

eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Técnicas de Inteligencia Artificial

Ingeniería Informática de Gestión y Sistemas de Información

Practica 2

Búsqueda Multi-Agente

Autor(es):

Xabier Gabiña

Diego Montoya

22 de octubre de 2024

Índice general

1. Introducción	4
2. Ejercicios	5
2.1. Agente Reflex	5
2.2. Minimax	7
2.3. Podado Alpha-Beta	11
2.4. Expectimax	13
2.5. Función de Evaluación	16
3. Resultados	18
3.1. Casos de prueba	18
3.2. Autograder	19

Índice de figuras

2.1. Ejemplo de árbol de búsqueda Minimax	7
2.2. Ejemplo de poda Alpha-Beta	11

Índice de Códigos

2.1. Implementación inicial del agente reflex	5
2.2. Implementación final del agente reflex	5
2.3. Implementación inicial del agente Minimax	7
2.4. Implementación final del agente Minimax	8
2.5. Implementación inicial del agente Alpha-Beta	11
2.6. Implementación final del agente Alpha-Beta	11
2.7. Implementación inicial del agente Expectimax	13
2.8. Implementación final del agente Expectimax	14
2.9. Implementación inicial de la función de evaluación	16
2.10. Implementación final de la función de evaluación	16
3.1. Resultados del Autograder	19

1. Introducción

En esta práctica se implementarán varios algoritmos de búsqueda multi-agente para el juego de Pacman. Los algoritmos a implementar son los siguientes:

- Agente Reflex: Un agente que selecciona acciones a corto plazo basándose en la percepción actual del entorno.
- Minimax: Un algoritmo de búsqueda en árboles que se utiliza en juegos de dos o más jugadores y deterministas.
- Podado Alpha-Beta: Una mejora del algoritmo Minimax que reduce el número de nodos evaluados en el árbol de búsqueda.
- Expectimax: Una variante del algoritmo Minimax que se utiliza cuando los oponentes no toman siempre la mejor decisión posible.

Además, se mejorará la función de evaluación del agente reflex para que tome en cuenta más factores del estado del juego.

2. Ejercicios

2.1. Agente Reflex

Descripción

Un agente reflex es un agente que selecciona acciones a corto plazo basándose en la percepción actual del entorno. En este caso, el agente reflex se basa en una función de evaluación para seleccionar la mejor acción en cada estado. La función de evaluación asigna un valor numérico a cada estado, y el agente selecciona la acción que maximiza este valor. Para asignar un valor a cada estado, la función de evaluación considera varios factores, como la distancia a la comida, la proximidad a los fantasmas y la cantidad de comida restante.

Primera implementación

Código 2.1: Implementación inicial del agente reflex

Implementación final

```
1 class ReflexAgent(Agent):
2     """
3     Un agente reflexivo que elige acciones basándose en una función de evaluación.
4     """
5
6     def getAction(self, gameState):
7         """
8         Devuelve la mejor acción para Pacman basada en la función de evaluación.
9         """
10        # Obtiene las acciones legales para Pacman
11        legalMoves = gameState.getLegalActions()
12
13        # Calcula los puntajes para cada acción utilizando la función de evaluación
14        scores = [self.evaluationFunction(gameState, action) for action in legalMoves]
15        bestScore = max(scores) # Encuentra el puntaje más alto
16        bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
17        chosenIndex = random.choice(bestIndices) # Elige aleatoriamente entre las mejores acciones
18
19        return legalMoves[chosenIndex]
20
21    def evaluationFunction(self, currentGameState, action):
22        """
23        Calcula el valor del estado sucesor después de que Pacman toma la acción 'action'.
24        Devuelve un valor numérico mayor para estados más favorables.
25        """
26        # Generar el estado sucesor
27        successorGameState = currentGameState.generatePacmanSuccessor(action)
28        # Obtiene la posición de Pacman después de moverse
29        newPos = successorGameState.getPacmanPosition()
30        # Obtiene la matriz de comida en el estado sucesor
31        newFood = successorGameState.getFood()
32        # Obtiene la lista de estados de los fantasmas
33        newGhostStates = successorGameState.getGhostStates()
34        # Obtiene los tiempos restantes de los fantasmas asustados
35        newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
36
37        # Inicializar el puntaje con el puntaje base del sucesor
38        score = successorGameState.getScore()
39
40        # 1. Distancia a la comida más cercana
41        foodList = newFood.asList() # Convertir la matriz de comida a una lista de posiciones
42        if foodList: # Si hay comida disponible
43            # Calcular la distancia mínima a la comida más cercana
44            minFoodDistance = min([manhattanDistance(newPos, food) for food in foodList])
45            # Invertir la distancia para que un menor valor de distancia dé un mayor puntaje
```

```

46         score += 10.0 / minFoodDistance
47
48     # 2. Distancia a los fantasmas no asustados
49     for ghost in newGhostStates:
50         ghostPos = ghost.getPosition()
51         ghostDistance = manhattanDistance(newPos, ghostPos)
52         if ghostDistance < 2 and ghost.scaredTimer == 0: # Fantasma no asustado y muy cerca
53             score -= 1000 # Penalización fuerte por estar demasiado cerca de un fantasma
54                             peligroso
55
56     # 3. Incentivo por acercarse a fantasmas asustados
57     for i, ghost in enumerate(newGhostStates):
58         if newScaredTimes[i] > 0: # Si el fantasma está asustado
59             ghostDistance = manhattanDistance(newPos, ghost.getPosition())
60             score += 200.0 / (ghostDistance + 1) # Premiar estar cerca de un fantasma asustado
61
62     # 4. Penalización por comida restante
63     score -= len(foodList) * 10 # Penalizar por cada comida restante en el estado sucesor
64
65     return score

