BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

Técnicas de Inteligencia Artificial

Práctica 2 Pac-Man

2023-2024

...

Titulación:

Grado en Informática de Gestión y Sistemas de Información

...

4º Curso (1º Cuatrimestre)

...

Página de GitHub

Alan García, Álvaro Díez-Andino

3 de noviembre de 2023

Índice

1. Pregunta1: Agente Reflex

1.1. Descripción

Un algoritmo de evaluación es una herramienta fundamental para la toma de decisiones en juegos y otros escenarios. Su enfoque puede variar desde consideraciones inmediatas hasta estrategias a largo plazo, dependiendo de las necesidades específicas del juego o del sistema en cuestión. En nuestro caso se hará un enfoque cortoplacista del juego pacman donde se evaluará cada acción posible de PacMan.

Solamente necesita tener en cuenta lo siguiente para la puntuación:

- Distancia a la comida actual más cercana
- Distancia al fantasma de la siguiente acción
- Proporción de la distancia por comida

Es importante recalcar:

- Las distancias son inversos
- El inverso de la distancia a la comida suma puntuación.
- El inverso de la distancia al fantasma y la proporción distancia por comida resta puntuación.

1.1.1. Tipo de Algoritmo

- Algoritmo de **evaluación** usado para evaluar que tan beneficiosa o perjudicial es una acción. Es altamente cortoplacista, es decir no busca hacer una evaluación con el futuro en mente si no que únicamente con el futuro muy cercano.

1.2. Algoritmo

DFS(u)

Inicializar curFood con la lista de comida actual de GameState.

Inicializar posicionesNewComida como una lista vacía.

Inicializar posicionFantasmas como una lista vacía.

Inicializar distanciaTotalCom a 1.

Para cada comida en curFood:

Calcular la distancia de Manhattan entre newPos y comida.

Agregar la distancia calculada a posicionesNewComida.

Sumar la distancia calculada a distanciaTotalCom.

Calcular distNuevaCom como 1 dividido por la distancia mínima en posicionesNewComida más 1.

Para cada posición de los fantasmas en successorGameState:

Calcular la distancia de Manhattan entre newPos y la posición del fantasma.

Agregar la distancia calculada a posicionFantasmas.

Encontrar la posición mínima en posicionFantasmas y asignarla a posFant.

Si posFant es menor que 5, multiplicar fantMasCerc por 4.

Si no quedan comidas en newFood:

Devolver un valor muy grande, por ejemplo, 99999999, para indicar una victoria.

Calcular resultado como distNuevaCom menos fantMasCerc menos 0.1 multiplicado por distanciaTotalCom dividido por la cantidad de elementos en newFood.

Devolver 10000 veces el valor de resultado.

1.2.1. Código V1

```python
def evaluationFunction(self, currentGameState, action):
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in
    ↪ newGhostStates]
    " YOUR CODE HERE "
    # Usamos una funcion lambda con filter para obtener los valores True
    comidas = newFood.asList()
    comidas_disponibles = len(comidas)
    # Funcion lambda para calcular la distancia Manhatan entre newPos y
    ↪ las comidas
    dist = lambda x: util.manhattanDistance(newPos, x)
    dist_min = min(list(map(lambda c: dist(c), comidas)))


    value = 1 / ( comidas_disponibles + dist_min )
    print(f"sucesor: {action} tiene evaluacion: {value}")


    return value
```

Listing 1: Python example

```
Finished at 14:11:06

Provisional grades
==================
Question q1: 0/4
Question q2: 0/5
Question q3: 0/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 0/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 1: Primea versión Agente Reflex

1.2.2. Código V-Final

```python
def evaluationFunction(self, currentGameState, action):
    successorGameState = currentGameState.generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood().asList()
    newGhostStates = successorGameState.getGhostStates()
```

```
    newScaredTimes = [ghostState.scaredTimer for ghostState in
    ↪ newGhostStates]
    " YOUR CODE HERE "
    curFood=currentGameState.getFood().asList()
    posicionesNewComida = []
    posicionFantasmas = []
    distanciaTotalCom=1
    for comida in curFood:
        posicionesNewComida.append(util.manhattanDistance(newPos, comida))
        distanciaTotalCom+=util.manhattanDistance(newPos, comida)

    distNuevaCom = 1/(min(posicionesNewComida) + 1)
    for fantasmas in successorGameState.getGhostPositions():
        posicionFantasmas.append(util.manhattanDistance(newPos,fantasmas))
    posFant=min(posicionFantasmas)
    fantMasCerc = 1/(posFant+1)
    if(posFant<5):
        fantMasCerc=fantMasCerc*4

    if(len(newFood)==0):
        return(99999999)
    resultado=distNuevaCom - fantMasCerc-0.1*distanciaTotalCom/(len(
    ↪ newFood))

    return (10000*resultado) #por decimales
```

Listing 2: Python example

```
Finished at 14:22:19

Provisional grades
==================
Question q1: 4/4
Question q2: 0/5
Question q3: 0/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 4/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 2: Versión final DFS

1.2.3. Problemas y dificultades encarados en el ejercicio

El primer problema que tuvimos que afrontar era como lidiar en el momento de comer la comida, ya que, al realizarlo en el siguiente movimiento la distancia a la siguiente comida ahora sería mayor y, por ende, no favorecía

esa acción y además le causaba una preferencia a quedarse quieto puesto que así se quedaría al lado de la comida debido a que obtenía una mayor puntuación.
Para solucionarlo modificamos la distancia de la siguiente posición a la actual de manera que siempre favoreciera comer la comida en vez de quedarse al lado, en resumen, le hicimos una pequeña trampa a pacman para que no vea que en esa acción hubiera comido la comida evitando así que se quedara quieto.
Con la distancia al fantasma y la comida más cercana no obteníamos la puntuación máxima por lo que planeamos usar la proporción de distancia por comida resultante de una acción con el objetivo de premiar la menor distancia por comida, por eso lo añadimos restando a la puntuación.Con este ultimo cambio obtuvimos la puntuación máxima.

