Artificial intelligence - Project 1
- Search problems -

Stecko Daiana, Timis Iulia, Viman Andrei

3/11/2021

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:

```python
def depthFirstSearch(problem):

    start = problem.getStartState()
    "nodul de start de unde pleaca dfs"
    curent = start
    "nodul in care ma aflu"
    noduriExplorate = []
    "noduri pe care deja le am vizitat"
    stiva = util.Stack()
    stiva.push((start,[]))
    "pun in stiva nodul de start"
    "cat timp stiva nu e goala si nu am ajuns la punctul final (mancare)"
    while not stiva.isEmpty() and not problem.isGoalState(curent):
        nod, directii= stiva.pop()
        noduriExplorate.append(nod)
        "marchez ca si vizitat nodul pe care urmeaza sa il explorez"
        succesori = problem.expand(nod)
        "ma uit la succesoruii nodului curent"
        for i in succesori:
            coordonataCurenta = i[0]
            "coordonata nodului (x,y)"
            if not coordonataCurenta in noduriExplorate:
                "daca coordonata nu este in nodurile explorate"
                curent = i[0]
                directie = i[1]
                "directia pe unde trebuie sa o iau"
                stiva.push((curent,directii+[directie]))
                "pun i stiva coordonata curenta si directia "
    return directii+[directie]
    util.raiseNotDefined()
```

Explanation:

- Algoritmul DFS este un algoritm de cautare neinformat, el cautand solutia in adancime, altfel spus el va explora fiecare nod chiar daca toti fratii parintelui nodului curent inca nu au fost descoperiti. Acest algoritm foloseste ca structura de date o stiva. Verificam daca fiecare vecin a fost sau nu expandat. Daca nodul nu a fost expandat il scoatem din stiva si il expandam, iar in caz contrar nu vom face nimic. Ca sa nu ajungem intr-o structura infinita avem nevoie si de un vector de noduri vizitate unde verificam daca nodul se afla sau nu. La final cand pacman a ajuns la o solutie vom returna un vector de directii de la punctul de start spre final (goalState).

Commands:

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

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: Algoritmul nu este unul optimal deoarece nu are o euristica dupa care sa se ghideze, el parcurgand nodurile in ordinea in care acestea sunt introduse in stiva, acest lucru facand ca gasirea drumului cel mai scurt sa fie putin probabil.

Q2: Run *autograder python autograder.py* and write the points for Question 1.
A2: Question q1: 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      """Search the shallowest nodes in the search tree first."""
3      start = problem.getStartState()
4      "nodul de start de unde pleaca bfs"
5      noduriExplorate = []
6      "noduri pe care deja le am vizitat"
7      noduriExplorate.append(start)
8      coada = util.Queue()
9      coada.push((start, []))
10     "pun in coada nodul de start"
11     "cat timp coada nu e goala "
12
```

```
    while not coada.isEmpty():
        nod, directii = coada.pop()
        "daca am ajuns la final returnam directiile"
        if problem.isGoalState(nod):
            return directii
        succesori = problem.expand(nod)
        "ma uit la succesoruii nodului curent"
        for i in succesori:
            coordonataCurenta = i[0]
            "coordonata nodului (x,y)"
            if not coordonataCurenta in noduriExplorate:
                "daca coordonata nu este in nodurile explorate"
                directie = i[1]
                "directia pe unde trebuie sa o iau"
                noduriExplorate.append(coordonataCurenta)
                coada.push((coordonataCurenta, directii + [directie]))
                "pun i stiva coordonata curenta si directia "
    return []
    util.raiseNotDefined()

```

Explanation:

- Algoritmul BFS este foarte similar cu DFS, diferenta majora fiind faptul ca la BFS folosim o coada in locul stivei, astfel parcurgerea se va face in latime, nu in adancime.

Commands:

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

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: Solutia gasita este optimala, datorita faptului ca prin aceasta parcurgere vom gasi drumul cel mai scurt de la Start la final. Acest algoritm nu este optimal din alt punct de vedere, acesta fiind numarul mare de noduri care sunt expandate pentru gasirea acestui traseu optim.

Q2: Run autograder *python autograder.py* and write the points for Question 2.
A2: Question q2: 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:

