Planning and Approximate Reasoning

Practical Exercise 2: Planner

- The deadline for the delivery of this exercise is **January 10th, 2016**
- There is a second submission date set to **January 31st, 2016**, with a maximum grade of **7**.
- A zip file with the source code in Java must be sent. Preferably a file exported from Eclipse. If not, include the necessary instructions/ executable files to run the program (prepare a xx.bat file if possible).
- A detailed documentation in PDF is required (see details below)
- The submission of the source and documentation must be done using Moodle.
- This exercise must be solved in groups of two people.

Consider the following scenario:

There is a square building composed by 9 offices, which are located in a matrix of 3 rows and 3 columns. From each office it is possible to move (horizontally or vertically) to the adjacent offices. Each of the offices may be clean or dirty. There is an automated cleaning robot that can move from one office to another.

Each office may be empty or it may contain one box (there can't be more than one box in the same office). The number of boxes in the building may be between 1 and 8. The robot can push a box from one office to an adjacent office. An office can be cleaned only if it is empty.

The robot will start with a given initial configuration (different for each test), with some clean and dirty offices, and with a box in some rooms. In the goal state all rooms will be clean, and the final position of the boxes will be explicitly indicated.

In this practical exercise you have to design and implement (in Java) a **linear planner with a stack of goals** that can discover how to go efficiently from an initial state of the world to a target state.

The operators to be considered are the following:

- ***Clean-office(o)***: the robot cleans office o
  - Preconditions: robot-location(o), dirty(o), empty(o)
  - Add: clean(o)
  - Delete: dirty(o)
- ***Move(o1,o2)***: the robot moves from o1 to o2
  - Preconditions: robot-location(o1), adjacent(o1,o2)
  - Add: robot-location(o2)
  - Delete: robot-location(o1)
- ***Push(b,o1,o2)***: the robot pushes box b from o1 to o2
  - Preconditions: robot-location(o1), box-location(b,o1), adjacent(o1,o2), empty(o2)
  - Add: box-location(b,o2), robot-location(o2), empty(o1)
  - Delete: empty(o2), box-location(b,o1), robot-location(o1)

The predicates to be considered are the following:

- Robot-location(o): the robot is in office o.
- Box-location(b,o): box b is located in office o.
- Dirty(o): office o is dirty.
- Clean(o): office o is clean.
- Empty(o): there isn't any box in office o.
- Adjacent(o1,o2): offices o1 and o2 are horizontally or vertically adjacent (fixed information).

Example:

Initial state (grey offices are dirty):

o1	o2	o3
o4 Robot	o5 Box A	o6 Box B
o7	o8	o9

Final state:

o1	o2	o3
o4	o5	o6 Box B
o7	o8 Box A	o9 Robot

A possible solution could be:
Move(o4, o1), Clean-office(o1), Move(o1,o2), Move(o2, o5), Move (o5,o8), Clean-office(o8), Move(o8,o5), Push(b1,o5,o8), Move(o8,o5), Clean-office(o5), Move(o5,o6), Move(o6,o9), Clean-office(o9).

You have to:

- Implement the planner in Java to solve the problem, indicating the internal representation used to manage the preconditions, to check the applicability of the operators, etc. (*in case you don't know the Java language, contact the teacher to agree about other programming language*).
- Design some intelligent tactics (i.e. heuristics) to aid the planning process when stacking preconditions, applying operators, etc.
- Your program must be able to read from a text file the initial and final states of the world in order to represent them internally. Testing files used to evaluate the software will have the following format (for the example case):

```
Boxes=A,B
Offices=o1,o2,o3,o4,o5,o6,o7,o8,o9
InitialState=Dirty(o1);Clean(o2);Clean(o3);Clean(o4);
Dirty(05);Clean(o6);Clean(07); Dirty(o8);Dirty(o9);
Robot-location(o4);Box-location(A,o5);Box-location(B,o6);
Empty(o1);Empty(o2);Empty(o3);Empty(o4);Empty(o7);Empty(o8);
GoalState= Robot-location(o9); Box-location(A,o8);
Box-location(B,o6);Empty(o1);Empty(o2);Empty(o3);
Empty(o4);Empty(o5);Empty(o7);
```

You can see that the Dirty(x) predicate has not been included in the goal state, they are implicit, always the offices will be clean at the end of the task. If you prefer to have this predicate, you can add it. You can also modify the order in which the predicates are written in the initial and goal states.

- The algorithm execution output must be clearly printed out in a log text file. This file has to clearly display the states that are being generated and evaluated, and the contents of the goal stack.
- Test your code with a set of testing cases of increasing complexity (a minimum of 4). Discuss in the document the solutions your program found for these examples, and whether there may exist, or not, more optimal plans.

Documentation content:

1. Introduction to the problem
2. Analysis of the problem (search space, operators, pre-conditions, special situations, etc.)
3. Planning algorithm (explaining the domain knowledge used to define the strategies or heuristics in the different steps of the method).
4. Implementation design (class diagram and details of the methods you consider more relevant).
5. Testing cases and results (show the important steps to arrive to the solution, not only the final path). Analysis of the results (complexity, number of steps, etc.).
6. Instructions to execute the program.

Evaluation criteria:

25% analysis, 25% strategies for improving the planner, 25% implementation, 25% execution