1.2.4. Ejemplo

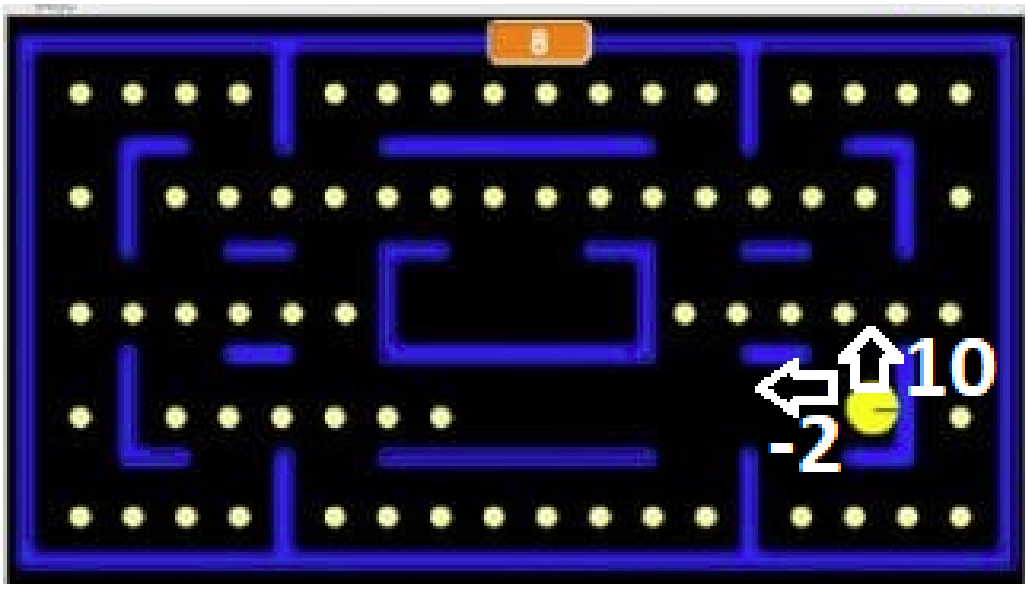

Figura 3: Ejemplo DFS

2. Pregunta2: Minimax

2.1. Descripción

El algoritmo Minimax es un enfoque fundamental en la toma de decisiones en juegos adversariales. Su objetivo es determinar la mejor estrategia para

un jugador, en nuestro caso PacMan, asumiendo que el oponente, en nuestro caso los fantasmas, juega de manera óptima para minimizar el valor máximo que el jugador puede obtener. El Minimax funciona explorando un árbol de juego, donde los nodos alternan entre representar al jugador (MAX) y al oponente (MIN). Solamente necesita:

- Función recursiva que obtenga la mínima puntuación de sus hijos.
- Función recursiva que obtenga la máxima puntuación de sus hijos.

Es importante recalcar:

- Los fantasmas buscarán minimizar la puntuación y habrá más de uno
- Pacma buscará maximizar la puntuación y solo habrá uno.
- El indice de los agentes comienza en Pacman y finaliza en el ultimo fantasma, este recorrido debe ser ascendente

2.1.1. Tipo de Algoritmo

- Algoritmo MinMax **no informado**

2.2. Algoritmo

DFS(u)

Función maxvalue(curState, agentIndex, numFant, profundidad):

Si profundidad es 0 o curState es victoria o derrota:

Devolver evaluación de curState

Inicializar puntos a un valor muy pequeño

Obtener acciones legales para agentIndex en curState

Para cada acción en acciones:

Calcular puntos como el máximo entre puntos y el resultado de minvalue para el siguiente estado

Devolver puntos

Función minvalue(curState, agentIndex, numFant, profundidad):

Si profundidad es 0 o curState es victoria o derrota:

Devolver evaluación de curState

Obtener acciones legales para agentIndex en curState

Inicializar puntos a un valor muy grande

Para cada acción en acciones:

Calcular puntos como el mínimo entre puntos y el resultado de maxvalue para el siguiente estado

Devolver puntos

Inicializar puntos y optAction con valores iniciales

Obtener el número de agentes en el juego y almacenarlo en numFant

Obtener las acciones legales para el agente 0 en el estado actual

Para cada acción en acciones:

Calcular curPunts como el resultado de minvalue para el siguiente estado

Actualizar puntos y optAction si curPunts es mayor que puntos

Devolver optAction como la mejor acción a tomar

2.2.1. Código V1

```python
def maxvalue(curState,agentIndex, numFant, profundidad ):
  #Si llegamos al final
  if profundidad == 0 or curState.isWin() or curState.isLose():
    return self.evaluationFunction(curState)
  puntos = -9999
  acciones= curState.getLegalActions(agentIndex)
  for accion in acciones:
    puntos = max(puntos, minvalue(curState.generateSuccessor(0, accion), numFant, numFant, profundidad))
  return puntos

def minvalue(curState,agentIndex, numFant, profundidad):
  #Si llegamos al final
  if profundidad == 0 or curState.isWin() or curState.isLose():
    return self.evaluationFunction(curState)

  acciones = curState.getLegalActions( agentIndex)
  puntos = 9999
  for accion in acciones:
    if  agentIndex == 1:
      puntos = min(puntos, maxvalue(curState.generateSuccessor( agentIndex, accion), 0, numFant, profundidad - 1))
    else:
      puntos = min(puntos, minvalue(curState.generateSuccessor( agentIndex, accion),agentIndex - 1, numFant, profundidad ))
  return puntos

puntos, optAction = -9999, None
numFant = game_state.getNumAgents()- 1
acciones = game_state.getLegalActions(0)
for accion in acciones:
  curPunts = minvalue(game_state.generateSuccessor(0, accion), numFant, numFant, self.depth)
  if curPunts > puntos:
    puntos, optAction = curPunts, accion
return optAction
```

Figura 4: Primera versión MinMax

```
Finished at 19:35:35

Provisional grades
==================
Question q1: 4/4
Question q2: 0/5
Question q3: 0/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 4/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Figura 5: Primea versión Agente MinMax

2.2.2. Código V-Final

