Artificial intelligence - Project 1
- Search problems -

Andriescu ANtoino

01/11/2021

1 Uninformed search

1.1 Question 1 - Depth-first search

In this section the solution for the following problem will be presented:

"In search.py, implement ***Depth-First search(DFS) algorithm*** *in function depthFirstSearch. Don't forget that DFS graph search is graph-search with the frontier as a LIFO queue(Stack)."*.

1.1.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1  def depthFirstSearch(problem):
2      frontier = util.Stack();
3      expanded = []
4      initial_state = problem.getStartState()
5      frontier.push((initial_state,[]))
6
7      while(not frontier.isEmpty()):
8          popped_element = frontier.pop()
9          current_state,actions = popped_element
10
11         if(current_state not in expanded):
12             expanded.append(current_state)
13             if(problem.isGoalState(current_state)):
14                 return actions
15
16             for successor in problem.expand(current_state):
17                 next_pos, next_action, cost = successor
18                 frontier.push((next_pos,actions+[next_action]))
19
20     util.raiseNotDefined()
```

Explanation:

- in linia 2 initializam frontier ca o stiva
- expanded va contine toate starile care au fost verificate
- la liniile 4 si 5 luam starea initiala a lui pacman si o adugam in lista frontier
- de la linia 7, atat timp cat lista frontier nu este goala, vom adauga starea actuala in lista expanded, daca nu este deja in lista, pentru a sti ca am trecut deja prin ea
- in linia 13 verificam daca am ajuns la goal, iar daca am ajuns, returnam lista de actiuni necesare pentru a ajunge in acest punct
- folosim urmatorul for pentru a lua toate pozitiile urmatoare posibile si adaugam in frontiera pozitia viitoare si actiunile necesare pentru a ajunge acolo

Commands:

- python pacman.py -l tinyMaze -p SearchAgent

- python pacman.py -l mediumMaze -p SearchAgent
- python pacman.py -l bigMaze -z .5 -p SearchAgent

1.1.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Is the found solution optimal? Explain your answer.
A1:

Q2: Run *autograder python autograder.py* and write the points for Question 1.
A2: 4/4

1.1.3 Personal observations and notes

1.2 Question 2 - Breadth-first search

In this section the solution for the following problem will be presented:

*"In **search.py**, implement the **Breadth-First search** algorithm in function breadthFirstSearch."*.

1.2.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1  def breadthFirstSearch(problem):
2      frontier = util.Queue();
3      expanded = []
4      initial_state = problem.getStartState()
5      frontier.push((initial_state, []))
6
7      while (not frontier.isEmpty()):
8          popped_element = frontier.pop()
9          current_state, actions = popped_element
10
11         if (current_state not in expanded):
12             expanded.append(current_state)
13             if (problem.isGoalState(current_state)):
14                 return actions
15             for successor in problem.expand(current_state):
16                 next_pos, next_action, cost = successor
17                 frontier.push((next_pos, actions + [next_action]))
18     util.raiseNotDefined()
```

Explanation:

- in linia 2 initializam frontier ca o lista FIFO(first-in-first-out)

- expanded va contine toate starile care au fost verificate
- la liniile 4 si 5 luam starea initiala a lui pacman si o adugam in lista frontier
- de la linia 7, atat timp cat lista frontier nu este goala, vom adauga starea actuala in lista expanded, daca nu este deja in lista, pentru a sti ca am trecut deja prin ea
- in linia 13 verificam daca am ajuns la goal, iar daca am ajuns, returnam lista de actiuni necesare pentru a ajunge in acest punct
- folosim urmatorul for pentru a lua toate pozitiile urmatoare posibile si adaugam in frontiera pozitia viitoare si actiunile necesare pentru a ajunge acolo

Commands:

- python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
- python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

1.2.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Is the found solution optimal? Explain your answer.
A1:

Q2: Run autograder *python autograder.py* and write the points for Question 2.
A2: 4/4

1.2.3 Personal observations and notes

1.3 Question 3 - Uniform-cost search

In this section the solution for the following problem will be presented:

