Problem formulation and Data Structure

Formulate goal: all tiles from 1 to 8 are in the correct location.

Formulate problem:

States: description specifies the location of each of the 8 tiles and the blank is one of the 9 squares

Actions: moving the blank tiles (up, down, left, right)

Search: A* algorithm using two heuristic functions.

Two classes:

Node Class: (declare the state configuration and generate successors from current state)

node attribute (data , level”depth”, fn)

DATA → matrix

Level → integer variable //g(n) path from start node to current node

Fn → integer variable // f(n) = g(n) +h(n)

Functions:

Generate successor(node)→ ***return*** *array of children of current node (successor[])*

// find the location (x,y) of blank tile in current node and then generate 4 new locations for the blank tiles and stored then in newloction[] array

Up → (x,y-1) Down → (x,y+1) Left → (x-1,y) Right → (x+1,y)

```
New_loc= [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]
```

//loop i in new_loc[ ] :

//generate children nodes by call swapping(node.date , x, y , i[0],i[1])with the current blank location

// check if the child is not a None value(means not out of the board) and then
// if yes, create a node object and then stored the child node in successor [ ]

//if not, go to the generate next child

swapping(node.date , x, y , i[0],i[1])
//check if the position is valid not out of the board and then swap

1	**2**	
4	**5**	**3**
7	**8**	**6**

1		**2**
4	**5**	**3**
7	**8**	**6**

1	**2**	**3**
4	**5**	
7	**8**	**6**

```
New_loc= [[2,-1],[2,1],[1,0],[3,0]]
```

Up X Down left right X

Grid class or puzzle: (accepts the initial and goal states from the user and provides function to calculate the **f(n)** , **h(n)** and **search ()**).

Grid attribute ( )

size → 3; //3 x 3 matrix

Explorted_list → [] array list to keep the explored node

Frontier list → [] array list that keep unexplored "successors" children

Functions:

read_input() →// read the states from the user return grid "matrix"

//for loop to accept the input

Evaluation_fn(current, goal) → return f(n) = g(n) + h(n)

// return heuristic(current.data , goal) + current.level;

heuristic(current , goal) → return h(n)

//this function will compute the misplace distance and Manhattan distance based on the programmer choice

// Misplace

```
// h= 0
        for i = 0 to 3
            for j = 0 to 3
                if current[i][j] != goal[i][j] and current[i][j] != '_':
                    h+= 1
        return h
```

//Manhattan

```
Search_fun( node ) {

//read initial and goal state  using read_input()

//start_state = read_input()      // goal_state = read_input()


//create a start (initial ) node object start_state= Node(start_state , 0,0)

//change the fn value of the node start_state.fn= Evaluation_fn(current, goal_state)

//add the start node to the unexplored list

//while(true)
{
current= frontier [0]

if ( heuristic( current , goal_state) == 0 )

Break // goal is found! ☺

// loop to generate the children and compute the evaluation_fn for all children

// add child to frontier list

// add the current node to the explored list

//sort the frontier based on the evaluation_fn
```

Create the puzzle object and call the search function :

```
grid = Grid();

grid.search();
```

```
 """ sort the opne list based on f value """

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