Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

TÉCNICAS DE INTELIGENCIA ARTIFICIAL
4º Curso, Grupo 46
1er cuatrimestre

Practica 2 - Búsqueda Multi-Agente

Nagore Gómez y Sergio Martín.

Bilbao, 27 de octubre de 2023

Índice

1. Pregunta 1: Agente Reflex

1.1. Descripción

Las funciones de evaluación son muy útiles para determinar cuan beneficioso es un estado respecto a los intereses de un agente. De modo que cuando el árbol de estados se torna demasiado extenso y se vuelve complicado computar todos los estados hasta llegar a los objetivos, puede ser conveniente desarrollar una función de evaluación.

La función evaluadora se puede aplicar sobre cualquier estado del espacio de estados contemplado, y para cada uno de ellos devolverá un valor. Se puede apreciar una funcionalidad similar a la que tienen los heurísticos en el laboratorio anterior. Por lo tanto, una mayor puntuación de la función de evaluación deberá significar que ese estado es mejor que aquellos estados que reciban una valoración inferior a la suya. Consecuentemente, el agente deberá escoger desplazarse al estado con mayor puntuación.

Finalmente, la definición de la función evaluadora depende de que agente vaya aplicarla. Precisamente ante un problema de búsqueda adversarial los agentes tienen intereses opuestos, por lo que las funciones de evaluación deberían de ser inversas también. Es evidente que un estado de alta puntuación para el agente pacman, supone un problema para un agente fantasma.

Solamente necesita:

- El objetivo del agente, para poder determinar que estados le convienen y cuales no.
- El estado actual del agente y la acción, para obtener el nodo sucesor y evaluarlo.

Es importante recalcar:

- Se presupone que el agente es de tipo reflex, por lo que tiene una tendencia cortoplacista.

1.2. Pseudocódigo de la función de evaluación

```
funcionEvaluadora():

    dist_f = Calcular distancia al fantasma mas cercano
    dist_c = Calcular distancia a la comida mas cercana

    SI dist_f == dist_c:
        RETURN -1 / dist_f

    SI dist_f > dist_c:
        RETURN 1 / dist_c
```

```
    SI NO
        RETURN -1 / dist_c
```

1.2.1. Código V1

```
def evaluationFunction(self, currentGameState, action):

    successorGameState = currentGameState.
        generatePacmanSuccessor(action)

    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood().asList()

    oldPos = currentGameState.getPacmanPosition()
    oldFood = currentGameState.getFood().asList()

    newGhostStates = successorGameState.getGhostStates()
    oldGhostStates = currentGameState.getGhostStates()


    # CASOS EXTREMOS

    if successorGameState.isLose():
        return -100000
    elif successorGameState.isWin():
        return 100000

    # CALCULAR DISTANCIAS A LA COMIDA (ANTES VS DESPUES)

    newFoodDist = [util.manhattanDistance(newPos, food) for food
        in newFood]
    oldFoodDist = [util.manhattanDistance(oldPos, food) for food
        in oldFood]

    minNewFoodDist = min(newFoodDist)
    minOldFoodDist = min(oldFoodDist)

    # CALCULAR DISTANCIAS A LOS FANTASMAS (ANTES VS DESPUES)

    newGhostDistances = [util.manhattanDistance(newPos,
        ghostState.getPosition()) for ghostState in
        newGhostStates]
    oldGhostDistances = [util.manhattanDistance(oldPos,
        ghostState.getPosition()) for ghostState in
        oldGhostStates]
```

```
    minNewGhostDist = min(newGhostDistances)
    minOldGhostDist = min(oldGhostDistances)

    # FACTOR 1 -> CERCANIA A LA COMIDA
    v1 = 0
    if newFoodDist and minNewFoodDist < minOldFoodDist:
        v1 = 40

    # FACTOR 2 -> NUMERO DE COMIDAS EN EL TABLERO
    v2 = 0
    if len(newFoodDist) < len(oldFoodDist):
        v2 = 170

    # FACTOR 3 -> CERCANIA A LOS FANTASMAS
    v3 = 0
    if minNewGhostDist > minOldGhostDist:
        v3 = 10
    else:
        v3 = -5

    value = v1 + v2 + v3

    return successorGameState.getScore() - currentGameState.
        getScore() + value
```

```
Question q1
===========

Pacman emerges victorious! Score: 1258
Pacman emerges victorious! Score: 1258
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1253
Pacman emerges victorious! Score: 1260
Pacman emerges victorious! Score: 1251
Pacman emerges victorious! Score: 1261
Pacman emerges victorious! Score: 1243
Pacman emerges victorious! Score: 1258
Pacman emerges victorious! Score: 1252
Average Score: 1253.3
Scores:        1258.0, 1258.0, 1239.0, 1253.0, 1260.0, 1251.0,
    1261.0, 1243.0, 1258.0, 1252.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q1\grade-agent.test (4 of 4 points)
***      1253.3 average score (2 of 2 points)
***          Grading scheme:
***           < 500:  0 points
```

```
***          >= 500:  1 points
***          >= 1000:  2 points
***      10 games not timed out (0 of 0 points)
***          Grading scheme:
***           < 10:  fail
***          >= 10:  0 points
***      10 wins (2 of 2 points)
***          Grading scheme:
***           < 1:  fail
***          >= 1:  0 points
***          >= 5:  1 points
***          >= 10:  2 points

### Question q1: 4/4 ###
```

1.2.2. Código V-Final

```
def evaluationFunction(self, currentGameState, action):

    # OBTENER LA INFORMACION NECESARIA

    successorGameState = currentGameState.
        generatePacmanSuccessor(action)
    newPos = successorGameState.getPacmanPosition()
    newFood = successorGameState.getFood()
    newGhostStates = successorGameState.getGhostStates()

    # ENCONTRAR LA COMIDA O CAPSULA MAS CERCANA

    food_min_distance=0
    foods = newFood.asList() + currentGameState.getCapsules()
    for food in foods:
        food_dist=util.manhattanDistance(food,newPos)
        if(food_min_distance>=food_dist or food_min_distance==0)
            :
            food_min_distance=food_dist

    # ENCONTRAR EL FANTASMA MAS CERCANO

    ghost_min_distance=0
    for ghostState in newGhostStates:
        ghost_dist=util.manhattanDistance(ghostState.getPosition
            (),newPos)
        if(ghost_min_distance>=ghost_dist or ghost_min_distance
            ==0):
            ghost_min_distance=ghost_dist
```

```
    # EVALUAR LA SITUACION

    if food_min_distance == 0:
        return sys.maxsize + successorGameState.getScore()
    elif ghost_min_distance == 0:
        return -sys.maxsize + successorGameState.getScore()
    elif food_min_distance == ghost_min_distance:
        return -1 /ghost_min_distance + successorGameState.
            getScore()
    elif food_min_distance < ghost_min_distance:
        return 1/food_min_distance + successorGameState.getScore
            ()
    else:
        return -food_min_distance + successorGameState.getScore
            ()
```

```
Question q1
===========

Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1423
Pacman emerges victorious! Score: 1428
Pacman emerges victorious! Score: 1240
Pacman emerges victorious! Score: 1211
Pacman emerges victorious! Score: 1243
Pacman emerges victorious! Score: 1094
Pacman emerges victorious! Score: 1229
Pacman emerges victorious! Score: 1255
Pacman emerges victorious! Score: 1251
Average Score: 1260.5
Scores:        1231.0, 1423.0, 1428.0, 1240.0, 1211.0, 1243.0,
    1094.0, 1229.0, 1255.0, 1251.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q1\grade-agent.test (4 of 4 points)
***     1260.5 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (0 of 0 points)
***         Grading scheme:
***          < 10:  fail
***         >= 10:  0 points
***     10 wins (2 of 2 points)
***         Grading scheme:
```

```
***          < 1:  fail
***         >= 1:  0 points
***         >= 5:  1 points
***         >= 10:  2 points

### Question q1: 4/4 ###
```

1.2.3. Problemas y dificultades encarados en el ejercicio

Tratándose de una estimación, el mayor problema en este ejercicio ha resultado en encontrar una combinación de valores que fuese consistente para esta situación. De modo que el orden de preferencia entre diferentes estados se adaptase a la realidad. Por otro lado, es la primera tarea que se realiza sobre este código, luego ha sido necesaria la adaptación al mismo. La adaptación no ha resultado un problema pero si que requiere de algún tiempo para dominarla completamente.

