Assignment No-2.

Implement A* algorithm for any game search problem

= Aim:- To implement informed search strategy A* algorithm

= Theory:- Informed search strategies.

- To improve efficiency of search algorithms, problem specific information must be incorporated to solve huge issues with large number of alternative states.

= Heuristic evaluation function:-

- A heuristic evaluatn functn evaluates the cost of an optimum path between two states in single agent path finding problem.

= pure heuristic search:-

- It expands nodes in order of their heuristic values. It creates two lists. for the already expanded nodes & an open list for created but unexpanded nodes.
- In each iteration, a node with minimum heuristic value is expanded all its child nodes are created & placed in the closed list.
- The shorter paths are saved & longer ones are disposed of.

= A* search:- It is best-known form of BRST first search. It avoids expanding paths that are already expensive, but expands most promising path first.

= conclusion:- Thus studied A star search.

main.py

Run

```
1  from queue import PriorityQueue
2
3  class State(object):
4      def __init__(self, value, parent,
5                   start = 0,
6                   goal = 0):
7
8          self.children = []
9          self.parent = parent
10         self.value = value
11         self.dist = 0
12
13         if parent:
14             self.start  = parent.start
15             self.goal   = parent.goal
16             self.path   = parent.path[:]
17             self.path.append(value)
18         else:
19             self.path   = [value]
20             self.start  = start
21             self.goal   = goal
22
23     def GetDistance(self):
24         pass
25
```

main.py

Run

```
25
26      def CreateChildren(self):
27          pass
28
29  class State_String(State):
30      def __init__(self,value,parent,
31                   start = 0,
32                   goal = 0):
33
34          super(State_String, self).__init__(value, parent, start, goal)
35          self.dist = self.GetDistance()
36
37      def GetDistance(self):
38
39          if self.value == self.goal:
40              return 0
41          dist = 0
42          for i in range(len(self.goal)):
43              letter = self.goal[i]
44              try:
45                  dist += abs(i - self.value.index(letter))
46              except:
47                  dist += abs(i - self.value.find(letter))
48          return dist
49
```

main.py

Run

```
49
50      def CreateChildren(self):
51          if not self.children:
52              for i in range(len(self.goal)-1):
53                  val = self.value
54                  val = val[:i] + val[i+1] + val[i] + val[i+2:]
55                  child = State_String(val, self)
56                  self.children.append(child)
57
58  class AStar_Solver:
59      def __init__(self, start , goal):
60          self.path          = []
61          self.visitedQueue  = []
62          self.priorityQueue = PriorityQueue()
63          self.start         = start
64          self.goal          = goal
65
66      def Solve(self):
67          startState = State_String(self.start,
68                                    0,
69                                    self.start,
70                                    self.goal)
71
72          count = 0
73          self.priorityQueue.put((0,count,startState))
```

main.py

```
73          self.priorityQueue.put((0,count,startState))
74
75          while(not self.path and self.priorityQueue.qsize()):
76              closestChild = self.priorityQueue.get()[2]
77              closestChild.CreateChildren()
78              self.visitedQueue.append(closestChild.value)
79
80              for child in closestChild.children:
81                  if child.value not in self.visitedQueue:
82                      count +=1
83                      if not child.dist:
84                          self.path = child.path
85                          break
86                      self.priorityQueue.put((child.dist,count,child))
87
88          if not self.path:
89              print("Goal of %s is not possible!" % (self.goal))
90
91          return self.path
92
93
94  if __name__ == "__main__":
95      start1 = "emah"
96      goal1  = "ahem"
97      print("Starting...")
```

Waiting for securepubads.g.doubleclick.net...

```
85                                  break
86                              self.priorityQueue.put((child.dist,count,
87
88          if not self.path:
89              print("Goal of %s is not possible!" % (self.goal))
90
91          return self.path
92
93
94  if __name__ == "__main__":
95      start1 = "emah"
96      goal1  = "ahem"
97      print("Starting...")
98
99      a = AStar_Solver(start1, goal1)
100     a.Solve()
101
102     for i in range(len(a.path)):
103         print("{0}) {1}".format(i, a.path[i]))
104
```

35°C
Mostly cloudy