```

Código 2.2: Implementación final del agente reflex

Comentarios

2.2. Minimax

Descripción

Minimax es un algoritmo de búsqueda en árboles que se utiliza en juegos de dos o más jugadores y deterministas. El algoritmo Minimax se basa en la idea de que los jugadores intentan maximizar su puntaje, mientras que sus oponentes intentan minimizarlo, es decir, juegos de suma cero. Este algoritmo es especialmente útil cuando el oponente toma siempre la mejor decisión posible.

El algoritmo Minimax funciona de la siguiente manera:

1. Se construye un árbol de búsqueda con todos los posibles movimientos.
2. Cada nodo del árbol representa un estado del juego y cada rama representa una acción.
3. El árbol alterna entre los jugadores, uno intenta maximizar el puntaje y los otros minimizarlo.
4. Se evalúan los nodos terminales del árbol y se asigna un valor usando una función de evaluación.
5. Dependiendo del jugador, se selecciona el nodo hijo con el valor máximo o mínimo.

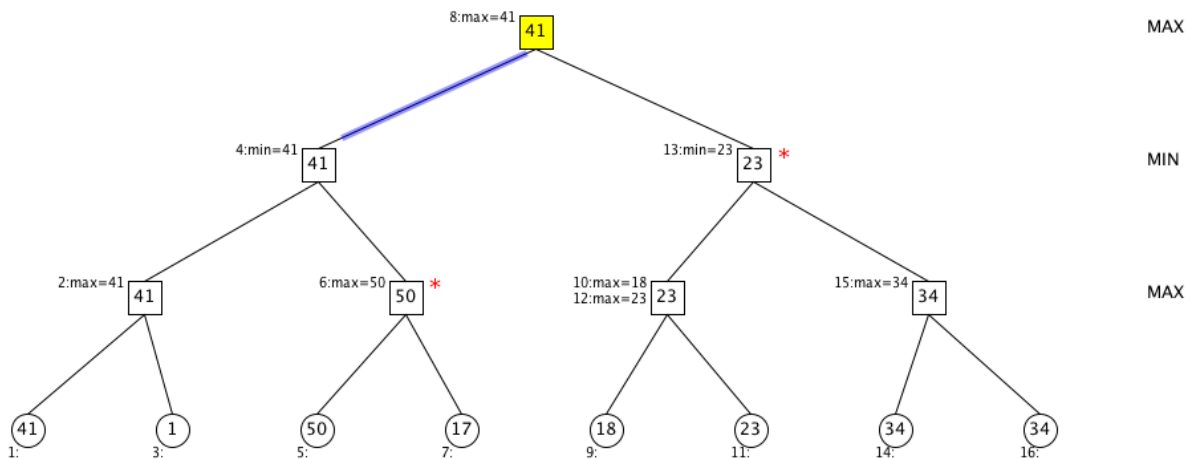


Figura 2.1: Ejemplo de árbol de búsqueda Minimax

Respecto al coste tanto en tiempo como en espacio, el algoritmo Minimax es, en esencia un DFS por lo que su coste en tiempo es $O(b^m)$ y en espacio $O(bm)$. Estos malos costes se deben a que el algoritmo explora todo el árbol de búsqueda, lo que lo hace inviable en juegos con un gran factor de ramificación y profundidad.

Primera implementación

```
1 class MinimaxAgent(MultiAgentSearchAgent):
2     """
3     Agente que implementa el algoritmo Minimax.
4     """
5
6     def getAction(self, gameState):
7         """
8         Devuelve la mejor acción para Pacman desde el estado actual 'gameState' usando Minimax.
9         """
10        # Llama a la función minimax empezando con el agente 0 (Pacman) y profundidad 0
11        best_action, _ = self.minimax(gameState, agentIndex=0, depth=0)
12        return best_action
13
14    def minimax(self, gameState, agentIndex, depth):
15        """
16        Función minimax que devuelve la mejor acción y su valor para el agente actual.
17        """
```



```

18     # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos el
    estado
19     if gameState.isWin() or gameState.isLose() or depth == self.depth:
20         return None, self.evaluationFunction(gameState)
21
22     if agentIndex == 0: # Pacman - Maximizador
23         return self.max_value(gameState, agentIndex, depth)
24     else:                # Fantasmas - Minimizadores
25         return self.min_value(gameState, agentIndex, depth)
26
27 def max_value(self, gameState, agentIndex, depth):
28     """
29     Calcula el valor máximo para el agente Pacman (maximizador).
30     """
31     # Inicializar el mejor valor y la mejor acción
32     best_value = float('-inf')
33     best_action = None
34
35     # Recorre todas las acciones legales para Pacman
36     for action in gameState.getLegalActions(agentIndex):
37         # Generar el estado sucesor
38         successorState = gameState.generateSuccessor(agentIndex, action)
39         # Calcular el valor del sucesor usando minimax con el siguiente agente
40         _, successor_value = self.minimax(successorState, 1, depth)
41         # Actualiza el valor máximo si se encuentra un mejor valor
42         if successor_value > best_value:
43             best_value = successor_value
44             best_action = action
45
46     return best_action, best_value
47
48 def min_value(self, gameState, agentIndex, depth):
49     """
50     Calcula el valor mínimo para los fantasmas (minimizador).
51     """
52     # Inicializar el peor valor y la mejor acción
53     worst_value = float('inf')
54     best_action = None
55
56     # Recorre todas las acciones legales para el fantasma actual
57     for action in gameState.getLegalActions(agentIndex):
58         # Generar el estado sucesor
59         successorState = gameState.generateSuccessor(agentIndex, action)
60         # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
61         _, successor_value = self.minimax(successorState, 0, depth + 1)
62
63         # Actualiza el valor mínimo si se encuentra un peor valor
64         if successor_value < worst_value:
65             worst_value = successor_value
66             best_action = action
67
68     return best_action, worst_value
69
70