La versión de *Código-V1* calcula la diferencia entre el estado actual y sus posibles sucesores, recompensando las transiciones que suponen una mejora y penalizando aquellas que suponen empeoramiento. A pesar de ser una método que funciona tiene un coste computacional que puede ser reducido si se evalúan los estados de forma independiente, sin la necesidad de calcular las diferencias entre si. Principalmente porque el estado "padre" sera el mismo en todos los casos.

1.2.4. Ejemplo (Figura 1)

Figura 1: Ejemplo de uso de la función evaluadora

2. Pregunta 2 Algoritmo MíniMax

2.1. Descripción

El algoritmo MíniMax es un algoritmo muy común en juegos de búsqueda adversarial, que se utiliza principalmente para encontrar la mejor acción a tomar en un juego adversarial de dos o mas jugadores. Este algoritmo expande el árbol de estados, totalmente o parcialmente, para a continuación calcular el valor de cada nodo del árbol expandido.

Para asignar los valores a los nodos se emplea una función de utilidad, como la que se ha desarrollado en el apartado anterior. En el caso particular del juego *Pac-Man* el agente pacman tratara siempre de maximizar su puntuación mientras que los agentes fantasma trataran de minimizar el valor devuelto por la función de utilidad de pacman.

La dimensión del árbol de estados en este caso es demasiado extensa como para desarrollarlo por completo en busca de estados terminales, por lo que se utiliza el factor *depth*. El valor de profundidad indica cuantas capas del árbol se deben desarrollar, y una vez se alcanza esa profundidad se ha de tomar una decisión con los valores de utilidad de la capa actual, sean terminales o no.

Finalmente, en el árbol de estados habrá capas de maximización (los turnos del agente pacman) donde se busque un valor de utilidad máximo entre sus sucesores y habrá capas de minimización (los turnos de los agentes fantasma) donde se buscara un valor de utilidad mínimo entre sus sucesores.

Solamente necesita:

- Un árbol que represente los estados posibles y la unión entre ellos.
- Una función de utilidad para evaluar la puntuación de los nodos.
- Una profundidad de búsqueda. Valor *depth*.

Es importante recalcar:

- Si se expande el árbol al completo encontrará la mejor solución para el agente pacman.
- Puede llegar a ser muy costoso expandir un árbol de estados completo.

2.1.1. Tipo de Algoritmo

- Algoritmo de búsqueda **adversarial**

2.2. Algoritmo

```
def getAction(self, game_state):
    initialize best_action = None
    initialize best_value = -inf
    FOR acciones posibles:
        v = value(sucesor)
        if v > best_value:
            best_value = v
            best_action = action
    return best_action

def value(state):
    SI profundidad = 0 o ganar o perder:
        return funcion de evaluacion
    SI agente = fantasma:
        return min_value(state)
    SINO:
        return max_value(state)

def max_value(state):
    initialize v = -inf
    FOR each sucessor of state:
        v = max (v,value(sucessor))
    return v

def min_value(state):
    initialize v = +inf
    FOR each sucessor of state:
        v = min (v,value(sucessor))
    return v
```

