Department of Computer Science
Technical University of Cluj-Napoca

Introduction to Artificial Intelligence

Laboratory activity

Name:Prata Dragos-Liviu
Group:30235
Email:pratadragos96@yahoo.com

Assoc. Prof. dr. eng. Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

Rules and policies

Lab organisation.

1. Laboratory work is 20% from the final grade.
2. There are 3 deliverables in total.
3. Before each deadline, you have to send your work (latex documentation/code) at moodle.cs.utcluj.ro

Class:	Introducere in Inteligenta Artificiala
Enrollment key:	Iia2017-2018

4. *Laptop policy*: you can use your own laptop as long you have Linux. One goal of the laboratory is to increase your competency in Linux. It is **your** task to set static IPs:

	IP:	192.168.1.51[1]
	MASK:	255.255.255.0
	GATEWAY:	192.168.1.2
	DNS:	192.168.1.2
	PROXY	192.168.1.2:3128
Wifi:	Network:	isg
	Password:	inteligentaartificiala

5. *Group change policy.* Maximum number of students in a class is 14.
6. *For students repeating the class*: A discussion for validating the previous grade is mandatory in the first week. I usually have no problem to validate your previous grades, as long you request this in the first week. Failing to do so, leads to the grade 1 for the laboratory work in the current semester.

Grading. Assessment aims to measure your knowledge and skills needed to function in realistic AI-related tasks. Assessment is based on your written report explaining the nature of the project, findings, and recommendations. Meeting the deadlines is also important. Your report is comparable to ones you would write if you were a consultant reporting to a client.

Grade inflation makes difficult to distinguish between students. It also discourages the best students to do their best. In my quest for "optimal ranking of the students", I do not use the following heuristics:

Table 1.1: Lab scheduling.

Activity	**Deadline**
Searching agents, linux, latex, python	W_1
Uninformed search	W_2
Informed Search	W_3
Adversial search	W_4
Propositional logic	W_5
First order logic	W_6
Inference in first order logic	W_7
Knowledge representation in first order logic	W_8
Classical planning	W_9
Contingent, conformant and probabilistic planning	W_{10}
Multi-agent planing	W_{11}
Modelling planning domains	W_{12}
Individual feedback to clarify the good/bad issues related to student activity/results during the semester.	W_{14}

- "He worked hard at the project". Our society do not like anymore individuals that are *trying*, but individual that *do* stuff. Such heuristic is not admissible in education, except the primary school.
- "I knew he could do much better". Such a heuristic is not admissible because it does not encourage you to spread yourself.
- 7 means that you: i) constantly worked during classes, ii) you proved competent to use the tool and its expressivity for a realistic scenario, iii) you understood theoretical concepts on which the tool rely on.
- 8, 9 mean that your code is large enough and the results proved by your experiments are significant.
- 10 means that you did very impressive work or more efficient that I expected or handled a lot of special cases for realistic scenarios.
- 5 means that you managed to develop something of your own, functional, with your own piece of code substantially different from the examples available.
- You obtain less than 5 in one of the following situations:

  1. few code written by yourself.
  2. too much similarity with the provided examples.
  3. non-seriosity (i.e. re-current late at classes, playing games, worked for other disciplines, poor/unprofessional documentation of your work, etc.)[2].

- You get 2 if you present the project but fail to submit the documentation or code. You get 1 if you do not present your project before the deadline. You get 0 for any line of code

[2]Consider non-seriosity as a immutable boolean value that is unconsciously activated in my brain when one of the above conditions occurs for the first time.

taken from other parts that appear in section *My own code.* For information on TUCN's regulations on plagiarism do consult the active norms.

If your grade is 0, 1, or 2, you do not satisfy the preconditions for participating to the written exam. The only possibility to increase your laboratory grade is to take another project in the next year, at the same class, and to make all the steps again.

However, don't forget that focus is on learning, not on grading.

Using Latex in your documentation. You have to show some competency on writing documentation in Latex. For instance, you have to employ various latex elements: lists, citations, footnotes, verbatim, maths, code, etc.

Plagiarism. Most of you consider plagiarism only a minor form of cheating. This is far from accurate. Plagiarism is passing off the work of others as your own to gain unfair advantage.