```

Código 2.3: Implementación inicial del agente Minimax

Implementación final

```

1 class MinimaxAgent(MultiAgentSearchAgent):
2     """
3     Agente que implementa el algoritmo Minimax.
4     """
5
6     def getAction(self, gameState):
7         """
8         Devuelve la mejor acción para Pacman desde el estado actual 'gameState' usando Minimax.
9         """
10        # Llama a la función minimax empezando con el agente 0 (Pacman) y profundidad 0

```

```

11     best_action, _ = self.minimax(gameState, agentIndex=0, depth=0)
12     return best_action
13
14 def minimax(self, gameState, agentIndex, depth):
15     """
16     Función minimax que devuelve la mejor acción y su valor para el agente actual.
17     """
18     # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos el
    estado
19     if gameState.isWin() or gameState.isLose() or depth == self.depth:
20         return None, self.evaluationFunction(gameState)
21
22     if agentIndex == 0: # Pacman - Maximizador
23         return self.max_value(gameState, agentIndex, depth)
24     else:                # Fantasmas - Minimizadores
25         return self.min_value(gameState, agentIndex, depth)
26
27 def max_value(self, gameState, agentIndex, depth):
28     """
29     Calcula el valor máximo para el agente Pacman (maximizador).
30     """
31     # Inicializar el mejor valor y la mejor acción
32     best_value = float('-inf')
33     best_action = None
34
35     # Recorre todas las acciones legales para Pacman
36     for action in gameState.getLegalActions(agentIndex):
37         # Generar el estado sucesor
38         successorState = gameState.generateSuccessor(agentIndex, action)
39         # Calcular el valor del sucesor usando minimax con el siguiente agente
40         _, successor_value = self.minimax(successorState, agentIndex + 1, depth)
41         # Actualiza el valor máximo si se encuentra un mejor valor
42         if successor_value > best_value:
43             best_value = successor_value
44             best_action = action
45
46     return best_action, best_value
47
48 def min_value(self, gameState, agentIndex, depth):
49     """
50     Calcula el valor mínimo para los fantasmas (minimizador).
51     """
52     # Inicializar el peor valor y la mejor acción
53     worst_value = float('inf')
54     best_action = None
55
56     # Recorre todas las acciones legales para el fantasma actual
57     for action in gameState.getLegalActions(agentIndex):
58         # Generar el estado sucesor
59         successorState = gameState.generateSuccessor(agentIndex, action)
60
61         # Si es el último fantasma, pasamos a Pacman incrementando la profundidad
62         if agentIndex == gameState.getNumAgents() - 1:
63             # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
64             _, successor_value = self.minimax(successorState, 0, depth + 1)
65         else:
66             # Calcula el valor del sucesor con el siguiente fantasma
67             _, successor_value = self.minimax(successorState, agentIndex + 1, depth)
68
69         # Actualiza el valor mínimo si se encuentra un peor valor
70         if successor_value < worst_value:
71             worst_value = successor_value
72             best_action = action
73
74     return best_action, worst_value
75
76

```

Código 2.4: Implementación final del agente Minimax

Comentarios

En la primera implementación no he tenido en cuenta el número de agentes, por lo que el algoritmo solo funcionaba con dos agentes. En la implementación final, he tenido en cuenta el número de agentes y he modificado la función `min_value` para que si el agente actual es el último fantasma, pase a Pacman y aumente la profundidad.

2.3. Podado Alpha-Beta

Descripción

El algoritmo Alpha-Beta es una mejora del algoritmo Minimax que reduce el número de nodos evaluados en el árbol de búsqueda. Alpha-Beta mantiene dos valores, alpha y beta, que representan los valores de los nodos máximos y mínimos encontrados hasta el momento, respectivamente. El algoritmo poda las ramas del árbol que no afectarán la decisión final, es decir, las ramas que no cambiarán el valor de la raíz del árbol. El algoritmo Alpha-Beta es especialmente útil en juegos con muchos nodos y profundidad, ya que reduce el tiempo de búsqueda en el árbol.