```
def maxvalue(estadoAct,agentIndex, numFant, profundidad ):
  #Si llegamos al final
  if profundidad == 0 or estadoAct.isWin() or estadoAct.isLose():
    return self.evaluationFunction(estadoAct)
  puntos = -9999999
  acciones= estadoAct.getLegalActions(agentIndex)
  for accion in acciones:
    puntos = max(puntos, minvalue(estadoAct.generateSuccessor(agentIndex, accion), agentIndex+1, numFant, profundidad))
  return puntos

def minvalue(estadoAct,agentIndex, numFant, profundidad):
  #Si llegamos al final
  if profundidad == 0 or estadoAct.isWin() or estadoAct.isLose():
    return self.evaluationFunction(estadoAct)

  acciones = estadoAct.getLegalActions(agentIndex)
  puntos = 9999
  for accion in acciones:
    if  agentIndex == numFant:
      puntos = min(puntos, maxvalue(estadoAct.generateSuccessor( agentIndex, accion), 0, numFant, profundidad-1))
    else:
      puntos = min(puntos, minvalue(estadoAct.generateSuccessor( agentIndex, accion),agentIndex + 1, numFant, profundidad ))
  return puntos

puntos, optAction = -99999, None
numFant = game_state.getNumAgents()- 1
acciones = game_state.getLegalActions(0)
for accion in acciones:
  curPunts = minvalue(game_state.generateSuccessor(0, accion), 1, numFant, self.depth)
  if curPunts > puntos:
    puntos, optAction = curPunts, accion
return optAction
```

Figura 6: MinMax versión final

```
Finished at 14:37:21

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 0/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 9/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 7: Versión final DFS

2.2.3. Problemas y dificultades encarados en el ejercicio

A la hora de implementar minmax no hubo mucho problema al partir del código que se ha impartido en clase, el mayor problema vino de que, a la hora de recorrer los agentes, empezamos por el ultimo fantasma para finalizar en pacman esto resultó en invertir, o más bien gastar, una cantidad relativa de horas buscando el motivo para, posteriormente, leyendo la documentación de la practica, ver que el orden de los agentes debe ser ascendente empezando por pacman y finalizando en el fantasma a diferencia de como estaba implementado al inicio. Tras cambiar el orden obtuvimos la máxima puntuación.

2.2.4. Ejemplo

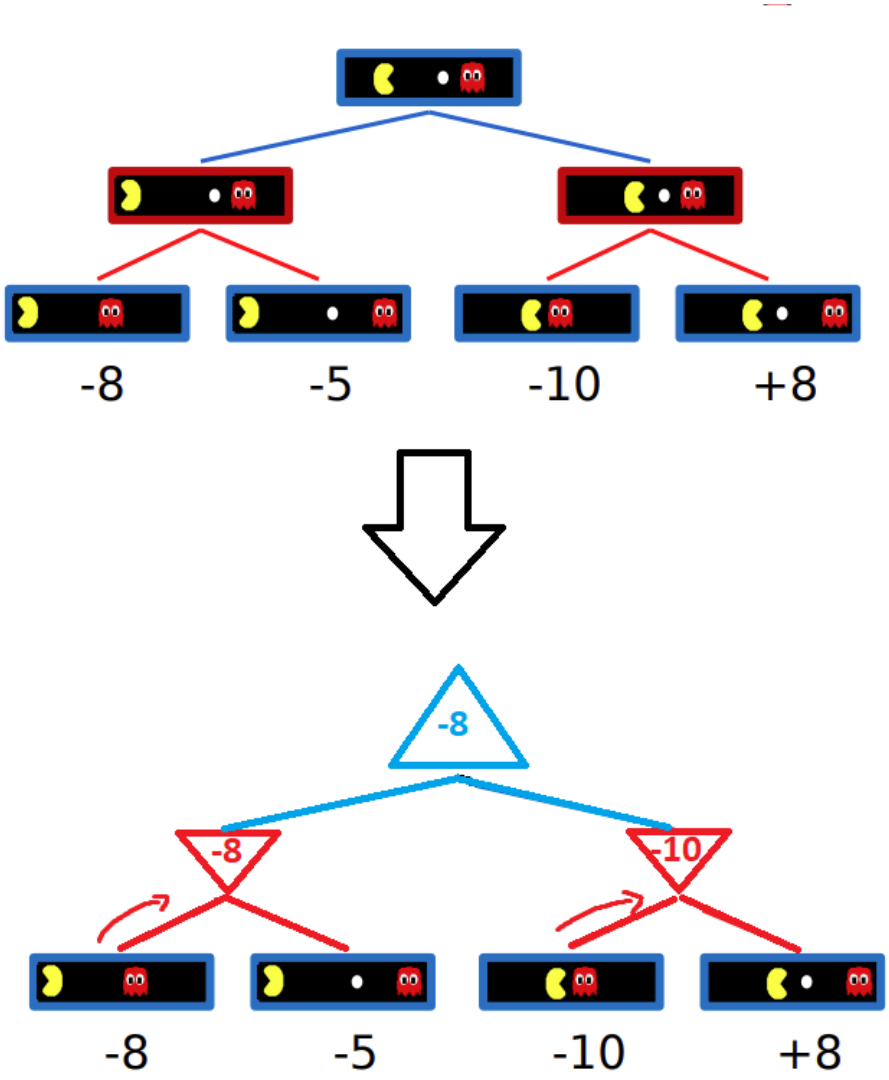

Figura 8: Ejemplo Min_Max

3. Pregunta3: Podado Alpha-Beta

3.1. Descripción

En esta pregunta se pide realizar la implementación de un algoritmo min-max con podado empleando el sistema aplfa-beta. El podado alfa-beta es

una optimización del algoritmo que reduce la cantidad de nodos del árbol de búsqueda que se deben explorar. Esto se logra mediante el seguimiento de dos valores: alfa y beta, que representan las mejores opciones para el jugador maximizador (jugador que intenta maximizar su puntuación) y el jugador minimizador (jugador que intenta minimizar la puntuación del oponente) respectivamente. La idea detrás del podado alfa-beta es que si en algún punto del árbol de búsqueda se encuentra una opción que es claramente peor que otra, no es necesario explorar más allá de esa opción, ya que el jugador no la elegirá.

