Lab 2, Artificial Intelligence, spring 2017

Table of Contents

Introduction

This is the second lab of a series of labs that will result to a program in which you can play and also automatic solve an n-puzzle, where n can be 3, 8 or 15. You have an example of such a puzzle at http://mypuzzle.org/sliding. This is a classical problem used in courses about AI. In your textbook (Luger), an 8-puzzle is used as an example. Use the textbook for support and help in your work.

You have to do your coding in Java and with help of a project in NetBeans IDE. You are not allowed to use any third-party software libraries; you are limited to use the Java SE.

You shall not hand in the parts of your solution that will be the result from each separate lab. You only have to hand in your final solution that you should have after the last lab. The hand in shall consist of the project directory that is created in NetBeans plus a short report where you explain your solution.

Although there are many solutions on the Internet you have to construct your own program that consists solely of your own written code, with the exception of the code skeleton in the section First Aid. You are allowed to use the complete skeleton or parts from it. You may not copy someone else's solution or parts of it and you may not allow someone else to copy your solution or parts of it. The fact that you send in a solution also confirms that you have read and understand the rules that apply to cheating and plagiarism, see http://www.du.se/en/Library/Academic-Writing/Copyright-and-Plagiarism/ and its sub sites. Your solution will be checked against other solutions with help of MOSS. MOSS is a system specially designed for checking program code for plagiarism. You can read more about MOSS on the following website, http://theory.stanford.edu/~aiken/moss/ .

The application

In this lab, you should add to your program a solver part that search for solutions of an n-puzzle. When you finish the lab you program shall has a facility for automatically search a solution of an n-puzzle. In the next lab you will improve the solver with heuristic search for solutions.

When you finish this lab you shall have a program that

- Execute in the command prompt
- Have a menu-system with at least following options:
  - Quit the program.
  - Create a new n-puzzle of a selectable size of n.
  - Mix the tiles by automatic move around them.
  - Manually solve the puzzle by move tile after tile.
  - Read in a state for an n-puzzle and assign this state as the start-state or goal state for automatically solving the puzzle
  - It shall be possible to select the type of search, breath or depth search. If the user select depth search, it must also be possible to limit the depth of the search.
- The program shall consist of at least classes that represent; the menu system, the solver, breadth search searching, depth search searching and the the puzzle.

The breadth search and depth search shall be implemented according to the algorithms in your textbook, chapter three (Luger). When you solve the textbooks examples of the 8-puzzle, your program shall give the same output as in chapter three.

A proposal of a work plan

1. Read through the algorithm for breadth first search, see figure 1.

```
Function breadth_first-search;

begin
   open := [Start];
   closed := [ ];
   while open != [ ] do
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed;
                  put remaining children on right end of open
               end
      end
   return FAIL
end
```

left side - only thing we have to change

Figure 1: An algorithm for breath first search, see chapter three in your textbook, (Luger).

Tiefe

You get the depth first algorithm if you change the statement of putting the remaining children to the left end of open list, instead of putting them to the right end.

As you can see; to get a breadth first search you have to add the states to the open list in one end and take out the states from the open list in the other end. The open list is working as a queue. To get depth first you have to add the states and remove the states from the same end of the open list. The open list is working as a stack.

2. Create a separate small program and checkup how you can use the Stack, Queue and LinkedList that is in the package java.util. To create a queue you have to use the interface Queue together with the class LinkedList, like this.

- Boxing
- superclase
- Interface class

google

```
   Queue<Double> myQueue = new LinkedList<Double>();
```

Try also to get the same behavior as for a queue and a stack by only using a LinkedList.

You can read about:
Queue at, https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html
Stack at, https://docs.oracle.com/javase/8/docs/api/java/util/Stack.html
LinkedList at, https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html

3. Read through the section First Aid in this lab. Decide if you shall continue with the program you constructed in the first lab an add the new functionality to that program, or if you shall use the skeleton that is given for lab 2 and copy in the part that you did in lab 1 to that skeleton, or if you only shall use some ideas from the First Aid and work out a the rest of the solution by your own.

4. Implement the tree of letters that is used as an example in chapter three of the textbook, see figure 2.

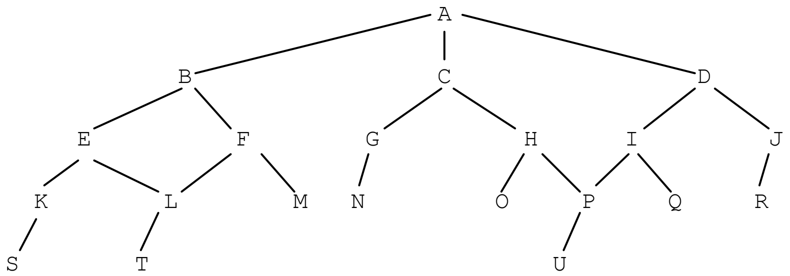

Figure 2: A tree of letters for testing the search algorithms, see chapter three in your textbook, (Luger).

With help of that tree you can test and debug your implementations of the search algorithms in a convenient way.

5. Implement the solver and the two search algorithms breadth first and depth first, and check if they give correct results for the tree of letters.

6. Update the class for the puzzle so it becomes possible to solve puzzles with help of the both the algorithms.

First aid!

If you have troubles to find a useful design of your program, maybe this section can be helpful.

Figure 3 shows a class-diagram over simple solution of the application you are going to construct.