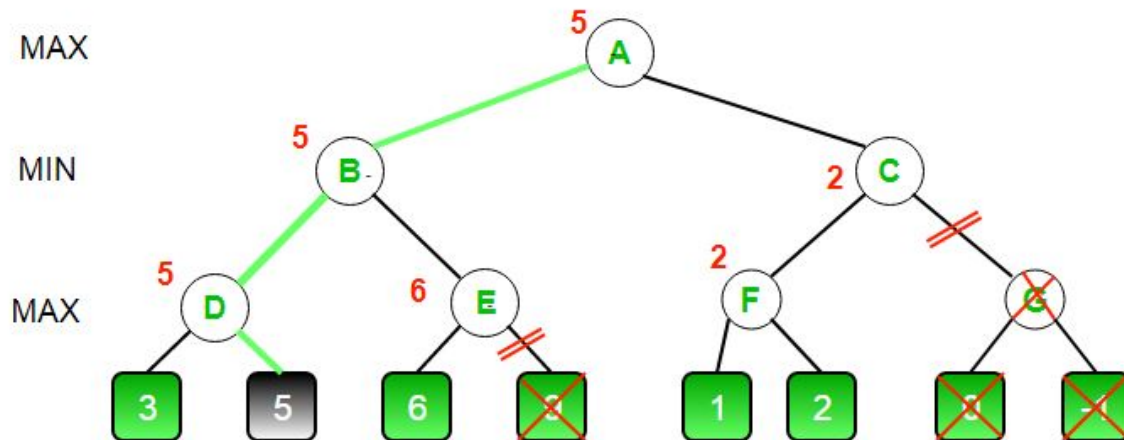


Figura 2.2: Ejemplo de poda Alpha-Beta

Primera implementación

1

Código 2.5: Implementación inicial del agente Alpha-Beta

Implementación final

```
1 class AlphaBetaAgent(MultiAgentSearchAgent):
2     """
3     Implementacion del minimax con poda alfa-beta.
4     """
5
6     def getAction(self, gameState):
7         """
8         Devuelve la mejor acción para Pacman desde el estado actual 'gameState' usando Minimax.
9         """
10        # Llama a la función minimax empezando con el agente 0 (Pacman) y profundidad 0
11        best_action, _ = self.minimax(gameState, agentIndex=0, depth=0, alpha=float('-inf'), beta=
float('inf'))
12        return best_action
13
14    def minimax(self, gameState, agentIndex, depth, alpha, beta):
15        """
16        Función minimax que devuelve la mejor acción y su valor para el agente actual.
17        """
18        # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos
el estado
19        if gameState.isWin() or gameState.isLose() or depth == self.depth:
20            return None, self.evaluationFunction(gameState)
21
22        if agentIndex == 0: # Pacman - Maximizador
23            return self.max_value(gameState, agentIndex, depth, alpha, beta)
```

```

24         else:                                # Fantasmas - Minimizadores
25             return self.min_value(gameState, agentIndex, depth, alpha, beta)
26
27     def max_value(self, gameState, agentIndex, depth, alpha, beta):
28         """
29         Calcula el valor máximo para el agente Pacman (maximizador).
30         """
31         # Inicializar el mejor valor y la mejor acción
32         best_value = float('-inf')
33         best_action = None
34
35         # Recorre todas las acciones legales para Pacman
36         for action in gameState.getLegalActions(agentIndex):
37             # Generar el estado sucesor
38             successorState = gameState.generateSuccessor(agentIndex, action)
39             # Calcular el valor del sucesor usando minimax con el siguiente agente
40             _, successor_value = self.minimax(successorState, agentIndex + 1, depth, alpha, beta)
41             # Actualiza el valor máximo si se encuentra un mejor valor
42             if successor_value > best_value:
43                 best_value = successor_value
44                 best_action = action
45
46             if best_value > beta:
47                 return best_action, best_value
48             alpha = max(alpha, best_value)
49
50         return best_action, best_value
51
52     def min_value(self, gameState, agentIndex, depth, alpha, beta):
53         """
54         Calcula el valor mínimo para los fantasmas (minimizador).
55         """
56         # Inicializar el peor valor y la mejor acción
57         worst_value = float('inf')
58         best_action = None
59
60         # Recorre todas las acciones legales para el fantasma actual
61         for action in gameState.getLegalActions(agentIndex):
62             # Generar el estado sucesor
63             successorState = gameState.generateSuccessor(agentIndex, action)
64
65             # Si es el último fantasma, pasamos a Pacman incrementando la profundidad
66             if agentIndex == gameState.getNumAgents() - 1:
67                 # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
68                 _, successor_value = self.minimax(successorState, 0, depth + 1, alpha, beta)
69             else:
70                 # Calcula el valor del sucesor con el siguiente fantasma
71                 _, successor_value = self.minimax(successorState, agentIndex + 1, depth, alpha,
72 beta)
73
74             # Actualiza el valor mínimo si se encuentra un peor valor
75             if successor_value < worst_value:
76                 worst_value = successor_value
77                 best_action = action
78
79             if worst_value < alpha:
80                 return best_action, worst_value
81             beta = min(beta, worst_value)
82
83         return best_action, worst_value

```

Código 2.6: Implementación final del agente Alpha-Beta

Comentarios

2.4. Expectimax

Descripción

El algoritmo Expectimax es una variante del algoritmo Minimax que se utiliza cuando los oponentes no toman siempre la mejor decisión posible. En lugar de minimizar el valor de los nodos de los oponentes, el algoritmo Expectimax toma la media de los valores de los nodos sucesores.