2.2.1. Código V1

```
def getAction(self, game_state):

     def value(state, agentIndex):
        if state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        elif agentIndex >= 1:
            return min_value(state, agentIndex)

        else:
            return max_value(state)
```

```
    def max_value(state):
        v = -10000
        legalActions = state.getLegalActions(0)
        for action in legalActions:
            successor = state.generateSuccessor(0, action)
            v = max(v, value(successor, 1))
        return v

    def min_value(state, agentIndex):
        v = 10000
        legalActions = state.getLegalActions(agentIndex)
        for action in legalActions:
            successor = state.generateSuccessor(agentIndex,
                action)
            v = min(v, value(successor, (agentIndex + 1) % 2))
        return v

    best_action = None
    best_value = float("-inf")
    legalActions = game_state.getLegalActions(0)
    for action in legalActions:
        successor = game_state.generateSuccessor(0, action)
        v = value(successor, 1)
        if v > best_value:
            best_value = v
            best_action = action

    return best_action
```

```
Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** FAIL: test_cases\q2\1-7-minmax.test
***     Incorrect move for depth=4
***         Student move: Right
```

```
***          Optimal move: Left
***      Tree:
***                    /------a------\
***                   /               \
***                  /                 \
***               b1                    b2
***               |                   /    \
***               cx              c3         c4
***               |             /    \      /    \
***               dx           d5    d6    d7    d8
***               |           / \   / \   / \   / \
***               Z           I  J  K  L  M  N  O  P
***            -1.99        -1 -9  4  7  2  5 -3 -2
***
***      a - max
***      b - min
***      c - min
***      d - max
***
***      Note that the minimax value of b2 is -2
*** PASS: test_cases\q2\1-8-minmax.test
*** FAIL: test_cases\q2\2-1a-vary-depth.test
***      Incorrect generated nodes for depth=1
***          Student generated nodes: a b1 b2 c1 c2 cx d1 d2 d3
    d4 dx
***          Correct generated nodes: a b1 b2 c1 c2 cx
***      Tree:
***                    /------a------\
***                   /               \
***                  /                 \
***               b1                    b2
***             /    \                   |
***      -4 c1          c2  9           cx  -4.01
***        /   \        /   \            |
***       d1    d2    d3    d4          dx
***      -3     -9    10     6        -4.01
***
***      a - max
***      b - min
***      c - max
***
***      Note that the minimax value of b1 is -3, but the depth=1
     limited value is -4.
***      The values next to c1, c2, and cx are the values of the
    evaluation function, not
***      necessarily the correct minimax backup.
*** PASS: test_cases\q2\2-1b-vary-depth.test
```

```
*** FAIL: test_cases\q2\2-2a-vary-depth.test
***     Incorrect move for depth=1
***         Student move: Left
***         Optimal move: Right
***     Incorrect generated nodes for depth=1
***         Student generated nodes: a b1 b2 c1 c2 cx d1 d2 d3
   d4 dx
***         Correct generated nodes: a b1 b2 c1 c2 cx
***     Tree:
***                    /------a------\
***                   /              \
***                  /                \
***                b1                  b2
***               /  \                 |
***     -4 c1          c2 9            cx -3.99
***       /   \        /   \           |
***      d1    d2   d3    d4          dx
***     -3     -9   10     6         -3.99
***
***     a - max
***     b - min
***     c - max
***
***     Note that the minimax value of b1 is -3, but the depth=1
    limited value is -4.
***     The values next to c1, c2, and cx are the values of the
   evaluation function, not
***     necessarily the correct minimax backup.
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** FAIL: test_cases\q2\2-3a-vary-depth.test
***     Incorrect generated nodes for depth=1
***         Student generated nodes: a b1 b2 c3 c4 cx d5 d6 d7
   d8 dx
***         Correct generated nodes: a b1 b2 c3 c4 cx
***     Tree:
***                    /------a------\
***                   /              \
***                  /                \
***                b1                  b2
***                 |                /    \
***      5.01      cx            8   c3     c4    5
***                 |              /  \     /  \
***                dx            d5   d6   d7   d8
***               5.01           4    -7   0    5
***
***     a - max
***     b - min
```

```
***      c - max
***
***      Note that the minimax value of b1 is 4, but the depth=1
    limited value is 5.
***      The values next to c3, c4, and cx are the values of the
    evaluation function, not
***      necessarily the correct minimax backup.
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** FAIL: test_cases\q2\2-4a-vary-depth.test
***      Incorrect move for depth=1
***          Student move: Left
***          Optimal move: Right
***      Incorrect generated nodes for depth=1
***          Student generated nodes: a b1 b2 c3 c4 cx d5 d6 d7
    d8 dx
***          Correct generated nodes: a b1 b2 c3 c4 cx
***      Tree:
***                      /------a------\
***                     /              \
***                    /                \
***                  b1                  b2
***                   |                /    \
***        4.99      cx             8  c3      c4    5
***                   |              /   \     /   \
***                  dx             d5   d6   d7   d8
***                 4.99            4    -7   0    5
***
***      a - max
***      b - min
***      c - max
***
***      Note that the minimax value of b1 is 4, but the depth=1
    limited value is 5.
***      The values next to c3, c4, and cx are the values of the
    evaluation function, not
***      necessarily the correct minimax backup.
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
*** FAIL: test_cases\q2\7-1a-check-depth-one-ghost.test
***      Incorrect move for depth=1
***          Student move: Right
***          Optimal move: Left
***      Incorrect generated nodes for depth=1
```

```
***         Student generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
   d3 e1 e2 e3 f1 f2 f3 g1 g2 g3
***         Correct generated nodes: a b1 b2 b3 c1 c2 c3
***     Tree:
***                    a
***                 /-/   |   \--\
***                /      |      \
***          0 b1      0 b2      b3 8
***             |         |       |
***         10 c1      0 c2      c3 8
***             |         |       |
***          0 d1      0 d2      d3 8
***             |         |       |
***          0 e1     10 e2      e3 8
***             |         |       |
***          0 f1      0 f2      f3 8
***             |         |       |
***             g1        g2      g3
***             0         0       8
***
***     a - max
***     b - min
***     c - max
***     d - min
***     e - max
***     f - min
***
***     At depth 1, the evaluation function is called at level c
   ,
***     so Left should be returned. If your algorithm is
   returning a
***     different action, check how you implemented your depth.
*** FAIL: test_cases\q2\7-1b-check-depth-one-ghost.test
***     Incorrect move for depth=2
***         Student move: Right
***         Optimal move: Center
***     Incorrect generated nodes for depth=2
***         Student generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
   d3 e1 e2 e3 f1 f2 f3 g1 g2 g3
***         Correct generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
   d3 e1 e2 e3
***     Tree:
***                    a
***                 /-/   |   \--\
***                /      |      \
***          0 b1      0 b2      b3 8
***             |         |       |
```

```
***       10 c1     0 c2       c3 8
***          |        |         |
***        0 d1     0 d2       d3 8
***          |        |         |
***        0 e1    10 e2       e3 8
***          |        |         |
***        0 f1     0 f2       f3 8
***          |        |         |
***          g1       g2       g3
***          0        0         8
***
***      a - max
***      b - min
***      c - max
***      d - min
***      e - max
***      f - min
***
***      At depth 2, the evaluation function is called at level e
    ,
***      so Center should be returned. If your algorithm is
    returning a
***      different action, check how you implemented your depth.
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** FAIL: test_cases\q2\7-2a-check-depth-two-ghosts.test
***      Incorrect move for depth=1
***          Student move: Right
***          Optimal move: Left
***      Incorrect generated nodes for depth=1
***          Student generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
    d3 e1 e2 e3 f1 f2 f3 g1 g2 g3 h1 h2 h3 i1 i2 i3 j1 j2 j3
***          Correct generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
    d3
***      Tree:
***                    a
***              /-/   |   \--\
***             /      |      \
***        0 b1     0 b2       b3 8
***          |        |         |
***        0 c1     0 c2       c3 8
***          |        |         |
***       10 d1     0 d2       d3 8
***          |        |         |
***        0 e1     0 e2       e3 8
***          |        |         |
***        0 f1     0 f2       f3 8
***          |        |         |
```

```
***        0 g1    10 g2       g3 8
***          |        |         |
***        0 h1     0 h2       h3 8
***          |        |         |
***        0 i1     0 i2       i3 8
***          |        |         |
***          j1       j2       j3
***          0        0         8
***
***      a - max
***      b - min
***      c - min
***      d - max
***      e - min
***      f - min
***      g - max
***      h - min
***      i - min
***
***      At depth 1, the evaluation function is called at level d
   ,
***      so Left should be returned. If your algorithm is
   returning a
***      different action, check how you implemented your depth.
*** FAIL: test_cases\q2\7-2b-check-depth-two-ghosts.test
***      Incorrect move for depth=2
***          Student move: Right
***          Optimal move: Center
***      Incorrect generated nodes for depth=2
***          Student generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
   d3 e1 e2 e3 f1 f2 f3 g1 g2 g3 h1 h2 h3 i1 i2 i3 j1 j2 j3
***          Correct generated nodes: a b1 b2 b3 c1 c2 c3 d1 d2
   d3 e1 e2 e3 f1 f2 f3 g1 g2 g3
***      Tree:
***                    a
***              /-/   |   \--\
***             /      |      \
***        0 b1     0 b2       b3 8
***          |        |         |
***        0 c1     0 c2       c3 8
***          |        |         |
***       10 d1     0 d2       d3 8
***          |        |         |
***        0 e1     0 e2       e3 8
***          |        |         |
***        0 f1     0 f2       f3 8
***          |        |         |
```

```
***         0 g1    10 g2       g3 8
***           |       |          |
***         0 h1     0 h2       h3 8
***           |       |          |
***         0 i1     0 i2       i3 8
***           |       |          |
***           j1      j2         j3
***           0       0          8
***
***      a - max
***      b - min
***      c - min
***      d - max
***      e - min
***      f - min
***      g - max
***      h - min
***      i - min
***
***      At depth 2, the evaluation function is called at level g
    ,
***      so Center should be returned. If your algorithm is
    returning
***      a different action, check how you implemented your depth
    .
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
RecursionError: maximum recursion depth exceeded in comparison
Pacman crashed
Average Score: 0.0
Scores:        0.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after
    0.37174415588378906 seconds.
*** Won 0 out of 1 games. Average score: 0.0 ***
*** FAIL: test_cases\q2\8-pacman-game.test
***      Agent crashed on smallClassic.  No credit
*** Tests failed.

### Question q2: 0/5 ###
```