Solamente necesita:

- Recibir como entrada el estado actual.
- Una función de evaluación para los estados.
- Una función que devuelva el estado resultado de realizar una acción para cada uno de los agentes (Pac-Man y fantasmas).

Es importante recalcar:

- Alfa y beta se inicializan con menos infinito e infinito respectivamente.
- Value se inicializa con menos infinito e infinito cuando se está maximizando y minimizando respectivamente.
- Se devuelve la acción óptima.

3.1.1. Tipo de Algoritmo

- Algoritmo de toma de decisiones en juegos de suma cero.

3.2. Algoritmo

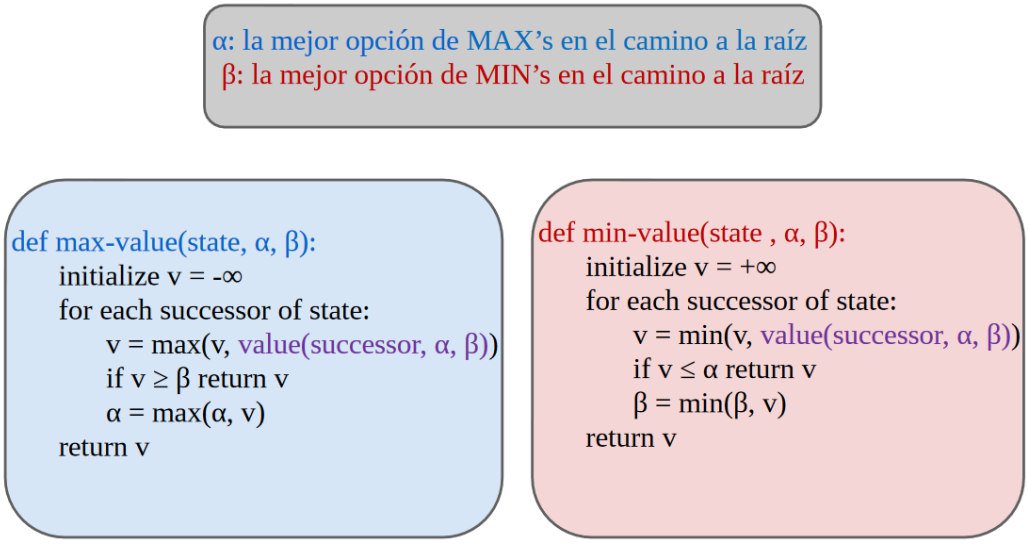

Figura 9: Algoritmo min-max con podado alpha-beta

3.2.1. Código V1

```
def getAction(self, game_state):
    """
    Returns the minimax action using self.depth and self.
    ↪ evaluationFunction
    """
    "*** YOUR CODE HERE ***"
    def min_value(game_state, agentIndex, depth, alpha, beta):
      # Comprobar si es un estado final
      if depth == 0 or game_state.isWin() or game_state.isLose():
        return self.evaluationFunction(game_state), None

      # Inicializacin
      value = float("inf")
      optAction = None

      # Algoritmo
      for action in game_state.getLegalActions(agentIndex):
        if agentIndex == game_state.getNumAgents() -1:
          # Se trata del Pac-Man -> cambiamos a maximizar
          v, a = max_value(game_state.generateSuccessor(agentIndex, action
    ↪ ), 0, depth, alpha, beta)
```

```
20
21          else:
22            # Se trata de un fantasma -> seguimoms minimizando
23            v, a = min_value(game_state.generateSuccessor(agentIndex, action
    ↪ ), agentIndex+1, depth, alpha, beta)
24          if v < value:
25            value, optAction = v, action
26
27          if value <= alpha:
28            return value, optAction
29
30          beta = min(beta, value)
31
32        return value, optAction
33
34
35      def max_value(game_state, agentIndex, depth, alpha, beta):
36        # Comprobar si es un estado final
37        if depth == 0 or game_state.isWin() or game_state.isLose():
38          return self.evaluationFunction(game_state), None
39
40        # Inicializacin
41        value = -float("inf")
42        optAction = None
43
44        # Algoritmo
45        for action in game_state.getLegalActions(agentIndex):
46          v, a = min_value(game_state.generateSuccessor(agentIndex, action),
    ↪  agentIndex+1, depth, alpha, beta)
47
48          if v > value:
49            value, optAction = v, action
50
51          if value >= beta:
52            return value, optAction
53
54          alpha = max(alpha, value)
55
56        return value, optAction
57
58
59        alpha = -float("inf")
60        beta = float("inf")
61
62        value, optAction = max_value(game_state, 0, self.depth, alpha, beta)
63
64        return value, optAction
65
66
```

```
    # LLamada principal
    alpha = -float("inf")
    beta = float("inf")

    value, optAction = max_value(game_state, 0, self.depth, alpha, beta)

    return optAction
```

Listing 3: Python example

```
Finished at 18:36:26

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 0/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 9/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 10: Primea versión AlphaBetaAgent

3.2.2. Código V-Final

```
def getAction(self, game_state):
    """
    Returns the minimax action using self.depth and self.
    ↪ evaluationFunction
    """
    "*** YOUR CODE HERE ***"
    def min_value(game_state, agentIndex, depth, alpha, beta):
      # Comprobar si es un estado final
      if depth == 0 or game_state.isWin() or game_state.isLose():
        return self.evaluationFunction(game_state), None

      # Inicializacin
      value = float("inf")
      optAction = None

      # Algoritmo
      for action in game_state.getLegalActions(agentIndex):
        if agentIndex == game_state.getNumAgents() -1:
          # Se trata del Pac-Man -> cambiamos a maximizar
          v, a = max_value(game_state.generateSuccessor(agentIndex, action
    ↪ ), 0, depth -1, alpha, beta)
```