During your project presentation and documentation, I must not be left with doubts on which parts of your project are your work or not. Always identify both: 1) who you worked with and 2) where you got your part of the code or solution. You should sign the declaration of originality.

Describe clearly the starting point of your solution. List explicitly any code re-used in your project. List explicitly any help (including debugging help, design discussions) provided by others (including colleagues or teaching assistant). Keep in mind that it is your own project and not the teaching assistant's project. Learning by collaborating does remain an effective method. You can use it, but don't forget to mention any kind of support. Learning by exploiting various knowledge-bases developed by your elder colleagues remain also an effective method for "learning by example". When comparing samples of good and poor assignments submitted by your colleagues in earlier years try to identify which is better and why. You can use this repository of previous assignments, but don't forget to mention any kind of inspiration source.

The assignment is designed to be individual and to pose you some difficulties (both technological and scientific) for which you should identify a working solution by the end of the semester. Each semester, a distinct AI tool is assigned to two students. Your are encouraged to collaborate, especially during the the installation and example understanding phases (W_1-W_4). The quicker you get throughout these preparatory stages, the more time you have for your own project.

Class attendance. I expecte active participation at all activities. Keep in mind the exam can include any topic that was covered during class, explained on the board, or which emerged from discussions among participants. Missing lab assignments or midterm leads to minimum grade for that part. You are free to manage your laboratory classes - meaning that you can submit the project earlier - as long as you meet all the constraints and deadlines.

Chapter 2

A1: Search

Chapter 3

A2: Logics

Chapter 4

A3: Planning

PDDL = Planning Domain Definition Language

Components of a PDDL planning tasks:

-**Objects**: Things in the world that interest us.
-**Predicates**: Properties of objects that we are interested in; can be true or false.
-**Initial state**: The state of the world that we start in.
-**Goal specification**: Things that we want to be true.
-**Actions/Operators**: Ways of changing the state of the world.

Planning tasks specified in PDDL are separated into two files:

1. A **domain file** for predicates and actions.
2. A **problem file** for objects, initial state and goal specification.

Farmer problem:
A farmer and his goat,wolf and cabbage are on the North bank of a river. They need to cross to the South bank. They have a boat, with a capacity of two; the farmer is the only one that can row. If the goat and the cabbage are left alone without the farmer,the goat will eat the cabbage. Similarly,if the wolf and the goat are together without the farmer, the goat will be eaten.

-**Objects**:The two realm(south,north),goat,wolf,cabbage,farmer,boat.
-**Predicates**:
-realm(r):true if r is a realm
-goat(g):true if g is a goat
-wolf(w):true if w is a wolf
-cabbage(c):true is c is a cabbage
-farmer(f):true if f is a farmer
-at-boat(b,f,o):true if b is boat and f is farmer and o is object and farmer and an object(goat,wolf or cabbage) is in boat
-at-realm(x):true if x is a realm and the farmer is on the realm x
-at-goat(x,y):true if x is goat,y is realm and x is on the realm y
-at-wolf(x,y):true if x is wolf,y is realm and x is on the realm y
-at-cabbage(x,y):true if x is cabbage,y is realm and x is on the realm y
-free(x):true if x is the farmer and x does not hold an object
-carry(x,y):true if x is the farmer, y is an object, and x holds y

-**Initial state**:The farmer,goat,wolf,cabbage,boat are on the north realm and the boat is empty
-**Goal specification**:All objects must be in the south realm
-**Actions/Operators**:The farmer can move between realms, pick or drop an objects

Bibliography

http://www.automatedreasoning.net/docs_and_pdfs/a_primer_for_automated_reasoning.pdf https://en.wikipedia.org/wiki/First-order_logic

