CS311 – Spring 2022
Introduction to Artificial Intelligence
HW Assignment #1

**Please see the accompanying video on the course Panopto site.**

**Due dates**:
Monday, March 7th by 11:15am (via the submission link on canvas)

***Solving 8-Puzzle via Search***

**Rules for group work**:

I suggest working in a GROUP on this assignment. You may work in groups of 2 students. **You will submit only ONE assignment for the group**. You may not "divide and conquer" this assignment. That is, all students must be present (virtually is ok) and contributing while any work on the project is being done. In other words, you must "pair-program" to complete the assignment. Along with the honor code, you must include, in a comment at the top of your assignment the following statement: "All group members were present and contributing during all work on this project." This statement will be treated in the same manner as the honor code, so please make sure it is present and true.

For this assignment, you will be solving the **8-Puzzle** using Breadth First Search and A* Search.

## **Part 1**: Breadth First Search

Much of the code is already written for you and is located here:

https://middlebury.instructure.com/courses/7984/files/folder/HWs/HW1

Node.java – Represents a **node** for the problem. Recall that each node corresponds to a possible **state** we may encounter as we search for the solution. A state for our problem is a configuration of the 3x3 board for the 8-Puzzle.

EightPlayer.java – This contains the **Intelligent Agent** for the 8-Puzzle. The program generates a random 8-Puzzle board and runs Breadth First Search to expand states and search for the goal state.

Requirements:
Fill in the code so that the program runs Breadth First Search to find a solution to the 8-Puzzle. The missing lines of code are indicated with a /*TO DO*/ comment. Remember that not all puzzles will have a solution. Since BFS is a **complete** search, your program should find a solution if one exists; otherwise it should indicate that a solution does not exist (see the Note below).

1. In `Node.java` do the following:

a. The code in the `expand()` method of Node.java generates the states that would result from expanding an initial state where the blank spot is in the upper left corner (i.e. `board[0][0]`). Complete the method by generating the states that would result from the other 8 possible initial positions of the blank. You will find the `moveBlank` methods useful (these are written for you).

2. In `EightPlayer.java` do the following:
   a. Fill in the `printSolution()` method so that each move from the initial state to the goal state is printed. You should print the board so that it is readable as a board, for example:

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

   You can use the print() method from the Node class to do this. You can use the `parent` node to keep track of the sequence of steps. Increment `nummoves` for every step required to get to the goal state.
   b. Write the Breadth First Search algorithm (in `runBFS()`). A bit of code (including the data structures you'll need) is provided for you. You can use the pseudocode we discussed in class (and on page 82) as a guide. Remember to increment `numnodes` whenever a new state is generated (i.e. inserted into Frontier).

   Note: Since the algorithm will generate **A LOT** of nodes, you should terminate it if it reaches a high depth (13) and have `runBFS()` return false, indicating that the solution doesn't exist (even if it really does).

   c. Run the program for 100 iterations and note the number of solutions, average number of moves, and the average number of nodes generated to solve (the provided code will report these values but your code must update `numnodes` and `nummoves` – updating `numsolutions` is done for you). You will report these values in a table (described below).

3. Modify only the methods described above. You are welcome to add more methods, global variables, import statements, etc. that will help you modify these methods but you should not have to make any changes to any other methods in the code – doing so will result in points deducted.

4. **Please comment your code to explain the steps of your program.** As with all assignments, part of your grade will depend on how well you comment your code.

Here is sample output:

```
single(0) or multiple boards(1)
0
BFS(0) or A* Manhattan Distance(1) or A* Improved Manhattan(2)
0
Enter board: ex. 012345678
123456078
```

```
 1 2 3
 4 5 6
   7 8

 1 2 3
 4 5 6
 7   8

 1 2 3
 4 5 6
 7 8

Number of nodes generated to solve: 7
Number of moves to solve: 2
Number of solutions so far: 1
_______

Number of iterations: 1
Average number of moves for 1 solutions: 2
Average number of nodes generated for 1 solutions: 7
```

**Don't worry if your program does not output the same number of nodes generated as there may be many correct answers since this will vary depending on the implementation.**