```
20
21          else:
22            # Se trata de un fantasma -> seguimoms minimizando
23            v, a = min_value(game_state.generateSuccessor(agentIndex, action
    ↪ ), agentIndex+1, depth, alpha, beta)
24          if v < value:
25            value, optAction = v, action
26
27          if value < alpha:
28            return value, optAction
29
30          beta = min(beta, value)
31
32        return value, optAction
33
34
35      def max_value(game_state, agentIndex, depth, alpha, beta):
36        # Comprobar si es un estado final
37        if depth == 0 or game_state.isWin() or game_state.isLose():
38          return self.evaluationFunction(game_state), None
39
40        # Inicializacin
41        value = -float("inf")
42        optAction = None
43
44        # Algoritmo
45        for action in game_state.getLegalActions(agentIndex):
46          v, a = min_value(game_state.generateSuccessor(agentIndex, action),
    ↪  agentIndex+1, depth, alpha, beta)
47
48          if v > value:
49            value, optAction = v, action
50
51          if value > beta:
52            return value, optAction
53
54          alpha = max(alpha, v)
55
56        return value, optAction
57
58
59        alpha = -float("inf")
60        beta = float("inf")
61
62        value, optAction = max_value(game_state, 0, self.depth, alpha, beta)
63
64        return value, optAction
65
66
```

```
67      # LLamada principal
68      alpha = -float("inf")
69      beta = float("inf")
70
71      value, optAction = max_value(game_state, 0, self.depth, alpha, beta)
72
73      return optAction
```

Listing 4: Python example

```
Finished at 10:43:00

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 0/5
Question q5: 0/6
------------------
Total: 14/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 11: Versión final AlphaBetaAgent

3.2.3. Problemas y dificultades encarados en el ejercicio

A parte de la implementación del algoritmo min-max, nos salia 0/5 en el autograder porque se expandía un numero incorrecto de nodos. La solución a esto ha sido cambiar el menor igual y el mayor igual del alpha y beta por un menor que y un mayor que. A parte de estos casos, no han habido complicaciones excesivas.

3.2.4. Ejemplo

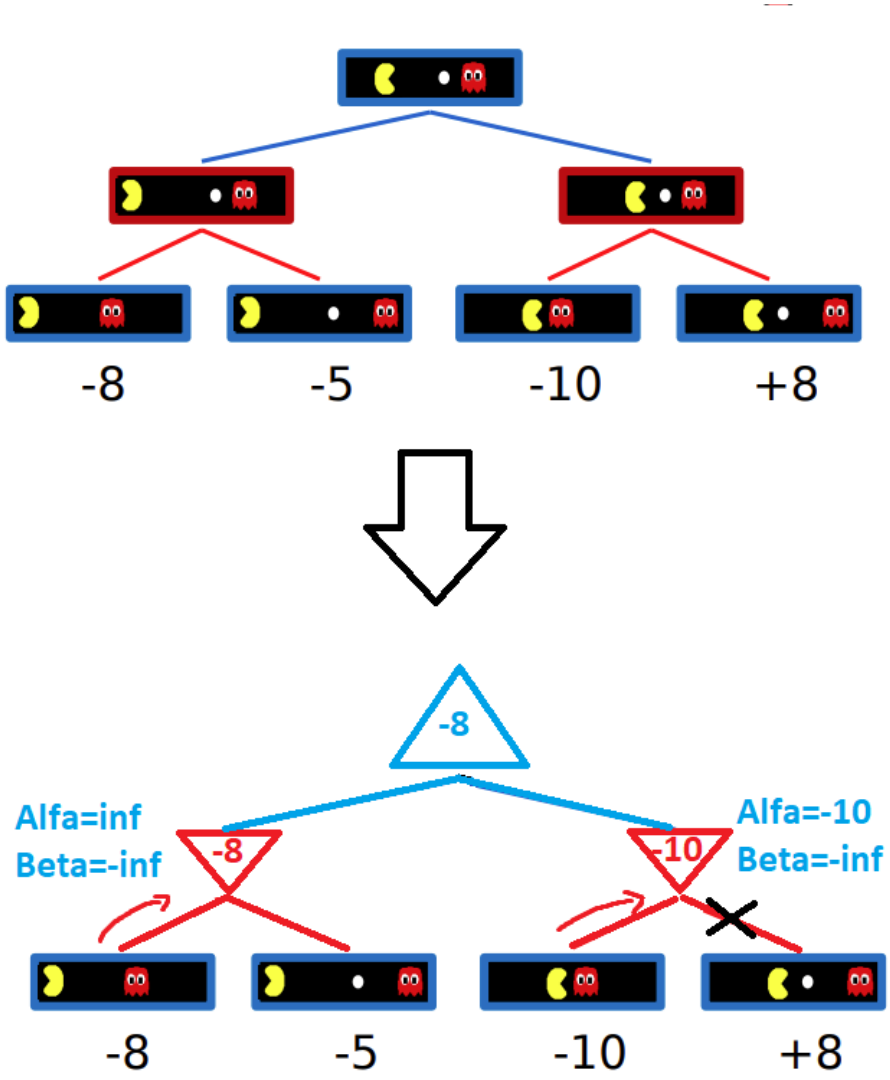

Figura 12: Ejemplo MinMax Poda

4. Pregunta4: Expectimax

4.1. Descripción

El algoritmo Minimax realiza una suposición muy fuerte: el contrincante siempre va a tomar la decisión óptima que maximiza su ganancia. Sin em-

bargo, en una situación real, asumir que el contrincante toma siempre las decisiones óptimas puede no ser la mejor estrategia. Es por ello que surge la necesidad de un algoritmo que incluya algo de incertidumbre en la toma de decisiones teniendo en cuenta que las acciones futuras son desconocidas. Este es el caso del algoritmo Expectimax, un algoritmo muy similar al Minimax, pero que en lugar de una fase de minimización tiene un proceso de calcular la expectativa de las recompensas resultantes.

