

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

13 de octubre de 2024

# Índice general

|   |           |
|---|-----------|
| <b>1. Introducción</b>                  | <b>4</b>  |
| <b>2. Ejercicios</b>                    | <b>5</b>  |
| 2.1. Agente Reflex . . . . .            | 5         |
| 2.1.1. Descripción . . . . .            | 5         |
| 2.1.2. Primera implementación . . . . . | 5         |
| 2.1.3. Implementación final . . . . .   | 5         |
| 2.1.4. Comentarios . . . . .            | 6         |
| 2.2. Minimax . . . . .                  | 7         |
| 2.2.1. Descripción . . . . .            | 7         |
| 2.2.2. Primera implementación . . . . . | 7         |
| 2.2.3. Implementación final . . . . .   | 8         |
| 2.2.4. Comentarios . . . . .            | 9         |
| 2.3. Podado Alpha-Beta . . . . .        | 10        |
| 2.4. Expectimax . . . . .               | 11        |
| 2.5. Función de Evaluación . . . . .    | 12        |
| <b>3. Resultados</b>                    | <b>13</b> |
| 3.1. Casos de prueba . . . . .          | 13        |
| 3.2. Autograder . . . . .               | 14        |

# Índice de figuras

# 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 |

# 1. Introducción

# 2. Ejercicios

## 2.1. Agente Reflex

### 2.1.1. Descripción

### 2.1.2. Primera implementación

1

Código 2.1: Implementación inicial del agente reflex

### 2.1.3. 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
    peligroso
54
55     # 3. Incentivo por acercarse a fantasmas asustados
56     for i, ghost in enumerate(newGhostStates):
57         if newScaredTimes[i] > 0: # Si el fantasma está asustado
58             ghostDistance = manhattanDistance(newPos, ghost.getPosition())
59             score += 200.0 / (ghostDistance + 1) # Premiar estar cerca de un fantasma asustado
60
61     # 4. Penalización por comida restante
62     score -= len(foodList) * 10 # Penalizar por cada comida restante en el estado sucesor
63
64     return score
65

```

Código 2.2: Implementación final del agente reflex

#### 2.1.4. Comentarios

## 2.2. Minimax

### 2.2.1. Descripción

### 2.2.2. 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
19        # estado
20        if gameState.isWin() or gameState.isLose() or depth == self.depth:
21            return None, self.evaluationFunction(gameState)
22
23        if agentIndex == 0: # Pacman - Maximizador
24            return self.max_value(gameState, agentIndex, depth)
25        else: # Fantasmas - Minimizadores
26            return self.min_value(gameState, agentIndex, depth)
27
28    def max_value(self, gameState, agentIndex, depth):
29        """
30        Calcula el valor máximo para el agente Pacman (maximizador).
31        """
32        # Inicializar el mejor valor y la mejor acción
33        best_value = float('-inf')
34        best_action = None
35
36        # Recorre todas las acciones legales para Pacman
37        for action in gameState.getLegalActions(agentIndex):
38            # Generar el estado sucesor
39            successorState = gameState.generateSuccessor(agentIndex, action)
40            # Calcular el valor del sucesor usando minimax con el siguiente agente
41            _, successor_value = self.minimax(successorState, 1, depth)
42            # Actualiza el valor máximo si se encuentra un mejor valor
43            if successor_value > best_value:
44                best_value = successor_value
45                best_action = action
46
47        return best_action, best_value
48
49    def min_value(self, gameState, agentIndex, depth):
50        """
51        Calcula el valor mínimo para los fantasmas (minimizador).
52        """
53        # Inicializar el peor valor y la mejor acción
54        worst_value = float('inf')
55        best_action = None
56
57        # Recorre todas las acciones legales para el fantasma actual
58        for action in gameState.getLegalActions(agentIndex):
59            # Generar el estado sucesor
60            successorState = gameState.generateSuccessor(agentIndex, action)
61            # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
62            _, successor_value = self.minimax(successorState, 0, depth + 1)
63
64            # 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

### 2.2.3. 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

#### 2.2.4. 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

## 2.4. Expectimax

## 2.5. Función de Evaluación

## 3. Resultados

### 3.1. Casos de prueba

## 3.2. Autograder