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

Índice de figuras

Índice de Códigos

1. Introducción

2. Ejercicios

2.1. Agente Reflex

2.1.1. Descripción

2.1.2. Primera implementación

```
1
```

Código 2.1: Implementación del agente reflex

2.1.3. Implementación final

```python
class ReflexAgent(Agent):
  """
  Un agente reflexivo que elige acciones basándose en una función de evaluación.
  """

  def getAction(self, gameState):
      """
      Devuelve la mejor acción para Pacman basada en la función de evaluación.
      """
      # Obtiene las acciones legales para Pacman
      legalMoves = gameState.getLegalActions()

      # Calcula los puntajes para cada acción utilizando la función de evaluación
      scores = [self.evaluationFunction(gameState, action) for action in legalMoves]
      bestScore = max(scores)  # Encuentra el puntaje más alto
      bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
      chosenIndex = random.choice(bestIndices)  # Elige aleatoriamente entre las mejores acciones

      return legalMoves[chosenIndex]

  def evaluationFunction(self, currentGameState, action):
      """
      Calcula el valor del estado sucesor después de que Pacman toma la acción `action`.
      Devuelve un valor numérico mayor para estados más favorables.
      """
      # Generar el estado sucesor
      successorGameState = currentGameState.generatePacmanSuccessor(action)
      # Obtiene la posición de Pacman después de moverse
      newPos = successorGameState.getPacmanPosition()
      # Obtiene la matriz de comida en el estado sucesor
      newFood = successorGameState.getFood()
      # Obtiene la lista de estados de los fantasmas
      newGhostStates = successorGameState.getGhostStates()
      # Obtiene los tiempos restantes de los fantasmas asustados
      newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

      # Inicializar el puntaje con el puntaje base del sucesor
      score = successorGameState.getScore()

      # 1. Distancia a la comida más cercana
      foodList = newFood.asList()  # Convertir la matriz de comida a una lista de posiciones
      if foodList:  # Si hay comida disponible
          # Calcular la distancia mínima a la comida más cercana
          minFoodDistance = min([manhattanDistance(newPos, food) for food in foodList])
          # Invertir la distancia para que un menor valor de distancia dé un mayor puntaje
          score += 10.0 / minFoodDistance

      # 2. Distancia a los fantasmas no asustados
      for ghost in newGhostStates:
          ghostPos = ghost.getPosition()
          ghostDistance = manhattanDistance(newPos, ghostPos)
          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

2.1.5. Comentarios

2.2. Minimax

2.2.1. Descripción

2.2.2. Primera implementación

```python
class MinimaxAgent(MultiAgentSearchAgent):
  """
  Agente que implementa el algoritmo Minimax.
  """

  def getAction(self, gameState):
      """
      Devuelve la mejor acción para Pacman desde el estado actual `gameState` usando Minimax.
      """
      # Llama a la función minimax empezando con el agente 0 (Pacman) y profundidad 0
      best_action, _ = self.minimax(gameState, agentIndex=0, depth=0)
      return best_action

  def minimax(self, gameState, agentIndex, depth):
      """
      Función minimax que devuelve la mejor acción y su valor para el agente actual.
      """
      # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos el
    estado
      if gameState.isWin() or gameState.isLose() or depth == self.depth:
          return None, self.evaluationFunction(gameState)

      if agentIndex == 0: # Pacman - Maximizador
          return self.max_value(gameState, agentIndex, depth)
      else:               # Fantasmas - Minimizadores
          return self.min_value(gameState, agentIndex, depth)

  def max_value(self, gameState, agentIndex, depth):
      """
      Calcula el valor máximo para el agente Pacman (maximizador).
      """
      # Inicializar el mejor valor y la mejor acción
      best_value = float('-inf')
      best_action = None

      # Recorre todas las acciones legales para Pacman
      for action in gameState.getLegalActions(agentIndex):
          # Generar el estado sucesor
          successorState = gameState.generateSuccessor(agentIndex, action)
          # Calcular el valor del sucesor usando minimax con el siguiente agente
          _, successor_value = self.minimax(successorState, 1, depth)
          # Actualiza el valor máximo si se encuentra un mejor valor
          if successor_value > best_value:
              best_value = successor_value
              best_action = action

      return best_action, best_value

  def min_value(self, gameState, agentIndex, depth):
      """
      Calcula el valor mínimo para los fantasmas (minimizador).
      """
      # Inicializar el peor valor y la mejor acción
      worst_value = float('inf')
      best_action = None

      # Recorre todas las acciones legales para el fantasma actual
      for action in gameState.getLegalActions(agentIndex):
          # Generar el estado sucesor
          successorState = gameState.generateSuccessor(agentIndex, action)
          # Calcula el valor del sucesor con Pacman y siguiente nivel de profundidad
          _, successor_value = self.minimax(successorState, 0, depth + 1)

          # Actualiza el valor mínimo si se encuentra un peor valor
```

```
            if successor_value < worst_value:
                worst_value = successor_value
                best_action = action

        return best_action, worst_value


```

Código 2.3: Implementación final del BFS

2.2.3. Implementación final

```
class MinimaxAgent(MultiAgentSearchAgent):
  """
  Agente que implementa el algoritmo Minimax.
  """

  def getAction(self, gameState):
      """
      Devuelve la mejor acción para Pacman desde el estado actual `gameState` usando Minimax.
      """
      # Llama a la función minimax empezando con el agente 0 (Pacman) y profundidad 0
      best_action, _ = self.minimax(gameState, agentIndex=0, depth=0)
      return best_action

  def minimax(self, gameState, agentIndex, depth):
      """
      Función minimax que devuelve la mejor acción y su valor para el agente actual.
      """
      # Si el estado es terminal (gana o pierde) o alcanzamos la profundidad máxima, evaluamos el
    estado
      if gameState.isWin() or gameState.isLose() or depth == self.depth:
          return None, self.evaluationFunction(gameState)

      if agentIndex == 0: # Pacman - Maximizador
          return self.max_value(gameState, agentIndex, depth)
      else:               # Fantasmas - Minimizadores
          return self.min_value(gameState, agentIndex, depth)

  def max_value(self, gameState, agentIndex, depth):
      """
      Calcula el valor máximo para el agente Pacman (maximizador).
      """
      # Inicializar el mejor valor y la mejor acción
      best_value = float('-inf')
      best_action = None

      # Recorre todas las acciones legales para Pacman
      for action in gameState.getLegalActions(agentIndex):
          # Generar el estado sucesor
          successorState = gameState.generateSuccessor(agentIndex, action)
          # Calcular el valor del sucesor usando minimax con el siguiente agente
          _, successor_value = self.minimax(successorState, agentIndex + 1, depth)
          # Actualiza el valor máximo si se encuentra un mejor valor
          if successor_value > best_value:
              best_value = successor_value
              best_action = action

      return best_action, best_value

  def min_value(self, gameState, agentIndex, depth):
      """
      Calcula el valor mínimo para los fantasmas (minimizador).
      """
      # Inicializar el peor valor y la mejor acción
      worst_value = float('inf')
      best_action = None

      # 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 BFS

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.

3. Resultados