Solamente necesita:

- Recibir como entrada el estado actual.
- Una función de evaluación para los estados.
- Una función que devuelva el estado resultado de realizar una acción para cada uno de los agentes (Pac-Man y fantasmas).

Es importante recalcar:

- Value se inicializa con menos infinito y 0 cuando se está maximizando y calculando la expectativa de las recompensas resultantes respectivamente.
- Se devuelve la acción óptima.

4.1.1. Tipo de Algoritmo

- Algoritmo de toma de decisiones en juegos donde el contrincante realiza acciones de forma estocástica.

4.2. Algoritmo

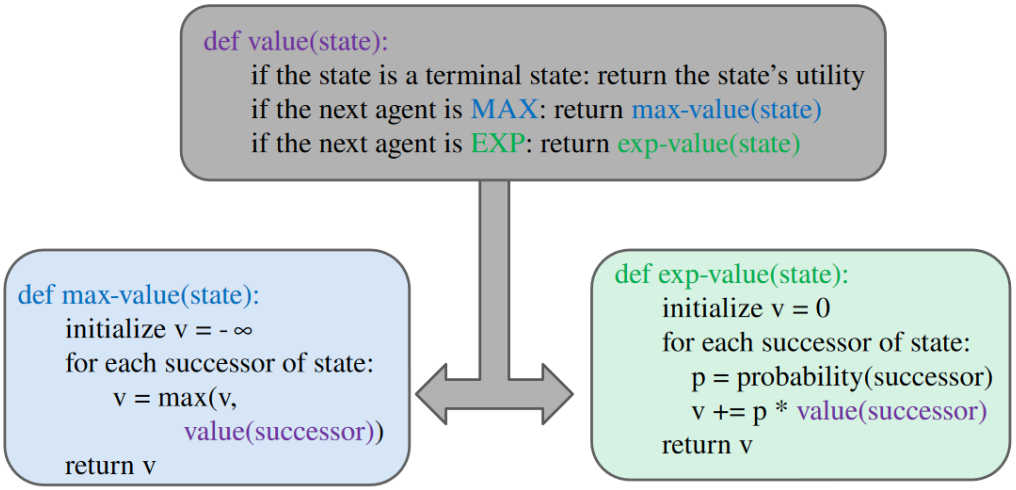

Figura 13: Algoritmo Expectimax

4.2.1. Código V-Final

```
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 exp_value(game_state, agentIndex, depth):
      # Comprobar si es un estado final
      if depth == 0 or game_state.isWin() or game_state.isLose():
        return self.evaluationFunction(game_state)
      value = 0
      actions = game_state.getLegalActions(agentIndex)

      for action in actions:
        if agentIndex == game_state.getNumAgents() -1:
          # Se trata del Pac-Man -> cambiamos a maximizar
          v, a = max_value(game_state.generateSuccessor(agentIndex, action
    ↪ ), 0, depth -1)
```

```
20
21          else:
22            # Se trata de un fantasma -> seguimoms minimizando
23            v = exp_value(game_state.generateSuccessor(agentIndex, action),
    ↪ agentIndex+1, depth)
24
25          p = 1 / len(actions) # La probabilidad es uniforme
26          value += p * v
27
28        return  value
29
30      def max_value(game_state, agentIndex, depth):
31        # Comprobar si es un estado final
32        if depth == 0 or game_state.isWin() or game_state.isLose():
33          return self.evaluationFunction(game_state), None
34
35        # Inicializacin
36        value = -float("inf")
37        optAction = None
38
39        # Algoritmo
40        for action in game_state.getLegalActions(agentIndex):
41          v = exp_value(game_state.generateSuccessor(agentIndex, action),
    ↪ agentIndex+1, depth)
42
43          if v > value:
44            value, optAction = v, action
45
46        return value, optAction
47
48      value, optAction = max_value(gameState, 0, self.depth)
49
50      return optAction
```