*"In search.py, implement **Uniform-cost graph search** algorithm in uniformCostSearchfunction"*

1.3.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1 def uniformCostSearch(problem):
2 
3     "*** YOUR CODE HERE ***"
4     util.raiseNotDefined()
```

Explanation:

-

Commands:

-

1.3.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Compare the results to the ones obtained with DFS. Are the solutions different? Is the number of extended (explored) states smaller? Explain your answer.
A1:

Q2: Consider that some positions are more desirable than others. This can be modeled by a cost function which sets different values for the actions of stepping into positions. Identify in **searchAgents.py** the description of agents StayEastSearchAgent and StayWestSearchAgent and analyze the cost function. Why the cost .5 ** x for stepping into (x,y) is associated to StayWestAgen.
A2:

Q3: Run autograder *python autograder.py* and write the points for Question 3.
A3:

1.3.3 Personal observations and notes

1.4 References

2 Informed search

2.1 Question 4 - A* search algorithm

In this section the solution for the following problem will be presented:

*"Go to aStarSearch in search.py and implement **A* search algorithm**. A* is graphs search with the frontier as a priorityQueue, where the priority is given bythe function g=f+h".*

2.1.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```python
def aStarSearch(problem, heuristic=nullHeuristic):
    frontier = util.PriorityQueue();
    expanded = []
    initial_state = problem.getStartState()
    frontier.push((initial_state,[],0 ),0)

    while(not frontier.isEmpty()):
        popped_element = frontier.pop()
        current_state,actions,total_cost = popped_element

        if current_state not in expanded:
            expanded.append(current_state)

            if(problem.isGoalState(current_state)):
                return actions
            for successors in problem.expand(current_state):
                next_pos, next_action, next_cost = successors
                new_cost = total_cost + next_cost
                f = new_cost + heuristic(next_pos,problem)
                frontier.push((next_pos,actions + [next_action], new_cost),f)

    util.raiseNotDefined()
```

Listing 1: Solution for the A* algorithm.

Explanation:

- A* este un algoritm de cautare a drumului de cost minim. Acesta foloseste o functie de estimare a costului din fiecare nod pana la goal: f(n)=g(n)+h(n), unde g(n) este costul drumului de la nodul initial pana la nodul curent, iar h(n) este estimarea costului drumului pana la goal.
- frontier este declarat ca o lista de prioritate din care putem scoate elementul cu euristica cea mai mica. Fiecare element introdus in frontier are ca parametri starea curenta, lista de actiuni necesare pentru a ajunge la acea stare, costul si euristica (f(n))
- pentru a putea gasi solutia trecem prin fiecare stare, calculandu costul si euristica tuturor starilor posibil urmatoare si le adaugam in coada

- in lista expanded salvam toate starile prin care trecem pentru a nu trece de doua ori prin aceleasi
- la linia 14 verificam daca starea curenta este goal si daca este returnam lista cu actiunile facute pentru a ajunge in acea pozitie
- in for-ul de la linia 16 pentru fiecare succesor posibil calculam costul in "new-cost", euristica in "f" si lista de actiuni, dupa care introducem toate datele, inclusiv pozitia, in frontiera

Commands:

- python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

2.1.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Does A* and UCS find the same solution or they are different?
A1:

Q2: Does A* finds the solution with fewer expanded nodes than UCS?
A2:

Q3: Does A* finds the solution with fewer expanded nodes than UCS?
A3:

Q4: Run autograder *python autograder.py* and write the points for Question 4 (min 3 points).
A4: 4/4

2.1.3 Personal observations and notes

2.2 Question 5 - Find all corners - problem implementation

In this section the solution for the following problem will be presented:

*"Pacman needs to find the shortest path to visit all the corners,regardless there is food dot there or not. Go to **CornersProblem** in searchAgents.py and propose a representation of the state of this search problem. It might help to look at the existing implementation for PositionSearchProblem. The representation should include only the information necessary to reach the goal. Read carefully the comments inside the class CornersProblem."*.