2.2.2. Código V-Final

```
def getAction(self, game_state):

```

```
def value(state, agentIndex, depth):
    # ESTADOS TERMINALES (GANAR/PERDER O PROFUNDIDAD
        ALCANZADA)
    if depth == 0 or state.isWin() or state.isLose():
        return self.evaluationFunction(state)

    # AGENTES FANTASMA
    elif agentIndex >= 1:
        return min_value(state, agentIndex, depth)

    # AGENTE PACMAN
    else:
        return max_value(state, depth)

def max_value(state, depth):  # SIEMPRE SE EJECUTARA PARA
    PACMAN

    v = float('-inf')
    legalActions = state.getLegalActions(0)  # ACCIONES
        POSIBLES PARA PACMAN

    for action in legalActions:
        successor = state.generateSuccessor(0, action)
        v = max(v, value(successor, 1, depth))  # ELEGIR EL
            MAXIMO DE LOS SUCESORES

    return v

def min_value(state, agentIndex, depth):  # SE EJECUTA EN
    LOS DIFERENTES FANTASMAS

    v = float('inf')
    legalActions = state.getLegalActions(agentIndex)

    for action in legalActions:
        successor = state.generateSuccessor(agentIndex,
            action)

        # SI ES EL ULTIMO FANTASMA, DESPUES PACMAN
        if agentIndex == state.getNumAgents() - 1:
            v = min(v, value(successor, 0, depth - 1))

        # SI NO LE TOCARA A OTRO FANTASMA
        else:
            v = min(v, value(successor, agentIndex + 1,
                depth))
```

```
        return v

    legal_actions = game_state.getLegalActions(0)  # OBTENER
       ACCIONES POSIBLES PARA PACMAN
    best_action = None
    best_value = float('-inf')

    # CALCULAR UN VALOR VALUE PARA CADA ACCION POSIBLE
    for action in legal_actions:
        sucessor = game_state.generateSuccessor(0, action)
        v = value(sucessor, 1, self.depth)
        if v > best_value:
            best_value = v
            best_action = action

    return best_action
```

```
Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
```

```
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after
    0.9217450618743896 seconds.
*** Won 0 out of 1 games. Average score: 84.0 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###
```

2.2.3. Problemas y dificultades encarados en el ejercicio

En este caso ha habido varios factores que han causado dificultades en el ejercicio. Uno de ellos ha sido el uso de la recursividad del algoritmo MíniMax, especialmente cuando por cada capa de maximizacion hay que contemplar varias capas de minimización.

Además, el uso del valor *depth* se ha pasado por alto en *Código-V1*, que sumada otros errores en la implementación causaban que el algoritmo no funcionase correctamente.

Tras la incorporación del valor *depth* y la correcta gestión de los turnos el algoritmo cumple con su cometido.

2.2.4. Ejemplo (Figura 2)

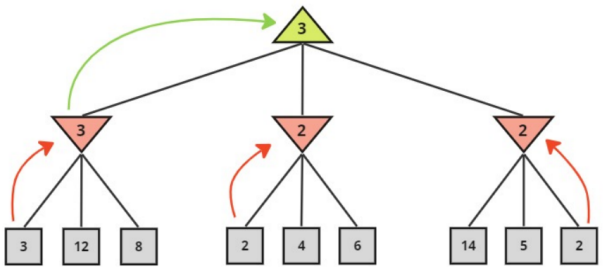

Figura 2: Ejemplo Mínimax

3. Pregunta 3: Podado *Alpha-Beta*

3.1. Descripción

El algoritmo Mínimax con poda alfa-beta es una extensión del algoritmo Mínimax corriente implementado en el apartado anterior. En esencia, la funcionalidad del algoritmo es la misma, solo que la implementación de la poda *Alpha-Beta* se consigue una mayor eficiencia en su aplicación.

La poda *Alpha-Beta* es una optimización clave que permite reducir el número de nodos evaluados en el árbol de búsqueda. Para ello, se basa en dos valores, *Alpha* y *Beta*, que se utilizan para comparar los valores de utilidad de los nodos. *Alpha* representa el valor mínimo que el agente minimizador está dispuesto a aceptar, mientras que *Beta* representa el valor máximo que el agente maximizador está dispuesto a pagar.

De modo que, en cada nodo del árbol, se exploran primero los estados sucesores que pueden mejorar el valor de *Alpha* o *Beta*. Si un estado sucesor no puede mejorar el valor de *Alpha* o *Beta*, se descartará la rama entera.

En resumen, el algoritmo Minimax con poda *Alpha-Beta* es una técnica efectiva para tomar decisiones en juegos competitivos, al reducir significativamente el espacio de búsqueda sin sacrificar la calidad de la solución. Los valores *Alpha* y *Beta* no hacen otra cosa que declarar una ventana de validez para que los valores de utilidad sean considerados.

Solamente necesita:

- Una función de utilidad para evaluar la puntuación de los nodos.
- Un árbol que represente los estados posibles y la unión entre ellos.
- Una profundidad de búsqueda. Valor *depth*.

Es importante recalcar:

- La poda *Alpha-Beta* puede eliminar ramas del árbol que no afectan la decisión final, lo que ahorra tiempo de computo.
- En juegos donde el factor de ramificación es elevado una técnica de poda puede marcar la diferencia entre que el problema sea computable o no.

3.1.1. Tipo de Algoritmo

- Algoritmo de búsqueda **adversarial** con poda

3.2. Algoritmo

```
def getAction(self, game_state):
    initialize best_action = None
    initialize best_value = -inf
    initialize alpha = -inf
    initialize beta = +inf
    FOR acciones posibles:
        v = value(sucesor)
        if v > best_value:
            actualizar = v
            actualizar = action
        alpha= max(alpha, best_value)
    return best_action

    def value(state,alpha,beta):
        SI profundidad = 0 o ganar o perder:
            return funcion de evaluacion
        SI el agente es un fantasma:
            return min_value(state,alpha,beta)
        SINO:
            return max_value(state,alpha,beta)

    def max_value(state,alpha,beta):
        initialize v = -inf
        FOR each sucessor of state:
            v = max (v,value(sucessor,alpha,beta))
            IF v > beta:
                return v
            alpha = max(alpha, v)
        return v

    def max_value(state,alpha,beta):
        initialize v = +inf
        FOR each sucessor of state:
            v = min (v,value(sucessor,alpha,beta))
            IF v < alpha:
                return v
            beta = min(beta, v)
        return v
```