Listing 5: Python example

```
Finished at 17:35:00

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 5/5
Question q5: 0/6
------------------
Total: 19/25

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 14: Versión final Expectimax

4.2.2. Problemas y dificultades encarados en el ejercicio

No han habido problemas de implementación. Se ha tomado el código del minimax de la pregunta 3 y, modificando la etapa de minimización para realizar la expectación, se ha conseguido toda la puntuación en el autograder.

4.2.3. Ejemplo

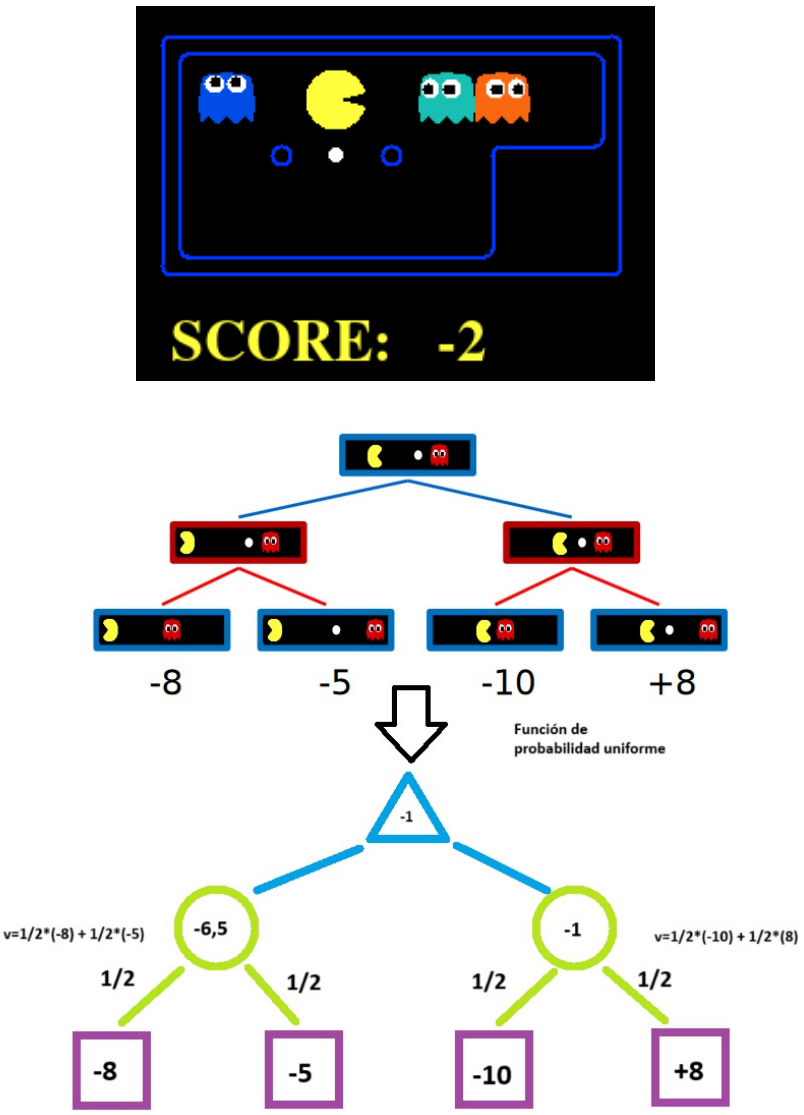

Figura 15: Ejemplo Expectimax

5. Pregunta5: Beter Evaluation Function

5.1. Descripción

El objetivo de este ejercicio es el de implementar una función que, dado un estado, proporcione una evaluación cuantitativa del mismo. La motivación de esta función es la de diferenciar dos estados distintos y decidir cual es mejor para obtener un objetivo. En nuestro caso, nuestro objetivo como Pac-Man es el de alcanzar todas las comidas evitando a los fantasmas sin olvidar que disponemos de cápsulas con las que podemos eliminarlos. Esta función de evaluación es crucial para los algoritmos de decisión que hemos implementado previamente, por lo que el buen desempeño del Pac-Man depende de una buena evaluación de los estados del juego.

Solamente necesita:

- Posición actual del Pac-Man.
- Posiciones actuales de las comidas.
- Posiciones de los fantasmas diferenciando entre asustados y no asustados.
- Posiciones de las capsulas.

Es importante recalcar:

- Si nos encontramos ante un estado ganador, devolveremos infinito.
- Si nos encontramos ante un estado perdedor, devolveremos menos infinito.
- Si no es ninguno de los dos casos anteriores, evaluaremos el estado.

5.2. Algoritmo

BETTER EVALUATION FUNCTION(*estado*)

Si es un estado final ganador

return ïnfinito”

Si es un estado final perdedor

return -ïnfinito”

min-dist-comida = calc-dist-min-comidas(estado)

min-dist-asust = calc-dist-min-fantAsust(estado)

min-dist-fantasma = calc-dist-min-fantNoAsust(estado)

puntuacion = obtener-puntuacion-estado(estado)

num-capsulas = obtener-num-capsulas(estado)

num-comidas = obtener-num-comidas(estado)

evaluacion = puntuacion

evaluacion += 1 / (min-dist-comida +1)

evaluacion += 1 / (min-dist-asust +1)

evaluacion -= num-capsulas

evaluacion -= num-comidas

return evaluacion

5.2.1. Código V1

```
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).

    DESCRIPTION: <write something here so we know what you did>
    """
    pacman_pos = currentGameState.getPacmanPosition()
```

```
    newFood = currentGameState.getFood().asList()
    newGhostStates = currentGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in
    ↪ newGhostStates]
    capsules = currentGameState.getCapsules()

    "*** YOUR CODE HERE ***"
    # Devolver casos finales
    if currentGameState.isWin():
      return float("inf")
    if currentGameState.isLose():
      return float("-inf")
    if len(newFood) == 0:
      return float("inf")

    # Calcular la distancia ms corta a una comida
    min_distancia_comida = min([manhattanDistance(pacman_pos, food) for
    ↪ food in newFood])

    # Calcular la distancia ms corta a un fantasma asustado
    fantasmas_austados  = [g for i, g in enumerate(newGhostStates) if
    ↪ newScaredTimes[i] > 0]
    min_dist_asust = float("inf")
    if len(fantasmas_austados) > 0:
      min_dist_asust = min([manhattanDistance(pacman_pos, g.getPosition())
    ↪  for g in fantasmas_austados])


    # Calcular la distancia ms corta a un fantasma no asustado
    fantasmas_no_aust   = [g for g in newGhostStates if g not in
    ↪ fantasmas_austados]
    min_dist_no_asust = float("inf")
    if len(fantasmas_no_aust) > 0:
      min_dist_asust = min([manhattanDistance(pacman_pos, g.getPosition())
    ↪  for g in fantasmas_no_aust])

    # Consideramos el nmero de capsulas
    num_capsulas = len(capsules)

    # Consideramos la puntuacin actual
    puntuacion = currentGameState.getScore()

    # Finalmente, valoramos el estado
    evaluation = puntuacion
    evaluation += 1 / (min_distancia_comida +1)   # +1 para evitar
    ↪ divisiones por 0
    evaluation += 1 / (min_dist_asust +1)         # +1 para evitar
    ↪ divisiones por 0
```

```
    evaluation -= 1 / (min_dist_no_asust +1)      # +1 para evitar
    ↪ divisiones por 0
    evaluation += num_capsulas

    return evaluation
```

Listing 6: Python example

```
Finished at 9:06:23

Provisional grades
==================
Question q1: 4/4
Question q2: 5/5
Question q3: 5/5
Question q4: 5/5
Question q5: 5/6
------------------
Total: 24/25

Your grades are NOT yet registered.  To register your gr
to follow your instructor's guidelines to receive credit

ag6154lk@msidealan:~/IngInfor/TIA/PacMan2$
```

Figura 16: Versión inicial Beter Evaluation Function

5.2.2. Código V-Final

```
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (question 5).

