PICT, PUNE

31118

LP-II

Assignment-02

DOP: 21-Jan'22
DOS: 22-Jan'22

Title: A-star Algorithm

Problem Statement:

Implement Astar Algorithm for any game search problem.

Objectives

i) To learn about graph-search algorithm.
ii) To formulate & implement constraints in search problems

Software & Hardware Requirements:

Windows-10 OS (64-bit)
8-GB RAM & 512GB SSD
VS-code - latest version (Jan'22)
Python 3.8

Theory Related Concepts:

A* Algorithm:

i) It is one of the best & popular technique used in path-finding & graph traversals
ii) It is a derivation of Dijkstra's pathfinding algorithm, where search is made "smart" using addition heuristic function.
iii) Time complexity: the time complexity mainly depends on heuristic function used.
In worst case it can be O(E) where E is a number of edges in a graph.

iv) Auxiliary Space: In worst case, the open list can have all vertices hence O(V) where V is the number of vertices in graph.

v) There are mainly two types of heuristic functions used:

A) Exact Heuristic: inefficient, slow.

B) Approximation Heuristic: generally fast.

eg. a) Manhattan Distance

b) Diagonal Distance

c) Euclidean Distance

Pseudocode

1. Initialize the open list
2. Initialize the closed list
3. while open list is not empty:
   a) find the node with least f one the open list, call it "q"
   b) pop q off the open list
   c) generate q's successors & set their parents to q.
   d) for each successor
      i) if successor is goal, stop search
         successor.g = q.g + distance b/w successor & q
         successor.h = distance from goal to successor
         successor.f = successor.g + successor.h
      ii) if a node with same position as successor is in open list which has a lower f than successor, skip this successor

PICT, PUNE

iii) if the node with same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list.

end for loop.

e) push q on the closed list

end while loop.

8-Puzzle Problem:

- It consists of 8 tiles & one empty space where tiles can be moved.
- Start & goal configurations of the puzzle are provided. The puzzle can be solved by moving the tiles one by one in the single empty space & thus achieving the goal configuration
- eg

Initial State

1	2	3
8		4
7	6	5

Goal State

2	8	1
	4	3
7	6	5

Testcases

Puzzle	Expected	Actual	Results
Initial state: 1 2 3 - 4 6 7 5 8	the puzzle should be solved & moves used should be displayed	1 2 3 ~~4~~ ~~4~~ 6 7 ~~5~~ ~~8~~ 1 2 3 4 - 6 7 5 8	Pass.

PICT, PUNE

final state		1 2 3 4 5 6 7 - 8
1 2 3 4 5 6 7 8 -		1 2 3 4 5 6 7 8 -

Conclusion

In this assignment. I learned about A* path-finding algorithm in graph search technique. Also, implemented the same for solving classic 8-puzzle problem.

Code:

```
# Shubham - 311118

class Node:
    def __init__(self, data, level, fval) -> None:
        self.data = data
        self.level = level
        self.fval = fval

    def generate_child(self):
        x, y = self.find(self.data, '_')

        directions = [[x, y-1], [x, y+1], [x-1, y], [x+1, y]]
        childrens = []
        for i in directions:
            child = self.move_tile(self.data, x, y, i[0], i[1])
            if child is not None:
                child_node = Node(child, self.level+1, 0)
                childrens.append(child_node)

        return childrens

    def move_tile(self, puz, x1, y1, x2, y2):
        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 <
len(self.data):
            tmp_puz = []
            tmp_puz = self.copy(puz)
            tmp = tmp_puz[x2][y2]
            tmp_puz[x2][y2] = tmp_puz[x1][y1]
            tmp_puz[x1][y1] = tmp
            return tmp_puz
        else:
            return None

    # create cope of same node
    def copy(self, root):
        tmp = []
        for i in root:
            t = []
            for j in i:
                t.append(j)
            tmp.append(t)
        return tmp

    # find position of blank space
    def find(self, puz, x):
        for i in range(0, len(self.data)):
            for j in range(0, len(self.data)):
```

```
                if puz[i][j] == x:
                    return i, j


class Puzzle:
    def __init__(self, size) -> None:
        self.n = size
        self.open = []
        self.closed = []

    def get_input(self):
        puz = []
        for i in range(0, self.n):
            tmp = input().split(" ")
            puz.append(tmp)
        return puz

    def f(self, start, goal):
        return self.h(start.data, goal)+start.level

    def h(self, start, goal):
        tmp = 0
        for i in range(0, self.n):
            for j in range(0, self.n):
                if start[i][j] != goal[i][j] and start[i][j] != '_':
                    tmp += 1
        return tmp

    def process(self):
        print("Enter Start State:\n")
        start = self.get_input()
        print()
        print("Enter Goal State:\n")
        goal = self.get_input()
        print()

        start = Node(start, 0, 0)
        start.fval = self.f(start, goal)

        self.open.append(start)

        print("Solving..\n")

        while True:
            cur = self.open[0]
            for i in cur.data:
                for j in i:
                    print(j, end=" ")
                print()
```

```
            print()

            if self.h(cur.data, goal) == 0:
                break

            for i in cur.generate_child():
                i.fval = self.f(i, goal)
                self.open.append(i)

            self.closed.append(cur)
            del self.open[0]

            self.open.sort(key=lambda x: x.fval, reverse=False)

        print("\nFinished!")


puz = Puzzle(3)
puz.process()
```

Output:

```
Shubham@Shubham-AcerSwift MINGW64 /d/College-Stuff-6th-Sem/LPII (main)
$ py 31118_A02.py
Enter Start State:

1 2 3
_ 4 6
7 5 8

Enter Goal State:

1 2 3
4 5 6
7 8 _

Solving..

1 2 3
_ 4 6
7 5 8

1 2 3
4 _ 6
7 5 8

1 2 3
4 5 6
7 _ 8

1 2 3
4 5 6
7 8 _


Finished!

Shubham@Shubham-AcerSwift MINGW64 /d/College-Stuff-6th-Sem/LPII (main)
```