AI(2180703)

Tutorial-4

Name : Yogesh Bavishi
Enrollment No.: 170200107003
Division/Batch: E/E1

Q: Write a program to implement Single Player Game (Using Heuristic Function)

Code(pract4.py):

```
import sys, copy

goal = [['1', '2', '3'], ['4', '5', '6'], ['7', '8', ' ']]

class node:

    def __init__(self):
        self.heuristic = 0
        self.depth = 0

    def printPuzzle(self):
        print('')
        print (self.puzzleState[0][0], self.puzzleState[0][1], self.puz
zleState[0][2])
        print (self.puzzleState[1][0], self.puzzleState[1][1], self.puz
zleState[1][2])
        print (self.puzzleState[2][0], self.puzzleState[2][1], self.puz
zleState[2][2])

    def setPuzzle(self, puzzle):
        self.puzzleState = puzzle

def main():
    input = puzzleInput()
    algoChoice = "misplacedTile"
    puzzleSearch(input, algoChoice)

def puzzleInput():

    puzzle = []

    print("\nHere,8-Puzzle Problem is Solved using Heuristic Search Fun
ction.")
    print("\nHeuristic Value is calculated based on Number of Misplaced
 Tiles.")
```

```
    print("\nGoal State is  :  1 2 3\n\t\t  4 5 6\n\t\t  7 8 0")
    print ("\nEnter your puzzle, use a zero to represent the blank.\n")
    firstrow = input(("Enter the first row, use a space between numbers
 :"))

    firstrow = firstrow.split(' ')

    if (firstrow.count('0') == 1):
        firstrow[firstrow.index('0')] = ' '

    secondrow = input(("Enter the second row, use a space between numbe
rs :"))
    secondrow = secondrow.split(' ')
    if (secondrow.count('0') == 1):
        secondrow[secondrow.index('0')] = ' '

    thirdrow = input(("Enter the third row, use a space between numbers
 :"))
    thirdrow = thirdrow.split(' ')
    if (thirdrow.count('0') == 1):
        thirdrow[thirdrow.index('0')] = ' '

    puzzle.append(firstrow)
    puzzle.append(secondrow)
    puzzle.append(thirdrow)
    print ("\n")

    return puzzle

def expand(puzzle):

    expandList = []

    puzzleLeft = copy.deepcopy(puzzle)
    for x in puzzleLeft:
        if (x.count(' ') == 1):
            if (x.index(' ') != 0):
                spaceindex = x.index(' ')
                x[spaceindex] = x[spaceindex - 1]
                x[spaceindex - 1] = ' '

                expandList.append(puzzleLeft)

    puzzleRight = copy.deepcopy(puzzle)
    for x in puzzleRight:

        if (x.count(' ') == 1):
            if (x.index(' ') != 2):
                spaceindex = x.index(' ')
                x[spaceindex] = x[spaceindex + 1]
                x[spaceindex + 1] = ' '
```

```
                expandList.append(puzzleRight)

    puzzleUp = copy.deepcopy(puzzle)
    for x in puzzle:
        if (x.count(' ') == 1):
            if (x != puzzleUp[0]):
                spaceindex = x.index(' ')
                if(x == puzzle[1]):
                    puzzleUp[1][spaceindex] = puzzleUp[0][spaceindex]
                    puzzleUp[0][spaceindex] = ' '
                    expandList.append(puzzleUp)
                else:
                    puzzleUp[2][spaceindex] = puzzleUp[1][spaceindex]
                    puzzleUp[1][spaceindex] = ' '
                    expandList.append(puzzleUp)


    puzzleDown = copy.deepcopy(puzzle)
    for x in puzzle:
        if (x.count(' ') == 1):
            if (x != puzzle[2]):
                spaceindex = x.index(' ')
                if(x == puzzle[0]):
                    puzzleDown[0][spaceindex] = puzzleDown[1][spaceinde
x]
                    puzzleDown[1][spaceindex] = ' '
                    expandList.append(puzzleDown)
                else:
                    puzzleDown[1][spaceindex] = puzzleDown[2][spaceinde
x]
                    puzzleDown[2][spaceindex] = ' '
                    expandList.append(puzzleDown)

    return expandList

def checkGoal(puzzle):
    return goal == puzzle

def misplacedTiles(puzzle):

    misplace = 0
    for x in range(3):
        for y in range(3):
            if (puzzle[x][y] != ' '):
                if (puzzle[x][y] != goal[x][y]):
                    misplace += 1

    return misplace
```

```
def bubblesort(queue):

    for passesLeft in range(len(queue)-1, 0, -1):
        for index in range(passesLeft):
            if (queue[index].heuristic + queue[index].depth) > \
                   (queue[index + 1].heuristic + queue[index + 1].depth)
:
                queue[index], queue[index + 1] = queue[index + 1], queu
e[index]

    return queue

def puzzleSearch(puzzle, algorithm):

    nodesExpanded = 0
    maxQueueSize = 0
    queue = []

    puzzleNode = node()
    puzzleNode.setPuzzle(puzzle)
    puzzleNode.depth = 0
    puzzleNode.heuristic = misplacedTiles(puzzleNode.puzzleState)

    queue.append(puzzleNode)

    while 1:

        if (len(queue) == 0):
            print ("Puzzle search exhausted")
            sys.exit(0)

        checkNode = node()
        checkNode.puzzleState = queue[0].puzzleState
        checkNode.heuristic = queue[0].heuristic
        checkNode.depth = queue[0].depth

        print ('')
        print ("The best node to expand with g(n) =", checkNode.depth,
\
              "and h(n) =", checkNode.heuristic, "is...")
        checkNode.printPuzzle()
        print("Expanding this node...")

        queue.pop(0)

        if (checkGoal(checkNode.puzzleState)):
            print ('')
            print ("Solution found!!")
            checkNode.printPuzzle()
            print ('')
            print ("Expanded a total of", nodesExpanded, "nodes")
```

```
            print ("Maximum number of nodes in the queue was", maxQueue
Size)
            print ("The depth of the goal node was", checkNode.depth)
            return

        expandedPuzzle = expand(checkNode.puzzleState)

        for x in expandedPuzzle:
            tempNode = node()
            tempNode.setPuzzle(x)
            tempNode.heuristic = misplacedTiles(tempNode.puzzleState)
            tempNode.depth = checkNode.depth + 1
            queue.append(tempNode)
            nodesExpanded += 1

            if(len(queue) > maxQueueSize):
                maxQueueSize = len(queue)

        queue = bubblesort(queue)

if __name__ == "__main__":
    main()
```

Output:

TERMINAL ··· 1: cmd

```
D:\PROJECTS\AI>python pract4.py

Here,8-Puzzle Problem is Solved using Heuristic Search Function
.

Heuristic Value is calculated based on Number of Misplaced Tile
s.

Goal State is  :  1 2 3
                  4 5 6
                  7 8 0

Enter your puzzle, use a zero to represent the blank.

Enter the first row, use a space between numbers :1 2 3
Enter the second row, use a space between numbers :4 0 6
Enter the third row, use a space between numbers :7 5 8



The best node to expand with g(n) = 0 and h(n) = 2 is...

1 2 3
4   6
7 5 8
Expanding this node...

The best node to expand with g(n) = 1 and h(n) = 1 is...

1 2 3
4 5 6
7   8
Expanding this node...

The best node to expand with g(n) = 2 and h(n) = 0 is...

1 2 3
4 5 6
7 8
Expanding this node...

Solution found!!

1 2 3
4 5 6
7 8

Expanded a total of 7 nodes
Maximum number of nodes in the queue was 6
The depth of the goal node was 2

D:\PROJECTS\AI>
```