    DESCRIPTION: <write something here so we know what you did>
    """
    pacman_pos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood().asList()
    newGhostStates = currentGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in
    ↪ newGhostStates]
    capsules = currentGameState.getCapsules()

    "*** YOUR CODE HERE ***"
    # Devolver casos finales
    if currentGameState.isWin():
      return float("inf")
    if currentGameState.isLose():
```

```
19        return -float("inf")
20      if len(newFood) == 0:
21        return float("inf")
22
23      # Calcular la distancia ms corta a una comida
24      min_distancia_comida = min([manhattanDistance(pacman_pos, food) for
        ↪ food in newFood])
25
26      # Calcular la distancia ms corta a un fantasma asustado
27      fantasmas_austados  = [g for i, g in enumerate(newGhostStates) if
        ↪ newScaredTimes[i] > 0]
28      min_dist_asust = 0
29      if len(fantasmas_austados) > 0:
30        min_dist_asust = min([manhattanDistance(pacman_pos, g.getPosition())
        ↪  for g in fantasmas_austados])
31
32
33      # Calcular la distancia ms corta a un fantasma no asustado
34      fantasmas_no_aust   = [g for g in newGhostStates if g not in
        ↪ fantasmas_austados]
35      min_dist_no_asust = 0
36      if len(fantasmas_no_aust) > 0:
37        min_dist_asust = min([manhattanDistance(pacman_pos, g.getPosition())
        ↪  for g in fantasmas_no_aust])
38
39      # Consideramos el nmero de capsulas
40      num_capsulas = len(capsules)
41
42      # Consideramos la puntuacin actual
43      puntuacion = currentGameState.getScore()
44
45      # Finalmente, valoramos el estado
46      evaluation = puntuacion                          # A mayor puntuacin
        ↪ -> Mejor
47      evaluation += 5 / (min_distancia_comida +0.1)   # A menor distancia
        ↪ -> Mejor    | +0.1 para evitar divisiones por 0
48      evaluation += 3 / (min_dist_asust +1)           # A menor distancia
        ↪ -> Mejor    | +1 para evitar divisiones por 0
49      evaluation -= 10 * min_dist_no_asust            # A mayor distancia
        ↪ -> Mejor
50      evaluation -= 2 * len(newFood)                  # A mayor numero
        ↪ -> Peor
51      evaluation -= 4 * num_capsulas                  # A mayor numero
        ↪ -> Peor (queremos que las coma)
52
53      return evaluation
```

Listing 7: Python example

Figura 17: Versión final Beter Evaluation Function

5.2.3. Problemas y dificultades encarados en el ejercicio

La mayor dificultad de este ejercicio ha sido conseguir los 6 puntos del autograder. En un primer momento, la evaluación del estado la realizamos teniendo en cuenta la puntuación del estado que mide el tiempo (cuanto más rápido se termine, mejor); la distancia inversa a la comida más cercana (a menor distancia mejor); la distancia inversa al fantasma asustado más cercano (a menor distancia mejor); la distancia inversa al fantasma más cercano de forma que cuanto más cerca esté menor puntuación tenga ese estado (resta) y el número de cápsulas (a mayor número de cápsulas, mejor). Sin embargo, con este sistema solo conseguíamos 5 de los 6 puntos del ejercicio, por lo que todavía habían mejoras por implementar.

Después de probar varios factores constantes de ponderación a cada una de las variables de evaluación sin obtener resultados significativos, decidimos incluir también el número de comidas como variable de evaluación. De esta manera, a mayor número de comidas, peor es la evaluación del estado (resta). Además, nos dimos cuenta de que el número de cápsulas debería restar en lugar de sumar valor (queremos que el Pac-Man intente comerlas para eliminar a los fantasmas). También, la distancia al fantasma más cercano debería de restar directamente en lugar de hacer la resta de la distancia inversa ya que nos interesa que el Pac-Man se aleje lo máximo posible de los fantasmas (tuvimos un error representando este hecho).

Con todo ello, finalmente conseguimos los 6 puntos del autograder. A pesar de ello, consideramos que puede existir algún punto de mejora ya que, en algunas ocasiones, el Pac-Man se queda .[es]perandoçuando el fantasma no está asustado y solo queda una comida. Este caso se muestra en la figura 18 en el cual el Pac-Man está al lado de la última comida hasta que el fantasma se le acerca y termina comiendo la comida. Esto se debe a que en este caso

se está valorando con la misma puntuación el hecho de comerse la comida o no, lo cual no llegamos a entender ya que hemos implementado que en caso de victoria se devuelva un infinito.

5.2.4. Ejemplo

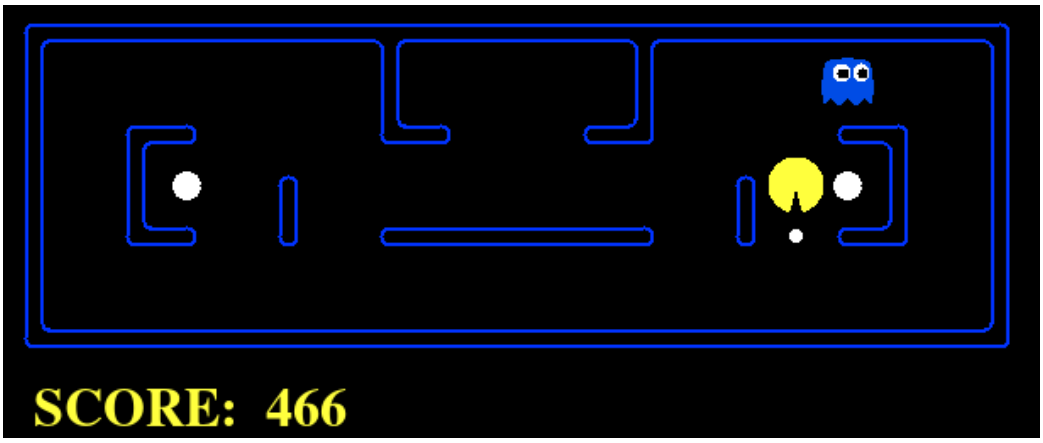

Figura 18: Ejemplo Better Evaluation Function