Appendix A

Your original code

domain.pddl file:

```
(define (domain farmer)
    (:predicates (realm ?r)
                 (goat ?g)
                 (wolf ?w)
                 (cabbage ?c)
                 (farmer ?f)
                 (at-boat ?b ?f ?o)
                 (at-realm ?x)
                 (at-goat ?x ?y)
                 (at-wolf ?x ?y)
                 (at-cabbage ?x ?y)
                 (free ?x)
                 (carry ?x ?y))

    (:action move
       :parameters (?from ?to)
       :precondition (and (realm ?from) (realm ?to) (at-realm ?from)
       :effect (and (at-realm ?to)
(not (at-realm ?from)))))

    (:action pick-goat
       :parameters (?goat ?wolf ?cabbage ?realm ?farmer ?boat)
       :precondition (and (goat ?goat) (realm ?realm) (farmer ?farmer)
(at-goat ?goat ?realm) (at-realm ?realm) (free ?farmer)
(not (at-boat ?boat ?farmer ?wolf))
(not (at-boat ?boat ?farmer ?cabbage)))
       :effect (and (at-boat ?boat ?farmer ?goat)
(carry ?farmer ?goat)
(not (at-goat ?goat ?realm))
(not (free ?farmer))))

    (:action pick-wolf
       :parameters (?goat ?wolf ?cabbage ?realm ?farmer ?boat)
```

```
        :precondition (and (wolf ?wolf) (realm ?realm) (farmer ?farmer)
(at-wolf ?wolf ?realm) (at-realm ?realm) (free ?farmer)
(not (at-boat ?boat ?farmer ?goat))
(not (at-boat ?boat ?farmer ?cabbage)))
        :effect (and (at-boat ?boat ?farmer ?wolf)
(carry ?farmer ?wolf)
(not (at-wolf ?wolf ?realm))
(not (free ?farmer))))

    (:action pick-cabbage
        :parameters (?goat ?wolf ?cabbage ?realm ?farmer ?boat)
        :precondition (and (cabbage ?cabbage) (realm ?realm) (farmer ?farmer)
(at-cabbage ?cabbage ?realm) (at-realm ?realm) (free ?farmer)
(not (at-boat ?boat ?farmer ?goat))
(not (at-boat ?boat ?farmer ?wolf)))
        :effect (and (at-boat ?boat ?farmer ?cabbage)
(carry ?farmer ?cabbage)
(not (at-cabbage ?cabbage ?realm))
(not (free ?farmer))))

    (:action drop-goat
        :parameters (?goat ?wolf ?cabbage ?realm ?farmer)
        :precondition (and (goat ?goat) (realm ?realm) (farmer ?farmer)
(carry ?goat ?farmer) (at-realm ?realm)
(not (at-cabbage ?cabbage ?realm))
(not (at-wolf ?wolf ?realm)))
        :effect (and (at-goat ?goat ?realm)
(free ?farmer)
(not (carry ?goat ?farmer))))

    (:action drop-wolf
        :parameters (?goat ?wolf ?cabbage ?realm ?farmer ?boat)
        :precondition (and (wolf ?wolf) (realm ?realm) (farmer ?farmer)
(carry ?wolf ?farmer) (at-realm ?realm)
(at-goat ?goat ?realm))
        :effect (and (at-wolf ?wolf ?realm)
(at-boat ?boat ?farmer ?goat)
(not (carry ?wolf ?farmer))))

    (:action drop-cabbage
        :parameters (?goat ?wolf ?cabbage ?realm ?farmer ?boat)
        :precondition (and (cabbage ?cabbage) (realm ?realm) (farmer ?farmer)
(carry ?cabbage ?farmer) (at-realm ?realm)
(at-wolf ?wolf ?realm))
        :effect (and (at-cabbage ?cabbage ?realm)
(free ?farmer)
(not (carry ?cabbage ?farmer)))))
```

problem.pddl file:

```
(define (problem farmer)
    (:domain farmer)
    (:objects northRealm,southRealm,goat,wolf,cabbage,farmer,boat)
    (:init (realm northRealm) (realm southRealm)
(farmer f1)
(boat b1)
(cabbage c1)
(goat g1)
(wolf w1)
(at-realm northRealm)
(free f1)
(at-goat g1 northRealm)
(at-wolf w1 northRealm)
(at-cabbage c1 northRealm))
    (:goal (and (at-realm southRealm) (at-goat g1 southRealm) (at-wolf w1 southRealm)
(at-cabbage c1 southRealm))))
```