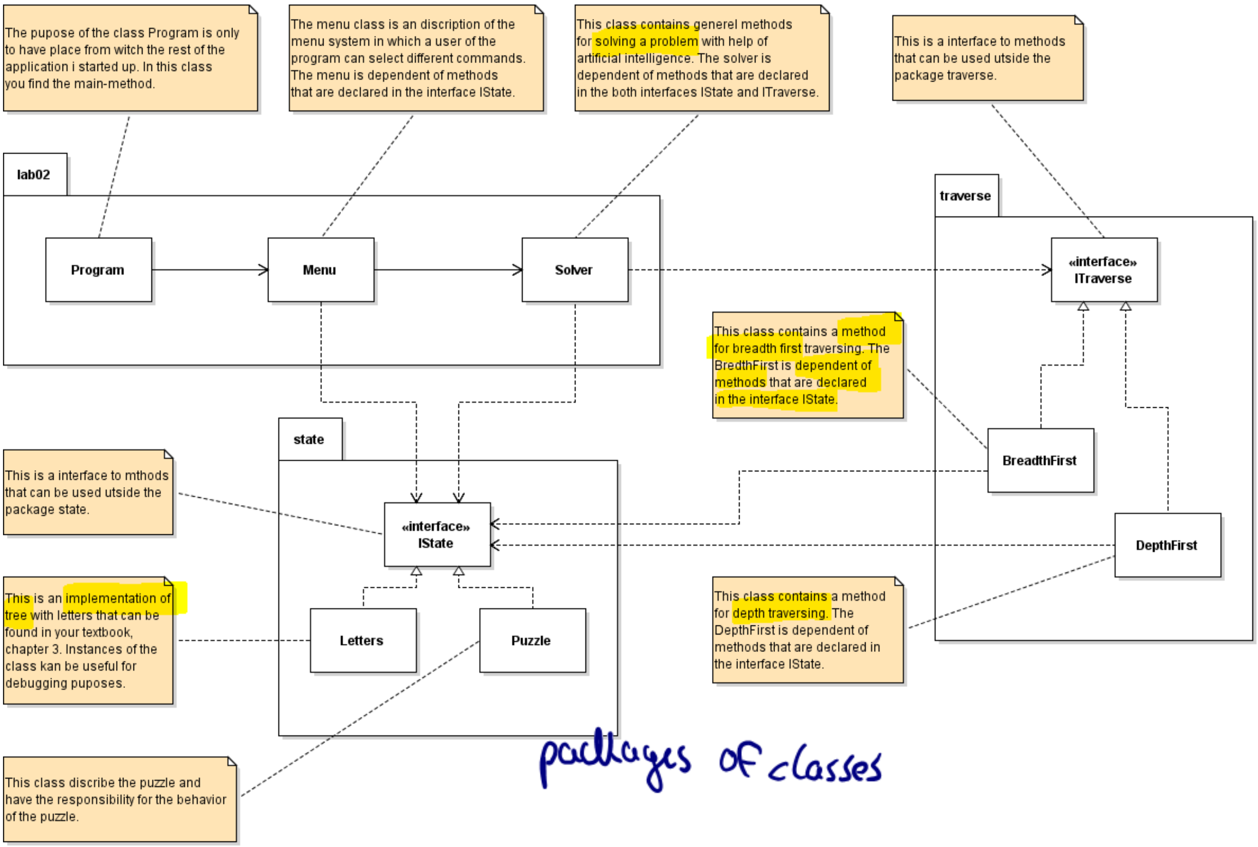
The pupose of the class Program is only to have place from witch the rest of the application i started up. In this class you find the main-method.
The menu class is an discription of the menu system in which a user of the program can select different commands. The menu is dependent of methods that are declared in the interface IState.
This class contains generel methods for solving a problem with help of artificial intelligence. The solver is dependent of methods that are declared in the both interfaces IState and ITraverse.
This is a interface to methods that can be used utside the package traverse.
lab02
Program
Menu
Solver
traverse
«interface» ITraverse
This class contains a method for breadth first traversing. The BredthFirst is dependent of methods that are declared in the interface IState.
BreadthFirst
DepthFirst
state
This is a interface to mthods that can be used utside the package state.
«interface» IState
This is an implementation of tree with letters that can be found in your textbook, chapter 3. Instances of the class kan be useful for debugging puposes.
Letters
Puzzle
This class contains a method for depth traversing. The DepthFirst is dependent of methods that are declared in the interface IState.
This class discribe the puzzle and have the responsibility for the behavior of the puzzle.
packages of classes

Figure 3: A class diagram that show a design of the application.

As you can see in figure 3 the design consists of three different packages, lab02, state and traverse. The classes Menu and Solver in the package lab02 are dependent of the interfaces of the other two packages. As Menu and Solver can work with different type of states, tree of letters or puzzle, they can also use different types of algorithms for searching solutions, breadth first or depth first. The flexibility is a result of the both classes Menu and Solver only access methods in the other packages by the packages interfaces.

You will find a proposal of a code skeleton for the above-described design in the file archive lab02.zip. If you want, you can download this code skeleton and continue to develop it to a solution for lab 2. The content of the file lab02.zip is a project directory that you can open up and continue to work with in NetBeans.

It is very easy to rename packages, classes, methods and variables in NetBeans. Highlight the name you want to change, right click and select refactor. In that way you can change the name of the package that you created in the previous lab. It is possible to move a class from one package to another by dragging the file of the class to the package that you want the class belong to.

Literature

Luger G. F., (2009) *Artificial intelligence Structures and Strategies for Complex Problem Solving*, sixth edition, Pearson Education Inc., ISBN-13: 978-0-13-209001-8.