**Testing**:
There is a test case already in the code (it is commented out at the end of method `getNewBoard()`). You can use this test case and the following to test the correctness of your program. You can either copy and paste the test cases into the `getNewBoard()` method or use the interface to type in the board in string format. Note that the number of nodes generated is not given since this value will vary depending on the order in which your program expands new states.

```
//Case 2: 2 moves
//string format: 123456078
brd[0][0] = 1;
brd[0][1] = 2;
brd[0][2] = 3;

brd[1][0] = 4;
brd[1][1] = 5;
brd[1][2] = 6;

brd[2][0] = 0;
brd[2][1] = 7;
brd[2][2] = 8;


//Case 3: 3 moves
//string format: 123460758
brd[0][0] = 1;
brd[0][1] = 2;
brd[0][2] = 3;

brd[1][0] = 4;
brd[1][1] = 6;
```

```
brd[1][2] = 0;

brd[2][0] = 7;
brd[2][1] = 5;
brd[2][2] = 8;



//Case 4: 5 moves
//string format: 412053786
brd[0][0] = 4;
brd[0][1] = 1;
brd[0][2] = 2;

brd[1][0] = 0;
brd[1][1] = 5;
brd[1][2] = 3;

brd[2][0] = 7;
brd[2][1] = 8;
brd[2][2] = 6;

//Case 5: 8 moves
//string format: 412763058
brd[0][0] = 4;
brd[0][1] = 1;
brd[0][2] = 2;

brd[1][0] = 7;
brd[1][1] = 6;
brd[1][2] = 3;

brd[2][0] = 0;
brd[2][1] = 5;
brd[2][2] = 8;


//Case 6: 10 moves
//string format: 412763580
brd[0][0] = 4;
brd[0][1] = 1;
brd[0][2] = 2;

brd[1][0] = 7;
brd[1][1] = 6;
brd[1][2] = 3;

brd[2][0] = 5;
brd[2][1] = 8;
brd[2][2] = 0;

//Case 7: unsolvable
//string format: 134805726
brd[0][0] = 1;
brd[0][1] = 3;
brd[0][2] = 4;

brd[1][0] = 8;
brd[1][1] = 0;
brd[1][2] = 5;

brd[2][0] = 7;
brd[2][1] = 2;
brd[2][2] = 6;
```

## **Part 2**: A* Search

For this part of the assignment, you will be solving the **8-Puzzle** *intelligently* using **A* Search** with the **Manhattan Distance** heuristic.

You may find it useful to reuse some of the code from Part 1. You may find the additional methods (below the comment "----A* Code Starts Here----") in Node.java useful.

Here is rough pseudocode of the A* search algorithm that we discussed in class:

```
A*(initNode)
      1. Initialize a Priority Queue (Min-Binary Heap) to represent the
         Frontier. Initialize a separate Queue for the Explored nodes.
      2. Set the f(n)-value for initNode. Remember: f(n) = g(n) + h(n).
         For the initial node, g(n) = 0 and h(n) is the sum of the
         Manhattan Distances to reach the goal state (as we discussed
         in class). Add initNode into Frontier; increment numnodes.
      3. While the goal has not been found and Frontier is not empty
         and depth < 13 :
            o Remove from Frontier the node with the lowest f(n)-value.
              Denote this node X.
            o Add X to Explored
            o If X is the goal state, then we are Done!
            o Otherwise, expand X:
              For each child node, C, of X:
                   o If C is in Explored, ignore it.
                   o Otherwise, set the f(n)-value for C. For each C,
                     g(n) is the cost to get to C (this should be 1
                     more than the g(n)-value for X) and h(n) is the
                     sum of the Manhattan Distances to reach the goal
                     from C.
                   o Set the parent of C to be X. If C is not already
                     in Frontier, add it to Frontier; increment
                     numnodes. If C is already in Frontier, but with a
                     higher cost, then update the Frontier to include
                     C with the lower cost.
      4. If the goal has been reached, output the solution. If the
         Frontier is empty or depth >= 13, then output that no solution
         exists.
```

Requirements:

1. In addition to the sum of Manhattan Distances heuristic, you should implement another heuristic function. **Be creative here!** The goal is to out-smart your peers and devise a heuristic that will produce an optimal solution while minimizing the number of nodes generated and the number of moves. **Make sure your heuristic is admissible** so that it always produces an optimal solution. **The most efficient program(s) will receive extra points.**

2. In Node.java:

a. Fill in the code for the `evaluateHeuristic()` method to compute the Manhattan Distance heuristic.
b. Add the code necessary to implement your own heuristic.

