacman
        for ghost in ghostPos:
            distance = util.manhattanDistance(newPos, ghost)
            ghostsDistance.append(distance)

        #if the list is empty, then we add the lower distance from ghostsDistance
        if len(ghostsDistance) > 0:
            minGhostDist = min(ghostsDistance)

        #The score will be determined by the ghost and the food missed
        return successorGameState.getScore()-(50.0 / (minGhostDist + 1.0))-(minFoodDist)
```

For this function, we have to provide a value with the best score in function of the food position and ghost (as their different types). The return has a ponderation that subtract more if the variable is close to 0, that's because be find some different resources that apply this same reduction.

Diego Sáez Mulero U137519
Amanda Pintado Lineros U137702

getAction from the class MinimaxAgent

```
          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
        """
        "*** YOUR CODE HERE ***"


        def minimax(agent, depth, gameState):

            if gameState.isLose() or gameState.isWin() or depth == self.depth:
                #devuelve la evaluacion en caso de perder/ganar/o llegar a la profundidad definida
                return self.evaluationFunction(gameState)

            if agent == 0:
                #Maximiza para Pacman
                maxim = max(minimax(1, depth, gameState.generateSuccessor(agent, newState)) for newState i
                return maxim

            else:
                #Minimiza para fantasmas
                nextAgent = agent + 1
                if gameState.getNumAgents() == nextAgent:
                    nextAgent = 0
                if nextAgent == 0:
                    depth = depth + 1#aumentamos la profundidad
                #Calculamos el valor minimo de los nodos
                minim = min(minimax(nextAgent, depth, gameState.generateSuccessor(agent, newState)) for ne
                return minim


        maximum = float("-inf") #Inicio raiz valor infinito negativo
        action = Directions.WEST


        for agentState in gameState.getLegalActions(0):#acciones

            suc = gameState.generateSuccessor(0, agentState)

            val = minimax(1, 0, suc)

            if val >= maximum:
                maximum = val
                action = agentState

        return action
```

For this, we implement an auxiliar function, and with this determine a value for the action and then make a return of this.

getAction from the class AlphaBetaAgent

```python
        "*** YOUR CODE HERE ***"

        #Funcion que maximiza
        def maximizer(agent, depth, game_state, alpha, beta):
            score = float("-inf")

            for newState in game_state.getLegalActions(agent):

                suc = game_state.generateSuccessor(agent, newState)
                score = max(score, alphaBetaPrune(1, depth, suc, alpha, beta))
                if score > beta:
                    return score
                alpha = max(alpha, score)

            return score
        #Funcion que minimiza
        def minimizer(agent, depth, game_state, alpha, beta):
            score = float("inf")

            next_agent = agent + 1
            if game_state.getNumAgents() == next_agent:
                next_agent = 0
            if next_agent == 0:
                depth = depth + 1 #aumentamos la profundidad

            for newState in game_state.getLegalActions(agent):

                suc = game_state.generateSuccessor(agent, newState)

                score = min(score, alphaBetaPrune(next_agent, depth, suc, alpha, beta))

                if score < alpha:
                    return score
                beta = min(beta, score)

            return score

        def alphaBetaPrune(agent, depth, game_state, alpha, beta):

            if game_state.isLose() or game_state.isWin() or depth == self.depth:
                #devuelve la evaluacion en caso de perder/ganar/o llegar a la profundidad definida
                return self.evaluationFunction(game_state)

            if agent == 0:
                #Maximiza para Pacman
                return maximizer(agent, depth, game_state, alpha, beta)
            else:
                #Minimiza para fantasmas
                return minimizer(agent, depth, game_state, alpha, beta)
```

As for the previous part, we had to implement some functions to help us to find the action. The first function gives us the maximized score determined by the agent and the game state be received, for the second is returned de minimal and with the tirth function takes the return from the two previous functions.

