CSC 680: Artificial Intelligence
Fall 2012
Assignment #1 (30 Points)
Due date: Wednesday 9/19/2012 (in class)

Objective: To gain programming experience using an uninformed search method.

The 15-puzzle is a sliding puzzle that consists of a grid of numbered squares with one square missing, and the labels on the squares jumbled up. The goal of the puzzle is to unjumble the squares by only making moves which slide squares into the empty space, in turn revealing another empty space in the position of the moved piece.

Programming Guidelines

1. You may assume the initial and the desired configurations of the puzzle can be represented as 1D or 2D arrays (0 indicates the missing square):

   Initial State = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0}
   Goal State = {2,3,4,0,1,6,7,8,5,10,11,12,9,13,14,15}

   Or,

   Initial State = {{1,2,3,4}, {5,6,7,8}, {9,10,11,12}, {13,14,15,0}}
   Goal State = {{2,3,4,0}, {1,6,7,8}, {5,10,11,12}, {9,13,14,15}}

2. Write an iterative algorithm (using **stacks**) called *solvePuzzle*(*initial_state, goal_state*) that uses **iterative deepening search (IDS)** to solve the puzzle.

3. Develop methods **left**(), **right**(), **up**() and **down**() which move the empty square one position to the left, right, up, and down, respectively.

Input/Output Guidelines

1. If the puzzle can be solved, your program should print the series of moves (e.g. LDDRULDR…) that transform the initial state to the goal state; otherwise, an appropriate error message should be displayed on the console.

2. I will use several test cases so your code should accept its input from an external input file called **test.dat** which contains only two lines: the first line is the input state and the second line the goal state

```
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  0
2  3  4  0  1  6  7  8  5  10  11  12  9  13  14  15
```

3. I expect to be able to run your code from the console passing the input file as a command line argument as follows:

   ```
   C:\java Puzzle test.dat
   ```

   where **Puzzle.java** is the name of your program and **test.dat** is the input file.

4. If you do not have JDK on your PC, you may install JDK1.7 using my installation notes.

Grading Guidelines

- I do not accept multiple submissions
- Read my policy on late submissions

Technical Correctness • Program compiles and executes correctly • Proper error checking is performed on input file and during execution • Assignment objectives are met • Specified I/O guidelines are followed	**90%**
Coding Style & Documentation • Opening comments (author name, assignment number, date, purpose, etc.) are included • General comments appear in code • Code is properly indented and spaced • Meaningful variable names are used • Code is properly modularized	**10%**

Coding Style Guidelines

1. Include the following opening comments in the beginning of each program you submit:

```
/* ********************************************************************************
```

```
Author's name(s):
Course Title:
Semester:
Assignment Number
Submission Date:
Purpose: This program …
Input:
Output:
Help: Acknowledge any help you might have received or simply indicate that you worked alone.
**************************************************************************  */
```

2. Recall from our in-class discussion that 10% of your grade will be based on how well you develop and present your work:

   a. Use suitable names for Java classes, variables, and constants.

   b. Provide appropriate and non-trivial comments for your well-indented code.

   c. Every method must have an opening **Purpose** comment with **Input** and **Output**, when appropriate.

```
//*******************************************************
//*** Purpose: This method adds a node to the head of a given list
//*** Input:     A DLL node
//*** Output:   None
//*******************************************************
public static void addFirst(DLL L){
…
}
```

   d. Modularize your programs.

3. The importance and benefits of a *consistent coding style* are well known. A consistent style:

   a. Improves the readability, and therefore, maintainability of code.

   b. Facilitates sharing of code among programmers, especially teams of programmers working on the same project.

   c. Allows easier development of automated tools to assist in program development, such as tools which automatically format or pretty-print source code.

   d. Makes it easier to conduct code reviews, another software engineering process with well-known benefits. In turn, a practice of regular code reviews can help enforce a consistent style.

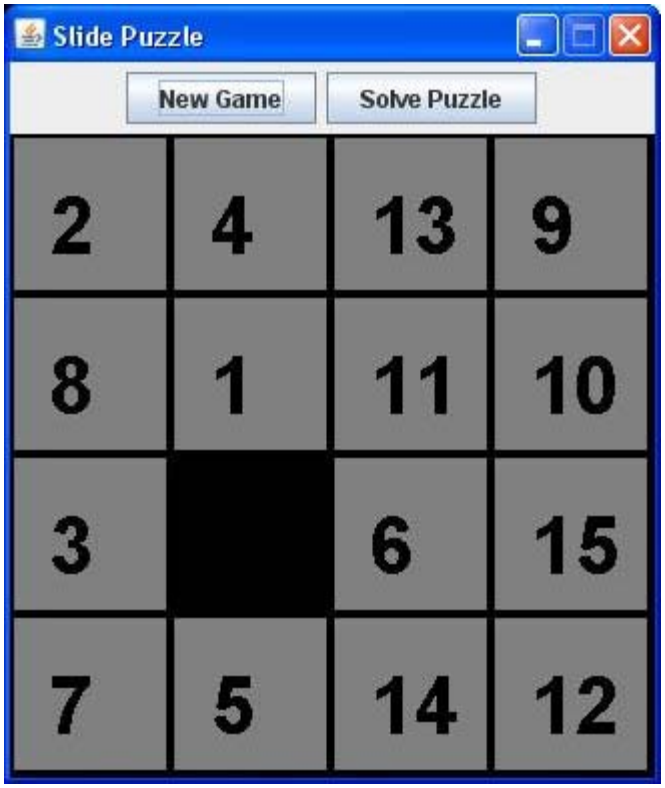
Extra Credit

- For those of you who are always looking for a challenge, I am including here a GUI interface written by Fred Swartz which utilizes three Java files but you would only need to modify the **setState** method in **SlidePuzzleModel.java** to create a GUI environment for **your own 15-puzzle solver**.

- To run the application, open a command window in DOS and type:

  C:\ javac *.java
  C:\ java SlidePuzzle

- Click the **New Game** button to get a random initial state



- Click the **Solve Puzzle** button to visually see how the original configuration is achieved. It is a good idea to print the moves in the background window (e.g. LLDDRU…) or an appropriate error message in case the puzzle is not solvable.

Submission Guidelines

1. Place your program files (**Puzzle.java**, etc.) and the supporting data files (if any) in a directory called **YourName-HW1** and archive it using Winzip or WinRAR. Include any special instructions to run your code in a **README** file.

2. Email me the zipped archive on or before the due date using the following subject line for your email message:

   Your Name – AI – HW1

   In addition, submit a **printout of all your program files** in class on the due date.