3.2.1. Código V1

```
def getAction(self, game_state):

    def value(state, agentIndex, depth, alpha, beta):
```

```
        # ESTADOS TERMINALES (GANAR/PERDER O PROFUNDIDAD
            ALCANZADA)
        if depth == 0 or state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        # AGENTES FANTASMA
        elif agentIndex >= 1:
            return min_value(state, agentIndex, depth, alpha,
                beta)

        # AGENTE PACMAN
        else:
            return max_value(state, depth, alpha, beta)

    def max_value(state, depth, alpha, beta): # SIEMPRE SE
        EJECUTARA PARA PACMAN

        v = float('-inf')
        legalActions = state.getLegalActions(0) # ACCIONES
            POSIBLES PARA PACMAN

        for action in legalActions:
            successor = state.generateSuccessor(0, action)
            v = max(v, value(successor, 1, depth, alpha, beta))
                # ELEGIR EL MAXIMO DE LOS SUCESORES

            # GESTION DE LA PODA
            if v > beta:
                return v
            alpha = max(alpha, v)

        return v

    def min_value(state, agentIndex, depth, alpha, beta):  # SE
        EJECUTA EN LOS DIFERENTES FANTASMAS

        v = float('inf')
        legalActions = state.getLegalActions(agentIndex)

        for action in legalActions:
            successor = state.generateSuccessor(agentIndex,
                action)

            # SI ES EL ULTIMO FANTASMA, DESPUES PACMAN
            if agentIndex == state.getNumAgents() - 1:
                v = min(v, value(successor, 0, depth - 1, alpha,
                    beta))
```

```
            # SI NO LE TOCARA A OTRO FANTASMA
            else:
                v = min(v, value(successor, agentIndex + 1,
                    depth, alpha, beta))

            # GESTION DE LA PODA
            if v < alpha:
                return v
            beta = min(beta, v)

        return v

    legal_actions = game_state.getLegalActions(0) # OBTENER
        ACCIONES POSIBLES PARA PACMAN

    # INICIALIZAR VARIABLES
    best_action = None
    best_value = float('-inf')
    alpha = float('-inf')
    beta = float('inf')

    # CALCULAR UN VALOR VALUE PARA CADA ACCION POSIBLE
    for action in legal_actions:
        sucessor = game_state.generateSuccessor(0, action)
        v = value(sucessor, 1, self.depth, alpha, beta)
        if v > best_value:
            best_value = v
            best_action = action

    return best_action
```

```
Question q3
===========

*** PASS: test_cases\q3\0-eval-function-lose-states-1.test
*** PASS: test_cases\q3\0-eval-function-lose-states-2.test
*** PASS: test_cases\q3\0-eval-function-win-states-1.test
*** PASS: test_cases\q3\0-eval-function-win-states-2.test
*** FAIL: test_cases\q3\0-lecture-6-tree.test
***     Incorrect generated nodes for depth=2
***         Student generated nodes: A B C D E F G H I max min1
    min2 min3
***         Correct generated nodes: A B C D E F G H max min1
    min2 min3
***     Tree:
***                 max
```

```
***            /-/    |    \--\
***           /       |       \
***          /        |        \
***       min1        min2       min3
***        /|\        /|\        /|\
***       / | \      / | \      / | \
***      A  B  C    D  E  F    G  H  I
***      3 12  8    5  4  6   14  1  11
*** FAIL: test_cases\q3\0-small-tree.test
***     Incorrect generated nodes for depth=3
***         Student generated nodes: A B C D deeper minLeft
    minRight root
***         Correct generated nodes: A B C minLeft minRight root
***     Tree:
***               root
***              /    \
***       minLeft   minRight
***        / \         /    \
***       A   B       C   deeper
***       4   3       2     |
***                         D
***                        1000
*** PASS: test_cases\q3\1-1-minmax.test
*** PASS: test_cases\q3\1-2-minmax.test
*** FAIL: test_cases\q3\1-3-minmax.test
***     Incorrect generated nodes for depth=3
***         Student generated nodes: a b1 b2 c3 c4 cx d5 d6 d7
    d8 dx
***         Correct generated nodes: a b1 b2 c3 cx d5 d6 dx
***     Tree:
***                    /------a-------\
***                   /               \
***                  /                 \
***                b1                   b2
***                 |                  /    \
***                cx                c3      c4
***                 |               /   \    /   \
***                dx              d5   d6  d7   d8
***               4.01             4   -7   0    5
***
***     a - max
***     b - min
***     c - max
***
***     Note that the minimax value of b2 is 4.
*** PASS: test_cases\q3\1-4-minmax.test
*** PASS: test_cases\q3\1-5-minmax.test
```

```
*** PASS: test_cases\q3\1-6-minmax.test
*** PASS: test_cases\q3\1-7-minmax.test
*** PASS: test_cases\q3\1-8-minmax.test
*** PASS: test_cases\q3\2-1a-vary-depth.test
*** PASS: test_cases\q3\2-1b-vary-depth.test
*** PASS: test_cases\q3\2-2a-vary-depth.test
*** PASS: test_cases\q3\2-2b-vary-depth.test
*** PASS: test_cases\q3\2-3a-vary-depth.test
*** FAIL: test_cases\q3\2-3b-vary-depth.test
***     Incorrect generated nodes for depth=2
***         Student generated nodes: a b1 b2 c3 c4 cx d5 d6 d7
   d8 dx
***         Correct generated nodes: a b1 b2 c3 cx d5 d6 dx
***     Tree:
***                    /------a-------\
***                   /               \
***                  /                 \
***               b1                     b2
***                |                   /    \
***       5.01    cx           8   c3          c4    5
***                |              /    \      /    \
***               dx            d5    d6    d7    d8
***              5.01            4    -7    0     5
***
***     a - max
***     b - min
***     c - max
***
***     Note that the minimax value of b1 is 4, but the depth=1
   limited value is 5.
***     The values next to c3, c4, and cx are the values of the
   evaluation function, not
***     necessarily the correct minimax backup.
*** PASS: test_cases\q3\2-4a-vary-depth.test
*** FAIL: test_cases\q3\2-4b-vary-depth.test
***     Incorrect generated nodes for depth=2
***         Student generated nodes: a b1 b2 c3 c4 cx d5 d6 d7
   d8 dx
***         Correct generated nodes: a b1 b2 c3 cx d5 d6 dx
***     Tree:
***                    /------a-------\
***                   /               \
***                  /                 \
***               b1                     b2
***                |                   /    \
***       4.99    cx           8   c3          c4    5
***                |              /    \      /    \
```

```
***              dx              d5    d6    d7    d8
***             4.99             4     -7    0     5
***
***      a - max
***      b - min
***      c - max
***
***      Note that the minimax value of b1 is 4, but the depth=1 limited value is 5.
***      The values next to c3, c4, and cx are the values of the evaluation function, not
***      necessarily the correct minimax backup.
*** FAIL: test_cases\q3\2-one-ghost-3level.test
***      Incorrect generated nodes for depth=3
***          Student generated nodes: a b1 b2 c1 c2 c3 c4 d1 d2 d3 d5 d6 d7 d8
***          Correct generated nodes: a b1 b2 c1 c2 c3 d1 d2 d3 d5 d6
***      Tree:
***                      /------a------\
***                     /              \
***                    /                \
***                  b1                  b2
***                /    \              /    \
***              c1      c2          c3      c4
***             /  \    /  \        /  \    /  \
***            d1  d2  d3  d4      d5  d6  d7  d8
***            3   9   10  6       4   7   0   5
***
***      a - max
***      b - min
***      c - max
*** FAIL: test_cases\q3\3-one-ghost-4level.test
***      Incorrect generated nodes for depth=4
***          Student generated nodes: A B C D E F I J K L M N a b1 b2 c1 c2 c3 c4 d1 d2 d3 d5 d6 d7
***          Correct generated nodes: A B C D E F I K a b1 b2 c1 c2 c3 d1 d2 d3 d5 d6
***      Tree:
***                      /------a------\
***                     /              \
***                    /                \
***                  b1                  b2
***                /    \              /    \
***              c1      c2          c3      c4
***             /  \    /  \        /  \    /  \
***            d1  d2  d3  d4      d5  d6  d7  d8
```

```
***      / \  / \  / \  / \   / \  / \  / \  / \
***      A B  C D  E F  G H   I J  K L  M N  O P
***      3 13 5 9 10 11 6 8   1 0  4 7 12 15 2 14
***
***      a - max
***      b - min
***      c - max
***      d - min
*** FAIL: test_cases\q3\4-two-ghosts-3level.test
***      Incorrect generated nodes for depth=3
***          Student generated nodes: a b1 b2 c1 c2 c3 c4 d1 d2
    d3 d4 d5 d6 d7 d8
***          Correct generated nodes: a b1 b2 c1 c2 c3 c4 d1 d2
    d3 d4 d5 d6 d7
***      Tree:
***                    /-------a-------\
***                   /                 \
***                  /                   \
***                b1                     b2
***              /    \                 /    \
***            c1      c2             c3      c4
***           /  \    /  \           /  \    /  \
***          d1  d2  d3  d4         d5  d6  d7  d8
***          3   9   10  6          4   7   0   5
***
***      a - max
***      b - min
***      c - min
*** FAIL: test_cases\q3\5-two-ghosts-4level.test
***      Incorrect generated nodes for depth=4
***          Student generated nodes: A B C D E G H I J K M O a
    b1 b2 c1 c2 c3 c4 d1 d2 d3 d4 d5 d6 d7 d8
***          Correct generated nodes: A B C D E G H I J a b1 b2
    c1 c2 c3 d1 d2 d3 d4 d5
***      Tree:
***                    /-------a-------\
***                   /                 \
***                  /                   \
***                b1                     b2
***              /    \                 /    \
***            c1      c2             c3      c4
***           /  \    /  \           /  \    /  \
***          d1  d2  d3  d4         d5  d6  d7  d8
***         / \  / \  / \  / \     / \  / \  / \  / \
***         A B  C D  E F  G H     I J  K L  M N  O P
***         3 13 5 9 10 11 6 8     1 0  4 7 12 15 2 14
***
```

```
***      a - max
***      b - min
***      c - min
***      d - max
*** PASS: test_cases\q3\6-tied-root.test
*** PASS: test_cases\q3\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after
    0.8332991600036621 seconds.
*** Won 0 out of 1 games. Average score: 84.0 ***
*** FAIL: test_cases\q3\8-pacman-game.test
***      Bug: Wrong number of states expanded.
*** Tests failed.

### Question q3: 0/5 ###
```