Primera implementación

```
1 class ExpectimaxAgent(MultiAgentSearchAgent):
2     """
3     Implementacion del algoritmo Expectimax.
4     """
5
6     def getAction(self, gameState):
7         return self.expectimax(gameState, agentIndex=0, depth=0)
8
9     def expectimax(self, gameState, agentIndex, depth):
10        """
11        Función expectimax que devuelve la mejor acción y su valor para el agente actual.
12        """
13        # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos
14        # el estado
15        if gameState.isWin() or gameState.isLose() or depth == self.depth:
16            return None, self.evaluationFunction(gameState)
17
18        if agentIndex == 0: # Pacman - Maximizador
19            return self.max_value(gameState, agentIndex, depth)
20        else: # Fantasmas - Minimizadores
21            return self.exp_value(gameState, agentIndex, depth)
22
23    def max_value(self, gameState, agentIndex, depth):
24        """
25        Calcula el valor máximo para el agente Pacman (maximizador).
26        """
27        # Inicializar el mejor valor y la mejor acción
28        best_value = float('-inf')
29        best_action = None
30
31        # Recorre todas las acciones legales para Pacman
32        for action in gameState.getLegalActions(agentIndex):
33            # Generar el estado sucesor
34            successorState = gameState.generateSuccessor(agentIndex, action)
35            # Calcular el valor del sucesor usando expectimax con el siguiente agente
36            _, successor_value = self.expectimax(successorState, agentIndex + 1, depth)
37            # Actualiza el valor máximo si se encuentra un mejor valor
38            if successor_value > best_value:
39                best_value = successor_value
40                best_action = action
41
42        return best_action, best_value
43
44    def exp_value(self, gameState, agentIndex, depth):
45        """
46        Calcula el valor esperado para los fantasmas (minimizador).
47        """
48        # Inicializar el valor esperado y la mejor acción
49        expected_value = 0
50        best_action = None
51
52        # Recorre todas las acciones legales para el fantasma actual
53        for action in gameState.getLegalActions(agentIndex):
54            # Generar el estado sucesor
55            successorState = gameState.generateSuccessor(agentIndex, action)
56
57            # Si es el último fantasma, pasamos a Pacman incrementando la profundidad
58            if agentIndex == gameState.getNumAgents() - 1:
```

```

58         # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
59         _, successor_value = self.expectimax(successorState, 0, depth + 1)
60     else:
61         # Calcula el valor del sucesor con el siguiente fantasma
62         _, successor_value = self.expectimax(successorState, agentIndex + 1, depth)
63
64     # Actualiza el valor esperado con el valor del sucesor
65     expected_value += successor_value / len(gameState.getLegalActions(agentIndex))
66
67     return best_action, expected_value
68

```

Código 2.7: Implementación inicial del agente Expectimax

Implementación final

```

1 class ExpectimaxAgent(MultiAgentSearchAgent):
2     """
3     Implementacion del algoritmo Expectimax.
4     """
5
6     def getAction(self, gameState):
7         best_action, _ = self.expectimax(gameState, agentIndex=0, depth=0)
8         return best_action
9
10    def expectimax(self, gameState, agentIndex, depth):
11        """
12        Función expectimax que devuelve la mejor acción y su valor para el agente actual.
13        """
14        # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos
15        # el estado
16        if gameState.isWin() or gameState.isLose() or depth == self.depth:
17            return None, self.evaluationFunction(gameState)
18
19        if agentIndex == 0: # Pacman - Maximizador
20            return self.max_value(gameState, agentIndex, depth)
21        else: # Fantasmas - Minimizadores
22            return self.exp_value(gameState, agentIndex, depth)
23
24    def max_value(self, gameState, agentIndex, depth):
25        """
26        Calcula el valor máximo para el agente Pacman (maximizador).
27        """
28        # Inicializar el mejor valor y la mejor acción
29        best_value = float('-inf')
30        best_action = None
31
32        # Recorre todas las acciones legales para Pacman
33        for action in gameState.getLegalActions(agentIndex):
34            # Generar el estado sucesor
35            successorState = gameState.generateSuccessor(agentIndex, action)
36            # Calcular el valor del sucesor usando expectimax con el siguiente agente
37            _, successor_value = self.expectimax(successorState, agentIndex + 1, depth)
38            # Actualiza el valor máximo si se encuentra un mejor valor
39            if successor_value > best_value:
40                best_value = successor_value
41                best_action = action
42
43        return best_action, best_value
44
45    def exp_value(self, gameState, agentIndex, depth):
46        """
47        Calcula el valor esperado para los fantasmas (minimizador).
48        """
49        # Inicializar el valor esperado y la mejor acción
50        expected_value = 0
51        best_action = None
52
53        # Recorre todas las acciones legales para el fantasma actual