```
def uniformCostSearch(problem):

    "*** YOUR CODE HERE ***"
    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):

    startState = problem.getStartState()
    coadaDePrioritati = util.PriorityQueue()
    noduriExplorate = []
    coadaDePrioritati.push((startState, [], 0), 0)

    while not coadaDePrioritati.isEmpty():

        nod, directii, cost = coadaDePrioritati.pop()
        currentNode = (nod, cost)
        noduriExplorate.append(currentNode)

        if problem.isGoalState(nod):
            return directii

        succesori = problem.expand(nod)

        for i in succesori:
            noulCost = problem.getCostOfActionSequence(directii + [i[1]])
            noulNod = (i[0], directii + [i[1]], noulCost)

            vizitat = False

            for j in noduriExplorate:
                jState, jCost = j

                if (i[0] == jState) and (noulCost >= jCost):
                    vizitat = True

            if not vizitat:
                coadaDePrioritati.push(noulNod, noulCost + heuristic(i[0], problem))
                noduriExplorate.append((i[0], noulCost))


    return []
    util.raiseNotDefined()
```

Explanation:

- Algoritmul A* este cel mai cunoscut algoritm de căutare. Evaluarea nodului combină costul de căutare al nodului din starea inițială cu costul estimat de obținere de la nod la obiectiv. f(n) = g(n) + h(n) unde h(n) este euristica admisibilă, consistentă si garantează optimitatea lui A*

Commands:

- python3 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic python3 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=nullHeuristic

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: Nu am implementat UCS

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

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

Q4: Run autograder *python autograder.py* and write the points for Question 4 (min 3 points).
A4: Question q3: 4/4

Q5: Comparatie intre BFS, DFS si A*
A5: Diferența majoră dintre BFS și DFS constă în faptul că BFS avansează la nivel cu nivel, în timp ce DFS urmează mai întâi o cale de la începutul până la capătul nodului, apoi o altă cale de la început până la sfârșit și așa mai departe până când toate nodurile sunt vizitate. În plus, BFS folosește coada pentru stocarea nodurilor, în timp ce DFS folosește stiva pentru traversarea nodurilor. Algoritmul A* este un BFS optimizat, acesta folosind o coada de prioritati.Algoritmul de căutare A* vizitează următoarea stare pe baza eristicii f(n) = h + g unde componenta h este aceeași euristică aplicată ca în căutarea Best-first, dar componenta g este calea de la starea inițială la starea particulară. Prin urmare, nu alege următoarea stare doar cu cea mai mică valoare euristică, ci una care oferă cea mai mică valoare atunci când se ia în considerare euristica și costul de a ajunge la acea stare.

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:

```
1  class CornersProblem(search.SearchProblem):
2      """
3      This search problem finds paths through all four corners of a layout.
4  
5      You must select a suitable state space and child function
6      """
7  
8      def __init__(self, startingGameState):
9          """
10         Stores the walls, pacman's starting position and corners.
11         """
12         self.walls = startingGameState.getWalls()
13         self.startingPosition = startingGameState.getPacmanPosition()
14         top, right = self.walls.height-2, self.walls.width-2
15         self.corners = ((1,1), (1,top), (right, 1), (right, top))
16         for corner in self.corners:
17             if not startingGameState.hasFood(*corner):
18                 print('Warning: no food in corner ' + str(corner))
19         self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
20         # Please add any code here which you would like to use
21         # in initializing the problem
22         "*** YOUR CODE HERE ***"
23         self.right = right
24         self.top = top
25 
26     def getStartState(self):
27         """
28         Returns the start state (in your state space, not the full Pacman state
29         space)
30         """
31         "*** YOUR CODE HERE ***"
32         allCorners = (False, False, False, False)
33         return  (self.startingPosition, allCorners)
34         #util.raiseNotDefined()
35 
36     def isGoalState(self, state):
37         """
38         Returns whether this search state is a goal state of the problem.
39         """
40         "*** YOUR CODE HERE ***"
41         colturiVizitate = state[1]
42         return colturiVizitate[0] and colturiVizitate[1] and colturiVizitate[2]  and colturiVizitate[3]
43         #util.raiseNotDefined()
44 
45     def expand(self, state):
        """
        Returns child states, the actions they require, and a cost of 1.