3.2.2. Código V-Final

```
def getAction(self, game_state):

    def value(state, agentIndex, depth, alpha, beta):
        # ESTADOS TERMINALES (GANAR/PERDER O PROFUNDIDAD
            ALCANZADA)
        if depth == 0 or state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        # AGENTES FANTASMA
        elif agentIndex >= 1:
            return min_value(state, agentIndex, depth, alpha,
                beta)

        # AGENTE PACMAN
        else:
            return max_value(state, depth, alpha, beta)
```

```
def max_value(state, depth, alpha, beta): # SIEMPRE SE
    EJECUTARA PARA PACMAN

    v = float('-inf')
    legalActions = state.getLegalActions(0) # ACCIONES
        POSIBLES PARA PACMAN

    for action in legalActions:
        successor = state.generateSuccessor(0, action)
        v = max(v, value(successor, 1, depth, alpha, beta))
            # ELEGIR EL MAXIMO DE LOS SUCESORES

        # GESTION DE LA PODA
        if v > beta:
            return v
        alpha = max(alpha, v)

    return v

def min_value(state, agentIndex, depth, alpha, beta):  # SE
    EJECUTA EN LOS DIFERENTES FANTASMAS

    v = float('inf')
    legalActions = state.getLegalActions(agentIndex)

    for action in legalActions:
        successor = state.generateSuccessor(agentIndex,
            action)

        # SI ES EL ULTIMO FANTASMA, DESPUES PACMAN
        if agentIndex == state.getNumAgents() - 1:
            v = min(v, value(successor, 0, depth - 1, alpha,
                beta))

        # SI NO LE TOCARA A OTRO FANTASMA
        else:
            v = min(v, value(successor, agentIndex + 1,
                depth, alpha, beta))

        # GESTION DE LA PODA
        if v < alpha:
            return v
        beta = min(beta, v)

    return v
```

```
    legal_actions = game_state.getLegalActions(0) # OBTENER
        ACCIONES POSIBLES PARA PACMAN

    # INICIALIZAR VARIABLES
    best_action = None
    best_value = float('-inf')
    alpha = float('-inf')
    beta = float('inf')

    # CALCULAR UN VALOR VALUE PARA CADA ACCION POSIBLE
    for action in legal_actions:
        sucessor = game_state.generateSuccessor(0, action)
        v = value(sucessor, 1, self.depth, alpha, beta)
        if v > best_value:
            best_value = v
            best_action = action

        # ACTUALIZAR EL VALOR DE ALPHA
        alpha = max(alpha, best_value)

    return best_action
```

```
Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
```

```
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after
    0.9068129062652588 seconds.
*** Won 0 out of 1 games. Average score: 84.0 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###
```

3.2.3. Problemas y dificultades encarados en el ejercicio

Este ejercicio no resulta demasiado complicado una vez resuelto el anterior ya que la estructura del código es la del Mínimax implementado. Únicamente se han de añadir los parámetros *alpha* y *beta* para que mediante condicionales discrimine que ramas ha de explorar y cuales no. La mayor dificultad, y la diferencia entre la primera y la última versión, es la actualización de *alpha* tras el calculo del valor *value* para cada acción posible.

3.2.4. Ejemplo (Figura 3)

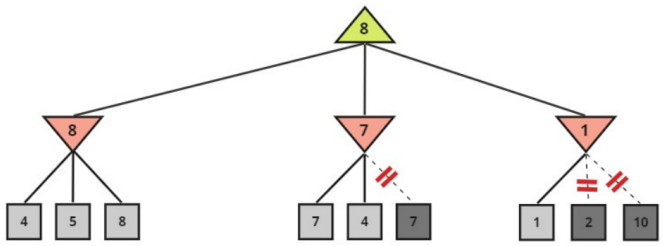

Figura 3: Ejemplo algoritmo Minimax con poda *Alpha-Beta*

4. Pregunta 4: Expectimax

4.1. Descripción

El algoritmo Expetimax es una alternativa al algoritmo Mínimax que se emplea en situaciones donde los agentes minimizadores no realizan siempre la acción óptima. Esto es, la actuación de los agentes es probabilística y se efectuara de una forma u otra en base a unas probabilidades.

Ante este problema se plantea que los sucesores de un estado son siempre equiprobables, por lo tanto la probabilidad de que cada uno de esos sucesores sera el inverso del numero de sucesores que el estado tenga: $\frac{1}{len(legalActions)}$

En un árbol de estados Expectimax no se puede aplicar el método de poda *Alpha-Beta* salvo en un caso en particular, en el que el valor que la función de evaluación devuelve para los estados sea comprendido en un rango finito. Si los valores están comprendidos en un rango finito se puede deducir que los futuros nodos no vayan a proporcionar un valor que supere el rango definido. En este escenario, la poda *Alpha-Beta* puede descartar las ramas del árbol que no tienen la posibilidad de mejorar la decisión actual, lo que reduce significativamente la cantidad de nodos que deben evaluarse y, por lo tanto, mejora la eficiencia de la búsqueda.

Solamente necesita:

- Una función de utilidad para evaluar la puntuación de los nodos.
- Un árbol que represente los estados posibles y la unión entre ellos.
- Una profundidad de búsqueda. Valor *depth*.

Es importante recalcar:

- Expectimax es especialmente adecuado para juegos y situaciones en las que existe incertidumbre o elementos de azar.
- Aunque no es tan directo como Minimax para la aplicación de la poda *Alpha-Beta*, Expectimax puede beneficiarse de esta técnica para reducir la complejidad de la búsqueda, especialmente cuando los valores de los estados están en un rango finito.

4.1.1. Tipo de Algoritmo

- Algoritmo de búsqueda **adversarial**

4.2. Algoritmo

```
def getAction(self, game_state):
    initialize best_action = None
    initialize best_value = -$\infty$
    FOR acciones posibles:
        v = value(sucesor)
        if v > best_value:
            best_value = v
            best_action = action
    return best_action

def value(state):
    SI profundidad = 0 o ganar o perder:
        return funcion de evaluaci n
    SI agente = fantasma:
        return exp_value(state)
    SINO:
        return max_value(state)

def max_value(state):
    initialize v= -$\infty$
    FOR each sucessor of state:
        v= max (v,value(sucessor))
    return v

def exp_value(state):
    initialize v = 0
    FOR each sucessor of state:
        p= probability (sucessor)
        v+= p* value(sucessor)
    return v
```