```

```

53     actions = gameState.getLegalActions(agentIndex)
54     for action in actions:
55         # Generar el estado sucesor
56         successorState = gameState.generateSuccessor(agentIndex, action)
57
58         # Si es el último fantasma, pasamos a Pacman incrementando la profundidad
59         if agentIndex == gameState.getNumAgents() - 1:
60             # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
61             _, successor_value = self.expectimax(successorState, 0, depth + 1)
62         else:
63             # Calcula el valor del sucesor con el siguiente fantasma
64             _, successor_value = self.expectimax(successorState, agentIndex + 1, depth)
65         # Actualiza el valor esperado con el valor del sucesor
66         expected_value += successor_value / len(actions)
67
68     return best_action, expected_value
69

```

Código 2.8: Implementación final del agente Expectimax

Comentarios

2.5. Función de Evaluación

Descripción

Este ejercicio, consiste en mejorar la función de evaluación del agente reflex. La función de evaluación asigna un valor numérico a cada estado, y el agente selecciona la acción que maximiza este valor. Para mejorar la implementación previa únicamente se han añadido más factores a tener en cuenta en la evaluación de los estados. Para asignar un valor a cada estado, la función de evaluación considera varios factores, como la distancia a la comida, la proximidad a los fantasmas y la cantidad de comida restante.

Primera implementación

Código 2.9: Implementación inicial de la función de evaluación

Implementación final

```
1 def betterEvaluationFunction(currentGameState):
2     """
3     Función de evaluación para Pacman que considera múltiples factores del estado de juego.
4     """
5
6     # Obtener el estado actual de Pacman
7     pacmanPos = currentGameState.getPacmanPosition()
8
9     # Obtener la comida restante y convertirla a una lista de posiciones
10    food = currentGameState.getFood().asList()
11
12    # Obtener las posiciones de los fantasmas y sus estados (asustados o no)
13    ghostStates = currentGameState.getGhostStates()
14    ghostPositions = [ghost.getPosition() for ghost in ghostStates]
15    scaredTimes = [ghostState.scaredTimer for ghostState in ghostStates]
16
17    # Obtener las pelets de poder restantes
18    capsules = currentGameState.getCapsules()
19
20    # Obtener el puntaje actual del juego
21    score = currentGameState.getScore()
22
23    # Inicializar la evaluación con el puntaje actual
24    evaluation = score
25
26    # 1. Distancia a la comida: Minimizar la distancia a la comida
27    if food:
28        minFoodDist = min([manhattanDistance(pacmanPos, foodPos) for foodPos in food])
29        # Dar un peso inversamente proporcional a la distancia a la comida
30        evaluation += 1.0 / (minFoodDist + 1) # Sumar al score, +1 para evitar división por 0
31
32    # 2. Distancia a los fantasmas: Maximizar la distancia a los fantasmas (si no están asustados)
33    for i, ghostPos in enumerate(ghostPositions):
34        ghostDist = manhattanDistance(pacmanPos, ghostPos)
35
36        if scaredTimes[i] > 0: # Fantasma asustado
37            # Acercarse a los fantasmas asustados para ganar puntos al comérselos
38            evaluation += 10.0 / (ghostDist + 1) # Dar un peso a la proximidad a fantasmas
39        else: # Fantasma no asustado
40            # Alejarse de los fantasmas si están demasiado cerca
41            if ghostDist > 0:
42                evaluation -= 10.0 / ghostDist # Penalizar por estar cerca de un fantasma
43            peligroso
44
45    # 3. Comida restante: Cuanta menos comida quede, mejor es el estado
46    evaluation -= 4.0 * len(food) # Penalizar más comida restante
```

```

47     # 4. Cápsulas de poder: Incentivar estar cerca de las pelets de poder
48     if capsules:
49         minCapsuleDist = min([manhattanDistance(pacmanPos, capsule) for capsule in capsules])
50         evaluation += 5.0 / (minCapsuleDist + 1) # Dar un peso inverso a la distancia a las
51         pelets
52         evaluation -= 100 * len(capsules) # Penalizar si quedan muchas pelets sin recoger
53
54     # Devolver la evaluación final ajustada por todos los factores
55     return evaluation
56