         As noted in search.py:
            For a given state, this should return a list of triples, (child,
            action, stepCost), where 'child' is a child to the current
            state, 'action' is the action required to get there, and 'stepCost'
            is the incremental cost of expanding to that child
        """

        children = []
        for action in self.getActions(state):
            # Add a child state to the child list if the action is legal
            # You should call getActions, getActionCost, and getNextState.
            "*** YOUR CODE HERE ***"
            pozitie, colturiVizitate = state
            x,y = pozitie
            dx,dy = Actions.directionToVector(action)
            nextx, nexty = int(x + dx), int(y + dy)
            hitsWall = self.walls[nextx][nexty]
            nextState = (nextx, nexty)
            if not hitsWall:
                if nextState in self.corners:
                    if nextState == (self.right, 1):
                        newCorners = [True, colturiVizitate[1], colturiVizitate[2], colturiVizitate[3]]
                    elif nextState == (self.right, self.top):
                        newCorners = [colturiVizitate[0], True, colturiVizitate[2], colturiVizitate[3]]
                    elif nextState == (1, self.top):
                        newCorners = [colturiVizitate[0], colturiVizitate[1], True, colturiVizitate[3]]
                    elif nextState == (1, 1):
                        newCorners = [colturiVizitate[0], colturiVizitate[1], colturiVizitate[2], True]
                    successor = ((nextState, newCorners), action, 1)
                else:
                    successor = ((nextState, colturiVizitate), action, 1)
                children.append(successor)


        self._expanded += 1 # DO NOT CHANGE
        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)
        "*** YOUR CODE HERE ***"
        return (nextx,nexty)
        #util.raiseNotDefined()

    def getCostOfActionSequence(self, actions):
        """
        Returns the cost of a particular sequence of actions.  If those actions
        include an illegal move, return 999999.  This is implemented for you.
        """
        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:

- Pentru inceput am initializat sistemul, iar pentru metoda de getStartState am initializat toate colturile cu False pentru ca pacman inca nu a ajuns la ele. In metoda isGoalState am facut o simpla verificare, daca toate colturile sunt pe true, mai exact daca pacman a ajuns in toate colturile inseamna ca am ajuns la final. In functia de expand am verificat daca pacman a ajuns in unul dintre colturi, punem acel colt pe True, marcand faptul ca am vizitat acel colt.

Commands:

- python3 pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
  python3 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: Search nodes expanded: 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 
3     "*** YOUR CODE HERE ***"
4     return 0 # Default to trivial solution
```

Explanation:

-

Commands:

-

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:

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 
3     position, foodGrid = state
4     "*** YOUR CODE HERE ***"
5     return 0
```

Explanation:

-

Commands:

-

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:

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  class ReflexAgent(Agent):
2      """
3      A reflex agent chooses an action at each choice point by examining
4      its alternatives via a state evaluation function.
5
6      The code below is provided as a guide.  You are welcome to change
7      it in any way you see fit, so long as you don't touch our method
8      headers.
9      """
10
11
12     def getAction(self, gameState):
13         """
14         You do not need to change this method, but you're welcome to.
15
16         getAction chooses among the best options according to the evaluation function.
17
18         Just like in the previous project, getAction takes a GameState and returns
19         some Directions.X for some X in the set {NORTH, SOUTH, WEST, EAST, STOP}
20         """
21         # Collect legal moves and child states
22         legalMoves = gameState.getLegalActions()
23
24         # Choose one of the best actions
25         scores = [self.evaluationFunction(gameState, action) for action in legalMoves]
26         bestScore = max(scores)
27         bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
28         chosenIndex = random.choice(bestIndices) # Pick randomly among the best
29
30         "Add more of your code here if you want to"
31
32         return legalMoves[chosenIndex]
33
34     def evaluationFunction(self, currentGameState, action):
35         """
36         Design a better evaluation function here.
```

```

        The evaluation function takes in the current and proposed child
        GameStates (pacman.py) and returns a number, where higher numbers are better.

