GLOBAL CAMPUS
NEBRIJA

PRACTICA 2

INTELIGENCIA ARTIFICIAL

José María Fernández Gómez

Pregunta 1 (4 puntos): Agente reflejo

----------CODIGO----------

```
comidaAdy = 0
for com in newFood[newPos[0]-1:newPos[0]+2]:
    for punt in com[newPos[1]-1:newPos[1]+2]:
        if punt: comidaAdy += 1

eatGhost = 0
for indice in range(len(newGhostStates)):
    dist = util.manhattanDistance(newPos, newGhostStates[indice].getPosition())
    if dist == 0: dist = 1e-8
    if dist <= newScaredTimes[indice] / 1.5:
        eatGhost += (1./dist)*100
    elif dist <= 3:
        eatGhost -= (1./dist)*100

com = 0
if newPos in currentGameState.getFood().asList():
    com = 10

tabu = 0
if newPos in self.tabu_list:
    tabu = -100

return comidaAdy + eatGhost + tabu + com
return successorGameState.getScore()
```

----------END CODIGO----------

Esta nueva función de evaluación devuelve un numero entero que representa por decirlo de alguna forma lo “buena” que es la casilla a la que nos vamos a mover, este numero es la suma de varios factores como la cantidad de puntos de comida adyacentes que tenemos, si en esa casilla hay un punto de comida, si hay un fantasma asustado que nos podamos comer o una suma de un numero negativo que representa si hay un fantasma que nos pueda comer, por lo que esta función solo devuelve el “valor” que le hemos otorgado a cada casilla en función a estos parámetros y luego será el juego quien se encargue de movernos a la casilla con valor mas grande

Pregunta 2 (5 puntos): Minimax

----------CODIGO----------

```
        res = self.value(gameState, 0)
        return res[0]

    # MiniMax Algorithm min function
    def min(self, gameState, depth):
        num_agents = gameState.getNumAgents()
        actions = gameState.getLegalActions(depth % num_agents)
        if len(actions) == 0:
            return (None, self.evaluationFunction(gameState))
        min_score = (None, float("inf"))
        for action in actions:
            next_state = gameState.generateSuccessor(depth % num_agents, action)
            res = self.value(next_state, depth+1)
            if res[1] < min_score[1]:
                min_score = (action, res[1])
        return min_score


    # MiniMax Algorithm max function
    def max(self, gameState, depth):
        actions = gameState.getLegalActions(0)
        if len(actions) == 0:
            return (None, self.evaluationFunction(gameState))
        max_score = (None,-float("inf"))
        for action in actions:
            next_state = gameState.generateSuccessor(0, action)
            res = self.value(next_state, depth + 1)
            if res[1] > max_score[1]:
                max_score = (action, res[1])
        return max_score


    # MiniMax Algorithm value function
    def value(self, gameState, depth):
        if depth == self.depth * gameState.getNumAgents() or gameState.isWin() or
gameState.isLose():
            return (None, self.evaluationFunction(gameState))

        if depth % gameState.getNumAgents() == 0:
            return self.max(gameState, depth)
        else:
            return self.min(gameState, depth)
```

----------END CODIGO----------

Se ha desarrollado un algoritmo Minimax con su función que evalua el caso en el que el personaje tiene que conseguir la mayor puntuación y los fantasmas deben minimizar esa puntuación del personaje, también tiene una función value que es la que se encarga de asignar los roles a cada uno de los personajes, mínimo a los fantasmas y máximo al comecoco