```

Código 2.10: Implementación final de la función de evaluación

Comentarios

3. Resultados

3.1. Casos de prueba

3.2. Autograder

```
1 Starting on 10-15 at 18:32:26
2
3 Question q1
4 =====
5
6 Pacman emerges victorious! Score: 1238
7 Pacman emerges victorious! Score: 1244
8 Pacman emerges victorious! Score: 1239
9 Pacman emerges victorious! Score: 1235
10 Pacman emerges victorious! Score: 1233
11 Pacman emerges victorious! Score: 1241
12 Pacman emerges victorious! Score: 1246
13 Pacman emerges victorious! Score: 1242
14 Pacman emerges victorious! Score: 1239
15 Pacman emerges victorious! Score: 1242
16 Average Score: 1239.9
17 Scores: 1238.0, 1244.0, 1239.0, 1235.0, 1233.0, 1241.0, 1246.0, 1242.0, 1239.0, 1242.0
18 Win Rate: 10/10 (1.00)
19 Record: Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
20 *** PASS: test_cases/q1/grade-agent.test (4 of 4 points)
21 *** 1239.9 average score (2 of 2 points)
22 *** Grading scheme:
23 *** < 500: 0 points
24 *** >= 500: 1 points
25 *** >= 1000: 2 points
26 *** 10 games not timed out (0 of 0 points)
27 *** Grading scheme:
28 *** < 10: fail
29 *** >= 10: 0 points
30 *** 10 wins (2 of 2 points)
31 *** Grading scheme:
32 *** < 1: fail
33 *** >= 1: 0 points
34 *** >= 5: 1 points
35 *** >= 10: 2 points
36
37 ### Question q1: 4/4 ###
38
39
40 Question q2
41 =====
42
43 *** PASS: test_cases/q2/0-eval-function-lose-states-1.test
44 *** PASS: test_cases/q2/0-eval-function-lose-states-2.test
45 *** PASS: test_cases/q2/0-eval-function-win-states-1.test
46 *** PASS: test_cases/q2/0-eval-function-win-states-2.test
47 *** PASS: test_cases/q2/0-lecture-6-tree.test
48 *** PASS: test_cases/q2/0-small-tree.test
49 *** PASS: test_cases/q2/1-1-minmax.test
50 *** PASS: test_cases/q2/1-2-minmax.test
51 *** PASS: test_cases/q2/1-3-minmax.test
52 *** PASS: test_cases/q2/1-4-minmax.test
53 *** PASS: test_cases/q2/1-5-minmax.test
54 *** PASS: test_cases/q2/1-6-minmax.test
55 *** PASS: test_cases/q2/1-7-minmax.test
56 *** PASS: test_cases/q2/1-8-minmax.test
57 *** PASS: test_cases/q2/2-1a-vary-depth.test
58 *** PASS: test_cases/q2/2-1b-vary-depth.test
59 *** PASS: test_cases/q2/2-2a-vary-depth.test
60 *** PASS: test_cases/q2/2-2b-vary-depth.test
61 *** PASS: test_cases/q2/2-3a-vary-depth.test
62 *** PASS: test_cases/q2/2-3b-vary-depth.test
63 *** PASS: test_cases/q2/2-4a-vary-depth.test
64 *** PASS: test_cases/q2/2-4b-vary-depth.test
65 *** PASS: test_cases/q2/2-one-ghost-3level.test
66 *** PASS: test_cases/q2/3-one-ghost-4level.test
67 *** PASS: test_cases/q2/4-two-ghosts-3level.test
68 *** PASS: test_cases/q2/5-two-ghosts-4level.test
```

```

69 *** PASS: test_cases/q2/6-tied-root.test
70 *** PASS: test_cases/q2/7-1a-check-depth-one-ghost.test
71 *** PASS: test_cases/q2/7-1b-check-depth-one-ghost.test
72 *** PASS: test_cases/q2/7-1c-check-depth-one-ghost.test
73 *** PASS: test_cases/q2/7-2a-check-depth-two-ghosts.test
74 *** PASS: test_cases/q2/7-2b-check-depth-two-ghosts.test
75 *** PASS: test_cases/q2/7-2c-check-depth-two-ghosts.test
76 *** Running MinimaxAgent on smallClassic 1 time(s).
77 Pacman died! Score: 84
78 Average Score: 84.0
79 Scores:      84.0
80 Win Rate:    0/1 (0.00)
81 Record:      Loss
82 *** Finished running MinimaxAgent on smallClassic after 0.7832581996917725 seconds.
83 *** Won 0 out of 1 games. Average score: 84.0 ***
84 *** PASS: test_cases/q2/8-pacman-game.test
85
86 ### Question q2: 5/5 ###
87
88
89 Question q3
90 =====
91
92 *** PASS: test_cases/q3/0-eval-function-lose-states-1.test
93 *** PASS: test_cases/q3/0-eval-function-lose-states-2.test
94 *** PASS: test_cases/q3/0-eval-function-win-states-1.test
95 *** PASS: test_cases/q3/0-eval-function-win-states-2.test
96 *** PASS: test_cases/q3/0-lecture-6-tree.test
97 *** PASS: test_cases/q3/0-small-tree.test
98 *** PASS: test_cases/q3/1-1-minmax.test
99 *** PASS: test_cases/q3/1-2-minmax.test
100 *** PASS: test_cases/q3/1-3-minmax.test
101 *** PASS: test_cases/q3/1-4-minmax.test
102 *** PASS: test_cases/q3/1-5-minmax.test
103 *** PASS: test_cases/q3/1-6-minmax.test
104 *** PASS: test_cases/q3/1-7-minmax.test
105 *** PASS: test_cases/q3/1-8-minmax.test
106 *** PASS: test_cases/q3/2-1a-vary-depth.test
107 *** PASS: test_cases/q3/2-1b-vary-depth.test
108 *** PASS: test_cases/q3/2-2a-vary-depth.test
109 *** PASS: test_cases/q3/2-2b-vary-depth.test
110 *** PASS: test_cases/q3/2-3a-vary-depth.test
111 *** PASS: test_cases/q3/2-3b-vary-depth.test
112 *** PASS: test_cases/q3/2-4a-vary-depth.test
113 *** PASS: test_cases/q3/2-4b-vary-depth.test
114 *** PASS: test_cases/q3/2-one-ghost-3level.test
115 *** PASS: test_cases/q3/3-one-ghost-4level.test
116 *** PASS: test_cases/q3/4-two-ghosts-3level.test
117 *** PASS: test_cases/q3/5-two-ghosts-4level.test
118 *** PASS: test_cases/q3/6-tied-root.test
119 *** PASS: test_cases/q3/7-1a-check-depth-one-ghost.test
120 *** PASS: test_cases/q3/7-1b-check-depth-one-ghost.test
121 *** PASS: test_cases/q3/7-1c-check-depth-one-ghost.test
122 *** PASS: test_cases/q3/7-2a-check-depth-two-ghosts.test
123 *** PASS: test_cases/q3/7-2b-check-depth-two-ghosts.test
124 *** PASS: test_cases/q3/7-2c-check-depth-two-ghosts.test
125 *** Running AlphaBetaAgent on smallClassic 1 time(s).
126 Pacman died! Score: 84
127 Average Score: 84.0
128 Scores:      84.0
129 Win Rate:    0/1 (0.00)
130 Record:      Loss
131 *** Finished running AlphaBetaAgent on smallClassic after 0.7023296356201172 seconds.
132 *** Won 0 out of 1 games. Average score: 84.0 ***
133 *** PASS: test_cases/q3/8-pacman-game.test
134
135 ### Question q3: 5/5 ###
136
137
138 Question q4
139 =====