        The code below extracts some useful information from the state, like the
        remaining food (newFood) and Pacman position after moving (newPos).
        newScaredTimes holds the number of moves that each ghost will remain
        scared because of Pacman having eaten a power pellet.

        Print out these variables to see what you're getting, then combine them
        to create a masterful evaluation function.
        """
        # Useful information you can extract from a GameState (pacman.py)
        childGameState = currentGameState.getPacmanNextState(action)
        newPos = childGameState.getPacmanPosition()
        newFood = childGameState.getFood()
        newGhostStates = childGameState.getGhostStates()
        newScaredTimes = [ghostState.scaredTimer for ghostState in newGhostStates]

        "*** YOUR CODE HERE ***"
        listFood = newFood.asList() # All remaining food as list
        ghostPos = childGameState.getGhostPositions()  # Get the ghost position

        mFoodDist = []
        # cautam distanta intre toata mancarea si pacman
        for food in listFood:
            mFoodDist.append(manhattanDistance(food, newPos))

        mGhostDist = []
        # gasim distanta intre toate fantomele si pacman
        for ghost in ghostPos:
            mGhostDist.append(manhattanDistance(ghost, newPos))
        # incercam sa il facem sa se miste
        if currentGameState.getPacmanPosition() == newPos:
            return -10000
        # daca fantoma se apropie returna un numar foarte mic
        for ghostDistance in mGhostDist:
            if ghostDistance < 2:
                return -10000
        #daca nu mai avem mancare returnam un numar foarte mare
        if len(mFoodDist) == 0:
            return 10000

        return  1000 / sum(mFoodDist) + 1000 / len(mFoodDist) + childGameState.getScore()
```

Explanation:

- In metoda de evaluationFunction am transformant mancarea ramasa intr-o lista si am calculat distanta manhattan intre pacman si fiecare bucatica de mancare. Dupa care am calculat distanta manhattan intre pacman si fiecare fantoma iar mai apoi verificam ca fantomele sa nu fie aproape de pacman daca sunt aproape il forsam sa se indeparteze de ele, cu alte cuvinte returnam o distanta foarte mica. Daca nu mai avem mancare returnam o distanta foarte mare. In final daca nu ai intrat pe niciunul dintre if-uri vom compune o distanta care variaza in functie de mancarea ramsa si scorul curent.

Commands:

- python3 pacman.py -p ReflexAgent
  python3 pacman.py -p ReflexAgent -l testClassic

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: Din 10 teste consecutive agentul meu a casticat de fiecare data cu un scor mediu de 550 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:

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

    def getAction(self, gameState):
        """
        Returns the minimax action from the current gameState using self.depth
        and self.evaluationFunction.

        Here are some method calls that might be useful when implementing minimax.

        gameState.getLegalActions(agentIndex):
        Returns a list of legal actions for an agent
        agentIndex=0 means Pacman, ghosts are >= 1

        gameState.getNextState(agentIndex, action):
        Returns the child game state after an agent takes an action

        gameState.getNumAgents():
```

```
        Returns the total number of agents in the game

        gameState.isWin():
        Returns whether or not the game state is a winning state

        gameState.isLose():
        Returns whether or not the game state is a losing state
        """
        "*** YOUR CODE HERE ***"
        # Format of result = [score, action]
        # Return the action from result
        return self.get_value(gameState, 0, 0)[1]

    def get_value(self, gameState, index, depth):
        "Returneaza o tupla de compusa din scor si directie "

        "Verificam daca jocul a ajuns la final"
        if gameState.isWin() or gameState.isLose() or depth == self.depth:
            return self.evaluationFunction(gameState), ""
        if index > 0:
            "Un index mai mare ca 0 inseamna o fantoma"
            return self.min_value(gameState, index, depth)
        else:
            "Un index egal cu 0 inseamna ca e pacman"
            return self.max_value(gameState, index, depth)

    def max_value(self, gameState, index, depth):
        "Returneaza tupla cu valoarea maxima"
        maxv = -1000
        maxa = ""
        "miscarile legale care se pot face"
        legalMoves = gameState.getLegalActions(index)
        for a in legalMoves:
            successor = gameState.getNextState(index, a)
            indexS = index + 1
            depthS = depth
            "Daca este pacman actualizam adncimea si indexul"
            if indexS == gameState.getNumAgents():
                indexS = 0
                depthS += 1
            valCurenta = self.get_value(successor, indexS, depthS)
            if valCurenta[0] > maxv:
                maxv = valCurenta[0]
                maxa = a
        return maxv, maxa

    def min_value(self, gameState, index, depth):
        "Returneaza tupla cu valoarea minima"
        minv = 1000
        maxv = ""
        "miscarile legale care se pot face"
        legalMoves = gameState.getLegalActions(index)
        for a in legalMoves:
            successor = gameState.getNextState(index, a)
            indexS = index + 1
            depthS = depth
            "Daca este pacman actualizam adncimea si indexul"
            if indexS == gameState.getNumAgents():
                indexS = 0
                depthS += 1
            valCurenta = self.get_value(successor, indexS, depthS)
            if valCurenta[0] < minv:
                minv = valCurenta[0]
                maxv = a
        return minv, maxv
```