4.2.1. Código V1

```
def getAction(self, gameState):

    def value(state, agentIndex, depth):

        # ESTADOS TERMINALES (GANAR/PERDER O PROFUNDIDAD
            ALCANZADA)
        if depth == 0 or state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        # AGENTES FANTASMA
        elif agentIndex >= 1:
```

```
            return exp_value(state, agentIndex, depth)

        # AGENTE PACMAN
        else:
            return max_value(state, depth)

    def max_value(state, depth):  # SIEMPRE SE EJECUTARA PARA
        PACMAN

        v = float('-inf')
        legalActions = state.getLegalActions(0)  # ACCIONES
            POSIBLES PARA PACMAN

        for action in legalActions:
            successor = state.generateSuccessor(0, action)
            v = max(v, value(successor, 1, depth))  # ELEGIR EL
                MAXIMO DE LOS SUCESORES

        return v

    def exp_value(state, agentIndex, depth):  # SE EJECUTA EN
        LOS DIFERENTES FANTASMAS

        v = 0
        legalActions = state.getLegalActions(agentIndex)
        p = 1 / len(legalActions)

        for action in legalActions:
            successor = state.generateSuccessor(agentIndex,
                action)

            # SI ES EL ULTIMO FANTASMA, DESPUES PACMAN
            if agentIndex == state.getNumAgents() - 1:
                v += p * value(successor, 0, depth - 1)

            # SI NO LE TOCARA A OTRO FANTASMA
            else:
                v += p * value(successor, agentIndex + 1, depth)

        return v

    legal_actions = gameState.getLegalActions(0)  # OBTENER
        ACCIONES POSIBLES PARA PACMAN

    # INICIALIZAR VARIABLES
    best_action = None
    best_value = float('-inf')
```

```
    # CALCULAR UN VALOR VALUE PARA CADA ACCION POSIBLE
    for action in legal_actions:
        sucessor = gameState.generateSuccessor(0, action)
        v = value(sucessor, 1, self.depth)
        if v > best_value:
            best_value = v
            best_action = action

    return best_action
```

```
Question q4
===========

*** PASS: test_cases/q4/0-eval-function-lose-states-1.test
*** PASS: test_cases/q4/0-eval-function-lose-states-2.test
*** PASS: test_cases/q4/0-eval-function-win-states-1.test
*** PASS: test_cases/q4/0-eval-function-win-states-2.test
*** PASS: test_cases/q4/0-expectimax1.test
*** PASS: test_cases/q4/1-expectimax2.test
*** PASS: test_cases/q4/2-one-ghost-3level.test
*** PASS: test_cases/q4/3-one-ghost-4level.test
*** PASS: test_cases/q4/4-two-ghosts-3level.test
*** PASS: test_cases/q4/5-two-ghosts-4level.test
*** PASS: test_cases/q4/6-1a-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-1b-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-1c-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q4/6-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q4/6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after
    0.5001223087310791 seconds.
*** Won 0 out of 1 games. Average score: 84.0 ***
*** PASS: test_cases/q4/7-pacman-game.test

### Question q4: 5/5 ###
```

4.2.2. Código V-Final

```
def getAction(self, gameState):

    def value(state, agentIndex, depth):

        # ESTADOS TERMINALES (GANAR/PERDER O PROFUNDIDAD
            ALCANZADA)
        if depth == 0 or state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        # AGENTES FANTASMA
        elif agentIndex >= 1:
            return exp_value(state, agentIndex, depth)

        # AGENTE PACMAN
        else:
            return max_value(state, depth)

    def max_value(state, depth):  # SIEMPRE SE EJECUTARA PARA
        PACMAN

        v = float('-inf')
        legalActions = state.getLegalActions(0)  # ACCIONES
            POSIBLES PARA PACMAN

        for action in legalActions:
            successor = state.generateSuccessor(0, action)
            v = max(v, value(successor, 1, depth))  # ELEGIR EL
                MAXIMO DE LOS SUCESORES

        return v

    def exp_value(state, agentIndex, depth):  # SE EJECUTA EN
        LOS DIFERENTES FANTASMAS

        v = 0
        legalActions = state.getLegalActions(agentIndex)
        p = 1 / len(legalActions)

        for action in legalActions:
            successor = state.generateSuccessor(agentIndex,
                action)

            # SI ES EL ULTIMO FANTASMA, DESPUES PACMAN
            if agentIndex == state.getNumAgents() - 1:
                v += p * value(successor, 0, depth - 1)
```

```
            # SI NO LE TOCARA A OTRO FANTASMA
            else:
                v += p * value(successor, agentIndex + 1, depth)

        return v

    legal_actions = gameState.getLegalActions(0)  # OBTENER
       ACCIONES POSIBLES PARA PACMAN

    # INICIALIZAR VARIABLES
    best_action = None
    best_value = float('-inf')

    # CALCULAR UN VALOR VALUE PARA CADA ACCION POSIBLE
    for action in legal_actions:
        sucessor = gameState.generateSuccessor(0, action)
        v = value(sucessor, 1, self.depth)
        if v > best_value:
            best_value = v
            best_action = action

    return best_action
```

```
Question q4
===========

*** PASS: test_cases/q4/0-eval-function-lose-states-1.test
*** PASS: test_cases/q4/0-eval-function-lose-states-2.test
*** PASS: test_cases/q4/0-eval-function-win-states-1.test
*** PASS: test_cases/q4/0-eval-function-win-states-2.test
*** PASS: test_cases/q4/0-expectimax1.test
*** PASS: test_cases/q4/1-expectimax2.test
*** PASS: test_cases/q4/2-one-ghost-3level.test
*** PASS: test_cases/q4/3-one-ghost-4level.test
*** PASS: test_cases/q4/4-two-ghosts-3level.test
*** PASS: test_cases/q4/5-two-ghosts-4level.test
*** PASS: test_cases/q4/6-1a-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-1b-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-1c-check-depth-one-ghost.test
*** PASS: test_cases/q4/6-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q4/6-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q4/6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
```

```
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after
    0.5001223087310791 seconds.
*** Won 0 out of 1 games. Average score: 84.0 ***
*** PASS: test_cases/q4/7-pacman-game.test

### Question q4: 5/5 ###
```

4.2.3. Problemas y dificultades encarados en el ejercicio

El desarrollo del algoritmo Expectimax no ha supuesto un problema, pues partiendo de el código escrito para ejecutar el algoritmo Minimax solo se han de aplicar unos cambios mínimos sobre la función *minvalue()*.

4.2.4. Ejemplo (Figura 4)

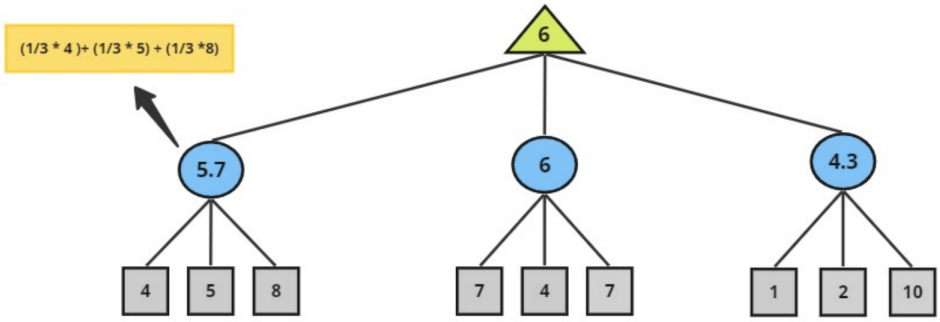

Figura 4: Ejemplo algoritmo Expectimax (nodos terminales equiprobables)

5. Pregunta 5: Función de evaluación

5.1. Descripción

Como se ha mencionado anteriormente, las funciones de evaluación son muy útiles para determinar cuan beneficioso es un estado respecto a los intereses de un agente cuando el árbol de estados se torna demasiado extenso.

En contraste con la pregunta 1, es este caso no se conoce la acción, únicamente se dispone de un estado que debe de ser evaluado. Asimismo, se propone encontrar una solución de mayor calidad y eficiencia respecto a lo obtenido en la pregunta 1.

Solamente necesita:

- El objetivo del agente, para poder determinar que estados le convienen y cuales no.
- El estado actual del agente para evaluarlo.