```

```

140
141 *** PASS: test_cases/q4/0-eval-function-lose-states-1.test
142 *** PASS: test_cases/q4/0-eval-function-lose-states-2.test
143 *** PASS: test_cases/q4/0-eval-function-win-states-1.test
144 *** PASS: test_cases/q4/0-eval-function-win-states-2.test
145 *** PASS: test_cases/q4/0-expectimax1.test
146 *** PASS: test_cases/q4/1-expectimax2.test
147 *** PASS: test_cases/q4/2-one-ghost-3level.test
148 *** PASS: test_cases/q4/3-one-ghost-4level.test
149 *** PASS: test_cases/q4/4-two-ghosts-3level.test
150 *** PASS: test_cases/q4/5-two-ghosts-4level.test
151 *** PASS: test_cases/q4/6-1a-check-depth-one-ghost.test
152 *** PASS: test_cases/q4/6-1b-check-depth-one-ghost.test
153 *** PASS: test_cases/q4/6-1c-check-depth-one-ghost.test
154 *** PASS: test_cases/q4/6-2a-check-depth-two-ghosts.test
155 *** PASS: test_cases/q4/6-2b-check-depth-two-ghosts.test
156 *** PASS: test_cases/q4/6-2c-check-depth-two-ghosts.test
157 *** Running ExpectimaxAgent on smallClassic 1 time(s).
158 Pacman died! Score: 84
159 Average Score: 84.0
160 Scores:      84.0
161 Win Rate:    0/1 (0.00)
162 Record:     Loss
163 *** Finished running ExpectimaxAgent on smallClassic after 0.8156166076660156 seconds.
164 *** Won 0 out of 1 games. Average score: 84.0 ***
165 *** PASS: test_cases/q4/7-pacman-game.test
166
167 ### Question q4: 5/5 ###
168
169
170 Question q5
171 =====
172
173 Pacman emerges victorious! Score: 1366
174 Pacman emerges victorious! Score: 1026
175 Pacman emerges victorious! Score: 1060
176 Pacman emerges victorious! Score: 1089
177 Pacman emerges victorious! Score: 1015
178 Pacman emerges victorious! Score: 1016
179 Pacman emerges victorious! Score: 1238
180 Pacman emerges victorious! Score: 760
181 Pacman emerges victorious! Score: 1372
182 Pacman emerges victorious! Score: 1321
183 Average Score: 1126.3
184 Scores:      1366.0, 1026.0, 1060.0, 1089.0, 1015.0, 1016.0, 1238.0, 760.0, 1372.0, 1321.0
185 Win Rate:    10/10 (1.00)
186 Record:     Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
187 *** PASS: test_cases/q5/grade-agent.test (6 of 6 points)
188 ***      1126.3 average score (2 of 2 points)
189 ***      Grading scheme:
190 ***          < 500:  0 points
191 ***          >= 500:  1 points
192 ***          >= 1000: 2 points
193 ***      10 games not timed out (1 of 1 points)
194 ***      Grading scheme:
195 ***          < 0:  fail
196 ***          >= 0:  0 points
197 ***          >= 10:  1 points
198 ***      10 wins (3 of 3 points)
199 ***      Grading scheme:
200 ***          < 1:  fail
201 ***          >= 1:  1 points
202 ***          >= 5:  2 points
203 ***          >= 10: 3 points
204
205 ### Question q5: 6/6 ###
206
207
208 Finished at 18:32:43
209
210 Provisional grades

```

```
211 =====
212 Question q1: 4/4
213 Question q2: 5/5
214 Question q3: 5/5
215 Question q4: 5/5
216 Question q5: 6/6
217 -----
218 Total: 25/25
219
220 Your grades are NOT yet registered. To register your grades, make sure
221 to follow your instructor's guidelines to receive credit on your project.
222
223
224
```

Código 3.1: Resultados del Autograder