2.2.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
class CornersProblem(search.SearchProblem):

    def __init__(self, startingGameState):

        self.walls = startingGameState.getWalls()
        self.startingPosition = startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners = ((1,1), (1,top), (right, 1), (right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print('Warning: no food in corner ' + str(corner))
        self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded

    def getStartState(self):
        state = (self.startingPosition, (0, 1, 2, 3))
        return state
        util.raiseNotDefined()

    def isGoalState(self, state):
        return not state[1]
        util.raiseNotDefined()

    def expand(self, state):
        children = []
        for action in self.getActions(state):
            x, y = state[0]
            dx, dy = Actions.directionToVector(action)
            nextX, nextY = int(x + dx), int(y + dy)

            if not self.walls[nextX][nextY]:
                remainedCorners = state[1] #lista cu colturile ramase
                nextLocation = (nextX, nextY)
                try:
                    idx = self.corners.index(nextLocation)
                except:
                    pass
                else:
                    if idx in remainedCorners:
                        temp = list(remainedCorners)
                        temp.remove(idx) #stergem colturile gasite
                        remainedCorners = tuple(temp)

                nextState = (nextLocation, remainedCorners)
                children.append((nextState, action, 1))
        self._expanded += 1
        return children

    def getActions(self, state):
        possible_directions = [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]
        valid_actions_from_state = []
        for action in possible_directions:
            x, y = state[0]
            dx, dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            if not self.walls[nextx][nexty]:
                valid_actions_from_state.append(action)
        return valid_actions_from_state

    def getActionCost(self, state, action, next_state):
        assert next_state == self.getNextState(state, action), (
            "Invalid next state passed to getActionCost().")
        return 1

    def getNextState(self, state, action):
        assert action in self.getActions(state), (
            "Invalid action passed to getActionCost().")
        x, y = state[0]
        dx, dy = Actions.directionToVector(action)
        nextx, nexty = int(x + dx), int(y + dy)

        util.raiseNotDefined()

    def getCostOfActionSequence(self, actions):

        if actions == None: return 999999
        x,y= self.startingPosition
        for action in actions:
            dx, dy = Actions.directionToVector(action)
            x, y = int(x + dx), int(y + dy)
            if self.walls[x][y]: return 999999
        return len(actions)
```

Explanation:

- In "getStartState" declaram state cu pozitia de start si pozitiile colturilor pe care urmeaza sa le aflam
- In "expand" returna starea unui succesor, actiunile necesare pentru a ajunge in acea stare si costul 1. Initializam "children[]", lista in care salvam informatiile despre succesor.
- Verificam daca starea urmatoare nu este perete, iar in acest caz salvam in remainedCorners lista cu colturile ramase si initializam urmatoarea locatie.
- Verificam daca succesorul este colt si ii salvam indexul in idx
- daca idx este in remainedCoreners atunci cream o lista temporara "temp" in care punem lista cu colturile ramase. Stergem din temp coltul cu indexul idx si declaram remainedCorners cu noua lista obtinuta
- In final expandam lista children cu starea prin care vom trece
- In functia "getActions" returnam actiunile valide pe care le poate face pacman din starea curenta. Declaram starea urmatoare folosind starea curenta si vectorul de directie dupa care verificam daca starea viitoare nu este zid, caz in care o adaugam la lista cu actiuni valide ("valid-actions-from-state")
- Returnam lista cu actiuni valide
- In functia "getActionCost" returnam 1 deoarece costul unei actiuni este 1.

Commands:

- python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
- python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

2.2.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: For mediumCorners, BFS expands a big number - around 2000 search nodes. It's time to see that A* with an admissible heuristic is able to reduce this number. Please provide your results on this matter. (Number of searched nodes).

A1: Number of searched nodes = 1966

2.2.3 Personal observations and notes

2.3 Question 6 - Find all corners - Heuristic definition

In this section the solution for the following problem will be presented:

"Implement a consistent heuristic for CornersProblem. Go to the function ***cornersHeuristic*** *in searchAgent.py."*.

2.3.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1  def cornersHeuristic(state, problem):
2      corners = problem.corners # These are the corner coordinates
3      walls = problem.walls # These are the walls of the maze, as a Grid (game.py)
4  
5      position = state[0]
6      cornersIndices = state[1]
7      if not cornersIndices:
8          return 0
9  
10     return max([util.manhattanDistance(position, corners[idx]) for idx in cornersIndices])
```