3. In EightPlayer. java:
   a. Fill in the code for the `runAStar()` method so that it can use either the Manhattan Distance heuristic or your own new heuristic.
   b. In `getAlgChoice()`, replace <Your New Heuristic> with the name of your heuristic.

4. Modify only the methods described above. You are welcome to add more methods, global variables, import statements, etc. that will help you modify these methods but you should not have to make any changes to any other methods in the code – doing so will result in points deducted.

5. As in Part 1, **please comment your code to explain the steps of your program.** As with all assignments, part of your grade will depend on how well you comment your code.

6. Run both BFS and A* on the 7 sample cases given (once for each) and for the specified number of random boards/iterations, and fill in the following tables. For the average, you should count only boards for which your program found a solution. Note that the number of iterations for A* is 1000 while for BFS, it is only 100. This is because running BFS for 1000 iterations will take a really long time since many of the random boards generated may be unsolvable. ***Running BFS for just 100 random boards may also take a long time – possibly even a few hours, so be prepared for this and plan accordingly!!*** Include this table in your ReadMe. Note that the number of nodes generated by your program may differ based on implementation, so don't worry if you get a different number than another group. The optimal number of moves, however, should be the same.

| **Case** | **Number of moves** | **Number of nodes generated** | | |
|---|---|---|---|---|
| | | **BFS** | **A* (Manhattan distance)** | **A* (your heuristic)** |
| 1 | | | | |
| 2 | | | | |
| … | | | | |
| 7 | | | | |
| Average for all iterations | N/A (should be the same) | | | |

**<u>Output</u>**:

We would like to grade your programs as efficiently as possible. So please make sure your program does not output anything other than what is asked for above. Points will be deducted for extraneous output/debugging statements.

**Submission**:

Please include, in a comment at the top of every file you submit, the following:

- Full names of students in the group
- Group honor code: "All group members were present and contributing during all work on this project"
- Middlebury honor code

Zip the following files and use the HW1 link on canvas to submit exactly **ONE** zipped file per group.

- EightPlayer.java
- Node.java
- ReadMe

Name your zip file as follows: HW1_<last names of all group members>. So, for example, if your group consists of the last names Alvarez and Baker, you should name your file **HW1_AlvarezBaker.**

**Breakdown of Points**:

[90] - Correctness of implementation of above methods
    [40] Part 1
    [50] Part 2
[10] - Comments in code/ReadMe
[+5-10] – Extra credit for best heuristic

**ReadMe**:
As always, please provide a ReadMe that includes the following:

1. The names of the (2) students in your group.
2. Name of files required to run your program.
3. Any known bugs in your program (you will be less penalized for bugs that you identify than for bugs that we identify).
4. The filled-in table described above.
5. A brief description of your new A* heuristic.

## **Part 3**: Written

The following problems are from the course reference text (available from our canvas page):
1) [14] 3.3 (Parts (a) – (d); for Part (a) describe the **state space**, **initial state**, **goal state**, and **step cost function, i.e. transition model**).
2) [20] 3.14: **If false, give an example**. Do parts a, b, d, and e, only (**not part c**).

Also:
3) [10] Consider a state space where the start state is number 1 and each state *k* has two successors: numbers *2k* and *2k*+1.

a) Draw the portion of the state space for states 1 to 15.

b) Suppose the goal state is 11. List the order in which nodes will be visited for (i) Breadth-First Search, (ii) Depth-First Search, and (iii) Iterative Deepening.

4) [20] For the graph in Figure 3.2 (page 68) and the table in Figure 3.22 (page 93), show the steps of applying A* search to get from Zerind to Bucharest. Specifically, for each step, show the node that gets expanded and the contents of the Frontier (as we did in class). Also give the optimal path.