Es importante recalcar:

- Se presupone que la función evaluadora aquí implementada debe de mejorar la desarrollada en la pregunta 1.

5.2. Pseudocódigo de la función de evaluación

```
funcionEvaluadora():

    dist_c = Calcular distancia a la comida mas cercana (incluir
        pellets)
    dist_f = Calcular la distancia al fantasma mas cercano

    SI existe algun fantasma asustado:
        Actualizar las distancias de comida (fantasmas asustados
            se consideran comida)

    SI dist_f == dist_c:
        RETURN -1 / dist_f

    SI dist_f > dist_c:
        RETURN 1 / dist_c

    SI NO
        RETURN -1 / dist_c
```

5.2.1. Código V1

```
def betterEvaluationFunction(currentGameState):

     # OBTENER LA INFORMACION NECESARIA

    newPos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    newGhostStates = currentGameState.getGhostStates()

    # ENCONTRAR LA COMIDA O CAPSULA MAS CERCANA

    food_min_distance=0
    foods = newFood.asList() + currentGameState.getCapsules()
    for food in foods:
        food_dist=util.manhattanDistance(food,newPos)
        if(food_min_distance>=food_dist or food_min_distance==0)
            :
             food_min_distance=food_dist

    # ENCONTRAR EL FANTASMA MAS CERCANO

    ghost_min_distance=0
    for ghostState in newGhostStates:
        ghost_dist=util.manhattanDistance(ghostState.getPosition
            (),newPos)
        if(ghost_min_distance>=ghost_dist or ghost_min_distance
            ==0):
             ghost_min_distance=ghost_dist

    # EVALUAR LA SITUACION

    if food_min_distance == 0:
        return sys.maxsize + currentGameState.getScore()
    elif ghost_min_distance == 0:
        return -sys.maxsize + currentGameState.getScore()
    elif food_min_distance == ghost_min_distance:
        return -1 /ghost_min_distance + currentGameState.
            getScore()
    elif food_min_distance < ghost_min_distance:
        return 1/food_min_distance + currentGameState.getScore()
    else:
        return -food_min_distance + currentGameState.getScore()
```

```
Question q5
===========
```

```
Pacman emerges victorious! Score: 901
Pacman emerges victorious! Score: 1294
Pacman emerges victorious! Score: 993
Pacman emerges victorious! Score: 897
Pacman emerges victorious! Score: 909
Pacman emerges victorious! Score: 672
Pacman emerges victorious! Score: 965
Pacman emerges victorious! Score: 1228
Pacman emerges victorious! Score: 852
Pacman emerges victorious! Score: 1162
Average Score: 987.3
Scores:        901.0, 1294.0, 993.0, 897.0, 909.0, 672.0, 965.0,
    1228.0, 852.0, 1162.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** FAIL: test_cases\q5\grade-agent.test (5 of 6 points)
***     987.3 average score (1 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

### Question q5: 5/6 ###
```

5.2.2. Código V-Final

```
def betterEvaluationFunction(currentGameState):

    # OBTENER LA INFORMACION NECESARIA

    pacman_pos = currentGameState.getPacmanPosition()
    newFood = currentGameState.getFood()
    newGhostStates = currentGameState.getGhostStates()
    newScaredTimes = [ghostState.scaredTimer for ghostState in
        newGhostStates]
```

```
    # ENCONTRAR LA COMIDA O CAPSULA MAS CERCANA

    food_min_distance=0
    foods = newFood.asList()+currentGameState.getCapsules()
    for food in foods:
        food_dist = util.manhattanDistance(food,pacman_pos)
        if (food_min_distance >= food_dist or food_min_distance
            == 0):
            food_min_distance = food_dist

    # ENCONTRAR EL FANTASMA MAS CERCANO (LOS FANTASMAS ASUSTADOS
        SE CONSIDERA COMIDA)

    ghost_min_distance=0
    for i,ghostState in enumerate(newGhostStates):
        if (newScaredTimes[i] == 0):
            ghost_distance = util.manhattanDistance(ghostState.
                getPosition(),pacman_pos)
            if(ghost_min_distance >= ghost_distance or
                ghost_min_distance==0):
                ghost_min_distance = ghost_distance
        else:
            scared_ghosts = util.manhattanDistance(ghostState.
                getPosition(),pacman_pos)
            if(food_min_distance >= scared_ghosts or
                food_min_distance == 0):
                food_min_distance = scared_ghosts

    if ghost_min_distance == 0:
        ghost_min_distance = sys.maxsize

    if food_min_distance == 0:
        return sys.maxsize + currentGameState.getScore()
    elif ghost_min_distance == 0:
        return -sys.maxsize + currentGameState.getScore()
    elif food_min_distance == ghost_min_distance:
        return -1 / ghost_min_distance + currentGameState.
            getScore()
    elif food_min_distance < ghost_min_distance:
        return 1/ food_min_distance + currentGameState.getScore
            ()
    else:
        return -food_min_distance + currentGameState.getScore()
```

```
Question q5
===========
```

```
Pacman emerges victorious! Score: 1101
Pacman emerges victorious! Score: 1296
Pacman emerges victorious! Score: 1097
Pacman emerges victorious! Score: 1126
Pacman emerges victorious! Score: 1190
Pacman emerges victorious! Score: 1095
Pacman emerges victorious! Score: 1305
Pacman emerges victorious! Score: 1025
Pacman emerges victorious! Score: 1274
Pacman emerges victorious! Score: 1118
Average Score: 1162.7
Scores:        1101.0, 1296.0, 1097.0, 1126.0, 1190.0, 1095.0,
    1305.0, 1025.0, 1274.0, 1118.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q5\grade-agent.test (6 of 6 points)
***      1162.7 average score (2 of 2 points)
***          Grading scheme:
***           < 500:  0 points
***          >= 500:  1 points
***          >= 1000:  2 points
***      10 games not timed out (1 of 1 points)
***          Grading scheme:
***           < 0:  fail
***          >= 0:  0 points
***          >= 10:  1 points
***      10 wins (3 of 3 points)
***          Grading scheme:
***           < 1:  fail
***          >= 1:  1 points
***          >= 5:  2 points
***          >= 10:  3 points

### Question q5: 6/6 ###
```

5.2.3. Problemas y dificultades encarados en el ejercicio

A decir verdad, este ejercicio no ha resultado complicado, puesto que empleando el código que se ha implementado para la pregunta 1 el resultado obtenido es bastante bueno. Sin embargo, no se obtiene la mejor puntuación del *autograder* con el *Código-V1*.

Como medida para mejorar la función de evaluación se ha contemplado el hecho de que los agentes fantasma puedan estar asustados, y en tal caso se consideran como comida para el agente pacman. Entonces, a la hora de establecer la distancia mínima a la comida, puede que un fantasma asustado cercano ocupe su lugar. Además se han considerado

como parte de la comida también los *pellets*.

Con esta ultima mejora implementada en el *Código-VFinal* se consigue la puntuación máxima en el autograder.

5.2.4. Ejemplo (Figura 5)

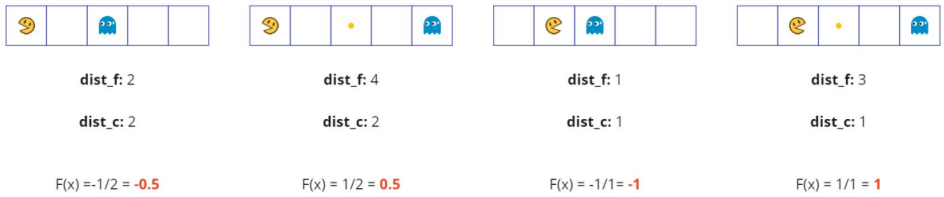

Figura 5: Ejemplo de uso de la función evaluadora