Explanation:

- Pentru ca aceasta euristica sa fie consistenta am decis sa returnez distanta cea mai mare de la starea initiala la unul dintre colturi
- La linia 5 am declarat pozitia curenta ca fiind state[0], adica pozitia de start.
- cornerIndices primeste state[1], care este pozitia fiecarui colt
- vom return distanta maxima dintre distantele Manhattan dintre fiecare colt si starea initiala
- Distanta Manhatten este distanta absoluta dintre oricare doua puncte ($|x1 - x2| + |y1 - y2|$)

Commands:

- python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5

2.3.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Test with on the mediumMaze layout. What is your number of expanded nodes?
A1: Au fost expandate 1136 de noduri.

2.3.3 Personal observations and notes

2.4 Question 7 - Eat all food dots - Heuristic definition

In this section the solution for the following problem will be presented:

"Propose a heuristic for the problem of eating all the food-dots. The problem of eating all food-dots is already implemented in ***FoodSearchProblem*** *in searchAgents.py."*.

2.4.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1  def foodHeuristic(state, problem):
2      position, foodGrid = state
3      foods = foodGrid.asList()
4      if not foods:
5          return 0
6
7      maxDistance = 0
8      for food in foods:
9          key = position + food
10         if key in problem.heuristicInfo:
11             distance = problem.heuristicInfo[key]
12         else:
13             distance = mazeDistance(position, food, problem.startingGameState)
14             problem.heuristicInfo[key] = distance
15
16         if distance > maxDistance:
17             maxDistance = distance
18
19     return maxDistance
```

Explanation:

- Pentru ca euristica sa fie consistenta am decis ca aceasta sa fie distanta de la starea curenta a lui pacman si cel mai indepartat punct de mancare.
- In foodGrid.asList() sunt salvate toate coordonatele punctelor de mancare, asa ca am initializat foods ca fiind aceasta lista
- daca nu exista nimic in foods atunci se returneaza valoarea "0"
- am initializat distanta maxima (maxDistance) cu 0
- pentru fiecare obiect din foods am salvat o cheie (key) ca fiind suma dintre pozitia curenta si pozitia punctului de mancare
- mai departe voi salva toate distantele dintre pacman si punctele de mancare in biblioteca heuristicInfo, fiecare distanta pana la un punct de mancare diferit sa aiba o cheie diferita prin care sa pot accesa acea distanta
- la randul 13 calculez distanta folosind functia mazeDistance care poate calcula dinstanta dintre oricare doua puncte

- testez pentru fiecare punct daca distanta este mai mare decat maxDistance, caz in care aceasta va fi actualizata

Commands:

- python pacman.py -l testSearch -p AStarFoodSearchAgent
- python pacman.py -l trickySearch -p AStarFoodSearchAgent

2.4.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Test with autograder *python autograder.py*. Your score depends on the number of expanded states by A* with your heuristic. What is that number?
A1: 5/4

2.4.3 Personal observations and notes

2.5 References

3 Adversarial search

3.1 Question 8 - Improve the ReflexAgent

In this section the solution for the following problem will be presented:

"Improve the ReflexAgent such that it selects a better action. Include in the score food locations and ghost locations. The layout testClassic should be solved more often.".

3.1.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1   def evaluationFunction(self, currentGameState, action):
2
3       # Useful information you can extract from a GameState (pacman.py)
4       childGameState = currentGameState.getPacmanNextState(action)
5       newPos = childGameState.getPacmanPosition()
6       newFood = childGameState.getFood()
7       newGhostStates = childGameState.getGhostStates()
8       newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]
9
10      "*** YOUR CODE HERE ***"
11      if childGameState.isWin():
12          return 9999999
13
14      closestFood = min([manhattanDistance(newPos, food) for food in newFood.asList()])
15
16      for ghost in newGhostStates:
17          if ghost.scaredTimer == 0 and manhattanDistance(ghost.getPosition(), newPos) < 2:
18              return -9999999
19
20      return childGameState.getScore() + 1 / closestFood
```