Explanation:

- Acest algoritm implementeaza o cautare adversariala foarte interesanta. Aceasta clasa contine mai multe functii, dintre care vreau sa le enumar pe cele mai importante: getAction, getvalue, maxvalue, minvalue. In metoda getAction se va apela metoda getvalue, cu starea jocului adancimea si indexul pacmanului. Metoda getvalue returneaza o tupla compusa din scor si directie, tot aici fiind aleasa valoarea de min si de max. Daca am ajuns la finalul jocului, altfel spus daca am castigat sau daca am pierdut sau daca adancimea curenta este egala cu adancimea cenerala atunci se va returna o tupla compusa sin scor si None. Daca indexul pasat in antetul functiei este 0, altfel spus daca este pacman se va apela functia maxim daca nu se apela functia de minim, deoarece fantomele au un index mai mare de 0. Mai departe vom explica functia maxvalue deoarece functia minvalue functioneaza in mod asemanator. Metoda maxvalue incearca sa gaseasca o valoare si niste mutari posibile maxime; le va cauta in actiunile care pot fi facute. Incearca sa gaseasca succesorul viitoarei mutari va creste indexul mutarii succesoare cu 1 si va seta aceeasi adancime. Daca agentul este pacman ii vom incrementa adancimea iar daca nu vom apela din nou functia getvalue cu succesorul mutarii indexul mutarii si adancimea. La intoarcerea din recursivitate trebuie sa verificam daca trebuie sa actualizam valoarea si actiunile maxime si le vom returna. Pentru functia de minvalue se intampla asemanator tot ce difera este faptul ca ea incearca sa gaseasca minimul nu maximul.

Commands:

- python3 pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
  python3 autograder.py -q q2
  python3 pacman.py -p MinimaxAgent -l trappedClassic -a depth=1
  python3 pacman.py -p MinimaxAgent -l trappedClassic -a depth=2
  python3 pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

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: Pentru aceasta comanda, la o adancime de 3 nu reuseste sa descopere fantoma din dreapta sa, adica pe cea rosie doar pe cea albastra, asa ca o va lua spre fantonma rosie. Daca punem o adancime de 2 descopera fantoma rosie si nu va merge spre ea ci va merge spre ce albastra. Sansa de castig este de cinzeci la suta in functie de directia pe care o va lua fantoma albastra. Daca o ia spre pacman va pierde in schimb daca o va lua in jos si spre dreapta pacman va castiga deoarece asfel reuseste sa manance toata mancarea.

3.2.3 Personal observations and notes

https://githubmemory.com/repo/khanhngg/CSC665-multi-agent-pacman

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:

```
1
```

Explanation:

•

Commands:

•

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:

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:

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