Pregunta 3 (5 puntos): Poda alfa-beta

----------CODIGO----------

```
        res = self.value(gameState, 0, -float("inf"), float("inf"))
        return res[0]
        # AlphaBeta Algorithm value function
    def value(self, gameState, depth, alpha, beta):
            if depth == self.depth * gameState.getNumAgents() or gameState.isWin() or
gameState.isLose():
                return (None, self.evaluationFunction(gameState))

            if depth % gameState.getNumAgents() == 0:
                return self.max(gameState, depth, alpha, beta)
            else:
                return self.min(gameState, depth, alpha, beta)

        # AlphaBeta Algorithm min function
    def min(self, gameState, depth, alpha, beta):
            num_agents = gameState.getNumAgents()
            actions = gameState.getLegalActions(depth % num_agents)
            if len(actions) == 0:
                return (None, self.evaluationFunction(gameState))
            min_score = (None, float("inf"))
            for action in actions:
                next_state = gameState.generateSuccessor(depth % num_agents, action)
                res = self.value(next_state, depth + 1, alpha, beta)
                if res[1] < min_score[1]:
                    min_score = (action, res[1])
                if min_score[1] < alpha:
                    return min_score
                beta = min(beta, min_score[1])
            return min_score

        # AlphaBeta Algorithm max function
    def max(self, gameState, depth, alpha, beta):
            actions = gameState.getLegalActions(0)
            if len(actions) == 0:
                return (None, self.evaluationFunction(gameState))
            max_score = (None, -float("inf"))
            for action in actions:
                next_state = gameState.generateSuccessor(0, action)
                res = self.value(next_state, depth + 1, alpha, beta)
                if res[1] > max_score[1]:
                    max_score = (action, res[1])
                if mesopore[1] > beta:
                    return max_score
                alpha = max(alpha, max_score[1])
            return max_score
```

----------END CODIGO----------

Este apartado es muy similar al anterior, de hecho la función de valor es exactamente igual, ya que solo se encarga de identificar cada personaje y asignarle el rol máximo/mínimo, donde cambia este apartado es en las funciones máximo/mínimo, donde todo se comporta igual que en el caso anterior pero este algoritmo es mas eficiente ya que va “podando” ramas del árbol, esto quiere decir que si encuentra un camino mejor que lo que va a evaluar, decide no seguir por esa rama, quitando así opciones peores que se tendrían que haber evaluado, esto funciona para los dos roles max y min.

Pregunta 4 (5 puntos): Expectimax

----------CODIGO----------

```
  res = self.value(gameState, 0)
        return res[0]
    def value(self,gameState,depth):
        if depth == self.depth * gameState.getNumAgents() or gameState.isWin() or
gameState.isLose():
            return (None, self.evaluationFunction(gameState))
        if depth % gameState.getNumAgents() == 0:
            return self.max(gameState, depth)
        else:
            return self.exp(gameState, depth)
    def max(self,gameState,depth):
        actions = gameState.getLegalActions(0)
        if len(actions) == 0:
            return (None, self.evaluationFunction(gameState))
        max_score = (None,-float("inf"))
        for action in actions:
            next_state = gameState.generateSuccessor(0, action)
            res = self.value(next_state, depth + 1)
            if res[1] > max_score[1]:
              max_score = (action, res[1])
        return max_score
    def exp(self,gameState,depth):
        num_agents = gameState.getNumAgents()
        actions = gameState.getLegalActions(depth % num_agents)
        if len(actions) == 0:
            return (None, self.evaluationFunction(gameState))
        exp_score = 0
        for action in actions:
            next_state = gameState.generateSuccessor(depth % num_agents, action)
            res = self.value(next_state, depth + 1)
            exp_score += res[1]
        exp_score /= len(actions)
        return (None,exp_score)
```

----------END CODIGO----------

En este apartado se ha modelado un algoritmo suponiendo un comportamiento aleatorio o desconocido de los fantasmas, por lo que solo modelamos el comportamiento de nuestro personaje en función de la aleatoriedad de los movimientos de los fantasmas. Esto se hace por medio de ujna función exp(expectativa) que se encarga de calcular el promedio de las posibles acciones que puede tomar el contrario en función de la probabilidad que tienen de ocurrir, esta probabilidad ha sido definida como se indica en el enunciado a base de una distribución normal que depende de sus propias acciones.

Pregunta 5 (6 puntos): Función de evaluación

No he sido capaz de realizar este apartado