Explanation:

- La liniie 11-12 returnez valoarea 9999999 in cazul in care pacman se afla deja in starea in care castiga jocul.
- Mai departe am calculat distanta de la pacman pana la cel mai apropiat punct de mancare utilizand Distanta Manhatten.
- In continuare am returnat valoarea cea mai mica in cazul in care distanta dintre starea urmatoare a lui pacman si fantoma (in cazul in care fantoma nu este speriata) este mai mica decat 2, pentru ca pacman sa nu mearga spre fantome.
- La final am returnat suma dintre scorul final si catul dintre 1 si distanta la cel mai apropiat punct de mancare

Commands:

- python pacman.py -p ReflexAgent -l testClassic
- python pacman.py –frameTime 0 -p ReflexAgent -k 1
- python pacman.py –frameTime 0 -p ReflexAgent -k 2

3.1.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Test your agent on the openClassic layout. Given a number of 10 consecutive tests, how many types did your agent win? What is your average score (points)?

A1: Dupa cele 10 teste consecutive agentul le-a castigat pe toate 10, cu o medie de 1237 de puncte.

3.1.3 Personal observations and notes

3.2 Question 9 - H-Minimax algorithm

In this section the solution for the following problem will be presented:

" Implement H-Minimax algorithm in MinimaxAgentclass from multiAgents.py. Since it can be more than one ghost, for each max layer there are one ormore min layers.".

3.2.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```python
class MinimaxAgent(MultiAgentSearchAgent):
    """
    Your minimax agent (question 2)
    """

    def maxValue(self, depth, state):
        actions = state.getLegalActions(0)
        if (depth > self.depth) or state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        maxEval = -9999999

        for action in actions:
            child = state.getNextState(0, action)
            maxEval = max(maxEval, self.minValue(depth, child, 1))
        return maxEval

    def minValue(self, depth, state, agentIndex):
        actions = state.getLegalActions(agentIndex)
        if state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        minEval = 9999999

        agentCount = state.getNumAgents()
        if agentIndex <= agentCount - 2:
            for action in actions:
                child = state.getNextState(agentIndex, action)
                minEval = min(minEval, self.minValue(depth, child, agentIndex + 1))
        else:
            for action in actions:
                child = state.getNextState(agentIndex, action)
                minEval = min(minEval, self.maxValue(depth + 1, child))

        return minEval

    def getAction(self, gameState):
        "*** YOUR CODE HERE ***"
        agent = 0