```python
        action = Directions.WEST
        val = float("-inf")
        alpha = float("-inf")
        beta = float("inf")
        for agentState in gameState.getLegalActions(0):

            suc = gameState.generateSuccessor(0, agentState)
            ghostVal = alphaBetaPrune(1, 0, suc, alpha, beta)

            if ghostVal > val:
                val = ghostVal
                action = agentState

            if val > beta:
                return val

            alpha = max(alpha, val)


        return action
        #util.raiseNotDefined()
```

The use of these functions is to calculate the value of the ghosts and make a return of the actions if it is suitable to win for the pacman.

getAction from the class ExpectimaxAgent

Diego Sáez Mulero U137519
Amanda Pintado Lineros U137702

```
def getAction(self, gameState):
    """
      Returns the expectimax action using self.depth and self.evaluationFunction

      All ghosts should be modeled as choosing uniformly at random from their
      legal moves.
    """
    "*** YOUR CODE HERE ***"


    def expectimax(agent, depth, gameState):

        if gameState.isLose() or gameState.isWin() or depth == self.depth:
            return self.evaluationFunction(gameState)

        if agent == 0:
            #Maximiza para Pacman
            return max(expectimax(1, depth, gameState.generateSuccessor(agent, newState)) for newState
        else:
            #No toma el minimo sino la expectativa
            nextAgent = agent + 1  # Calcula el siguiente agente
            if gameState.getNumAgents() == nextAgent:
                nextAgent = 0
            if nextAgent == 0:
                depth = depth + 1#aumentamos la profundidad
            return sum(expectimax(nextAgent, depth, gameState.generateSuccessor(agent, newState)) for


    maximum = float("-inf")
    action = Directions.WEST

    for agentState in gameState.getLegalActions(0):

        val = expectimax(1, 0, gameState.generateSuccessor(0, agentState))

        if val > maximum:
            maximum = val
            action = agentState


    return action #devuelve la accion
    #util.raiseNotDefined()
```

Using expectimax as auxiliar function to get the closest value to win, be have a return of the action in function of the depth of the actual state.

betterEvaluationFunction

We will use the code seen in the first part, on evaluationFunction.

```python
"*** YOUR CODE HERE ***"

#We used what be see in evaluationFunction
newPos = currentGameState.getPacmanPosition()
newFood = currentGameState.getFood()
newGhostStates = currentGameState.getGhostStates()
newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

#First, the food-------------
#List for food, minimum distance and actual position of the food
foodDistance = []
minFoodDist = 0
foodPos = newFood.asList()

#We will pass trhought all the food and get their distance wiht pacman
for food in foodPos:
    distances = util.manhattanDistance(newPos, food)
    foodDistance.append(distances)

#if the list is empty, then we add the lower distance from foodDistance
if len(foodDistance) > 0:
    minFoodDist = min(foodDistance)

#For the ghosts--------
#List of distances with the ghosts, minimun distance and actual position of ghosts
ghostsDistance = []
minGhostDist = 0
ghostPos = currentGameState.getGhostPositions()
#We well use also a varaible for index min ghost
indexGhost = -1

#We will pass trhought all the ghost and get their distance wiht pacman
for ghost in ghostPos:
    distance = util.manhattanDistance(newPos, ghost)
    ghostsDistance.append(distance)

#if the list is empty, then we add the lower distance from ghostsDistance
#and we save the distance
if len(ghostsDistance) > 0:
    minGhostDist = min(ghostsDistance)
    indexGhost = ghostsDistance.index(minGhostDist)

#If there is no ghost then the score will depend on the food
if(newScaredTimes[indexGhost] == -1):
    return currentGameState.getScore()-(minFoodDist)
#If there is a ghost available to eat with the minimum distance, you gave to get it (a
elif(newScaredTimes[indexGhost] > 0):
    return currentGameState.getScore() + (100.0 / (minGhostDist + 1.0))-(minFoodDist)
#If there is ghosts
else:
    return currentGameState.getScore()-(50.0 / (minGhostDist + 1.0))-(minFoodDist)
#util.raiseNotDefined()
```

Following the implementation of the first question, we have to add a way to determine the score in function of the food, ghosts available to eat and if there still are ghosts around. To determine the reductions we use some internet research to make the markdown the most fair and have a good score versus strategy.

Conclusions

For the reductions of the score, to make a good proportion, we have to do some internet research and adapt it to our problem.