        bestActionValue = -9999999
        bestAction = None
        for action in gameState.getLegalActions(0):
            nextState = gameState.getNextState(0, action)
            if self.minValue(1, nextState, 1) > bestActionValue:
                bestActionValue = self.minValue(1, nextState, 1)
                bestAction = action
        return bestAction
```

Explanation:

- In prima parte am implementat functiile minValue si MaxValue. Functia maxValue are rolul de a gasi valoarea maxima posibila pe care o poate lua agentul, iar functia minValue are rolul de a gasi valoarea minima posibila pe care o poate lua agentul. In functia maxValue caculam valoarea pentru un singur agent, dar in minValue trebuie sa calculam pentru toti agentii posibili deoarece pot fi una sau mai multe fantome.
- In functia getAction initializam bestActionValue cu -9999999 si bestAction cu None. Mai departe, pentru fiecare actiune din lista de actiuni posibile salvam in nextState starea urmatoarea a agentului. Mai departe salvam cea mai mare valoare dintre valorile fantomelor in bestActionValue si actiunea respectiva in bestAction, iar la final returnam bestAction.

Commands:

- python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
- python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

3.2.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Test Pacman on trappedClassic layout and try to explain its behaviour. Why Pacman rushes to the ghost?

A1:

3.2.3 Personal observations and notes

3.3 Question 10 - Use $\alpha - \beta$ pruning in AlphaBetaAgent

In this section the solution for the following problem will be presented:

*" Use alpha-beta prunning in **AlphaBetaAgent** from multiagents.py for a more efficient exploration of minimax tree."*.

3.3.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning (question 3)
    """

    def maxValue(self, depth, state, a, b):
        legalActions = state.getLegalActions(0)
        if (
                depth > self.depth) or state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        maxEval = -9999999

        for action in legalActions:
            child = state.getNextState(0, action)
            maxEval = max(maxEval, self.minValue(depth, child, 1, a, b))
            if maxEval > b:
                return maxEval
            a = max(a, maxEval)
        return maxEval

    def minValue(self, depth, state, agentIndex, a, b):
        legalActions = state.getLegalActions(agentIndex)
        if state.isWin() or state.isLose():
            return self.evaluationFunction(state)

        minEval = 9999999

        agentCount = state.getNumAgents()
        if agentIndex <= agentCount - 2:
            for action in legalActions:
                child = state.getNextState(agentIndex, action)
                minEval = min(minEval, self.minValue(depth, child, agentIndex + 1, a, b))
                if minEval < a:
                    return minEval
```

```
36                  b = min(b, minEval)
37          else:  # If there are no more ghosts after this one then it is pacman's (max) turn
38              for action in legalActions:  # find the min value of pacman actions after the ghost takes e
39                  child = state.getNextState(agentIndex, action)
40                  minEval = min(minEval, self.maxValue(depth + 1, child, a, b))
41                  if minEval < a:
42                      return minEval
43                  b = min(b, minEval)
44
45          return minEval
46
47      def getAction(self, gameState):
48          """
49          Returns the minimax action using self.depth and self.evaluationFunction
50          """
51          "*** YOUR CODE HERE ***"
52          bestActionValue = -9999999
53          bestAction = None
54          a = -9999999
55          b = 9999999
56          for action in gameState.getLegalActions(0):
57              childState = gameState.getNextState(0, action)
58              if self.minValue(1, childState, 1, a, b) > bestActionValue:
59                  bestActionValue = self.minValue(1, childState, 1, a, b)
60                  bestAction = action
61              if bestActionValue > b:
62                  return bestAction
63              a = max(a, bestActionValue)
64          return bestAction
```

Explanation:

- O mare parte din cod este la fel ca la algoritmul minimax, singura diferenta fiind cele doua variabile a si b.
- In functia maxValue (incepand cu randul 17) verificam daca valoarea maxima gasita este mai mare decat valoarea minima gasita in minValue, in acest caz returnam maxEval. In caz contrar, daca maxEval e mai mare ca a, a va lua valoarea lui si continuam cautarea.
- In functia minValue (incepand cu randul 22) verificam daca valoarea minima gasita este mai mica decat valoarea maxima gasita in maxValue, in acest caz returnam minValue. In caz contrar, daca minEval e mai mic decat b, b va luua valoarea lui si cautam cautarea.
- In functia geAction folosim a si b in acelasi mod ca in functia maxValue, doarece noi trebuie sa aflam valoarea maxima pe care o poate avea agentul nostru.

Commands:

- python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic

3.3.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

Q1: Test your implementation with autograder **python autograder.py** for Question 3. What are your results?

A1: 5/5

3.3.3 Personal observations and notes

3.4 References

4 Personal contribution

4.1 Question 11 - Define and solve your own problem.

In this section the solution for the following problem will be presented:

4.1.1 Code implementation

This sub-section is dedicated to showcasing your own solution that you came up with for solving the above question. One has to put here any **code** that has been used for solving the above task, along with **comments** that explain every design decision made. To reference the code, please make use of the *code lines number*. Additionally, complete this sub-section with any **command configurations** that you may have used during the implementation or testing process (please fill in *just the arguments*).

Code:

```
1
```

Explanation:

•

Commands:

•

4.1.2 Questions

This sub-section is dedicated to the additional questions that come along with the exercise. Please answer to the following questions:

4.1.3 Personal observations and notes

4.2 References