Teaching Intelligent Agents using NetLogo

Ilias Sakellariou[1], Petros Kefalas[2], Ioanna Stamatopoulou[2]

[1]Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, iliass@uom.gr

[2]Department of Computer Science, CITY College, Thessaloniki, Greece, kefalas@city.academic.gr, istamatopoulou@seerc.org

In the context of an Agent and Multi-Agent Systems course, satisfying the students demands for hands-on practice presents an interesting challenge. Educators have reported a variety of environments and techniques they use in order to increase active learning. In this paper we record our experience using NetLogo as part of the practical coursework that students need to carry out within an Intelligent Agents course. We argue that NetLogo meets most of the requirements that suit our criteria. In addition, we describe two extra NetLogo libraries provided to students, one for BDI-like agents (goal-oriented agents) and one for ACL-like (Agent Communication Language) communication. We present a few scenarios that we use in coursework handouts and how the partially developed environment we provide for each scenario facilitates practical agent design and simulation, thus satisfying the learning outcomes of the practical work and the course as a whole.

Keywords
Artificial Intelligence, Intelligent Agents, Practical Assessment, Agent Simulation Platforms

1. Introduction

Courses on Agents and Multi-Agent Systems (AMAS) have started to complement Computer Science and other related curricula during the last decade. AMAS is listed in the ACM/IEEE Computing Curricula [12] as part of Intelligent Systems and University Departments have chosen to offer a course on AMAS (under a wide variety of titles) either as a core or an optional course during undergraduate and/or postgraduate studies. Some chose to integrate AMAS principles into other courses. Due to the wide foundations and applicability of AMAS, it is expected that there is also a lot of diversity with respect to the learning outcomes and content (focus on theory or applications) as well as the context (Artificial Intelligence related or mainstream Computer Science related) in which such course is offered. This also explains the variety of valid options (teaching and assessment methods, practical work, tools, demonstrations etc.) when designing the syllabus as well as the wide variety of experiences reported in teaching.

It is therefore important to briefly define first the context to which this paper refers to. We introduced a course entitled "Intelligent Agents" (IA for short) 7 years ago in our 3 year Computer Science undergraduate curriculum. This is a final year, final semester course obligatory for all students, ranging from 25 to 50 since 2001. It covers a wide range of standard topics in AMAS (mixture of theory and practice as shown below) with no particular emphasis on any, and is assessed through coursework (practical work) and unseen final examinations. We felt that exposing the students to advanced technologies like those involved in AMAS would significantly broaden their horizons on cutting-edge information technologies.

During the first couple of years, we have intensively tried to enrich the lectures with software and video demonstrations, but the feedback from the students reported that, although they enjoyed the concept and score highly the overall teaching, they felt that the course was too theoretical with no hands-on experience. On the other hand, we knew that any attempt to assess them through some kind of IA program development would add significantly to the existing heavy load of the last semester and their effort to complete a parallel individual third year project, which is worth 4 times as much as a single course and 20 times as much a single coursework for any course in that semester. It was a challenge for us to solve the obvious problem, that is, keep the course but allow space for some practical program development. The requirements we set for driving our final decision were the following:

- have a simple environment that presents the minimum installation problems,
- provide easy visualization in order to better view of the agent behaviour and increase student interest,
- choose an easy to learn and use language thus keeping a small learning curve,
- the environment should clearly demonstrate the difficulties in AMAS programming,
- it should have support for at least reactive architectures,
- it should preferably support BDI-like and hybrid architectures,
- it should provide means for, at least limited, communication, message exchange and interaction.

Apparently, the final decision was not easy, since all these are not fully met by existing development environments. We came up with NetLogo [18] which at that time was at early stages of development but offered the minimum required to our needs. We dealt with the rest in a way that is described later on in this paper.

In section 2, we discuss in more detail the requirements and the choices available in order to integrate some practical experience in a IA course. Section 3 provides an analytic description of the IA course we offer at the Computer Science Department of CITY College. NetLogo is briefly presented in Section 4 and the extensions we suggested and implemented are listed in Section 5. In the following section 6, we present sample coursework assignments given to students. Finally, we conclude with discussion and future work.

2. Practical Work in AMAS Courses

It is widely recognised that some kind of practical program development related to AMAS courses is needed, in order for the students to understand the concepts and meet the learning outcomes of the course. It is also reported that there are several choices educators may follow, depending on the emphasis and focus they give to certain aspects of AMAS, e.g. architectures, communication and interaction protocols, applications etc. For example, various tools and environments for AMAS have been reported to assist the educational process, like RoboCup, NetLogo, TAC, FIPA-OS, JADE, JadeX, Jason, Protege etc. [2,10,19,3,5,6]. All aim to improve students' active learning in the context of AMAS, others by engaging students in writing code (e.g. Java), others by allowing development of peripheral to agents structures (e.g. Ontologies).

Obviously, educators feel happy about the value of integrating tools into their courses, but admittedly, most of the times express concerns about their complexity and the time spent by the students to reach the required level of skills in order to produce something useful to their eyes. Given the short period of a semester course, normally ranging from 10 to 15 weeks maximum, this is always going to be an issue.

On the other hand, students would like to see a more realistic outcome of their work. If this is the process of a competition like game (e.g. Trading Agent Competition) or a visualisation of the agents' environment (e.g. RoboCup), the understanding and satisfaction seems to be increasing. One could also argue that a simple robotic platform (e.g. Lego Mindstorms, RoboSapiens, i-SOFT etc.) [4,7] could also serve the purpose, since in the students perception a robot (from a science fiction perspective) matches with their perception of an agent. Although, such an approach is feasible and sometime desirable in more engineering-oriented courses, it still suffers from complexity issues due to the fact that students need to take into account non-symbolic percepts and additional hardware devices and protocols.

3. An Intelligent Agents Course

The IA course offered at CITY College is designed as a natural sequel of three other AI related courses taught in the previous three semesters, namely "Logic Programming", using Prolog, "Artificial Intelligence Techniques", introducing students to AI covering search and knowledge representation issues, and "Artificial Intelligence", which exposes students to more advanced AI issues such as genetic algorithms, fuzzy rule-based systems, planning, knowledge-based systems, machine learning etc. The aims of our IA course are to:

- introduce the student to the notions of intelligent agents and provide an introductory study of the various types of intelligent agents, their architecture, strengths and limitations;
- introduce multi-agent systems and the various issues involved in agent communication and interaction;
- discuss possible application areas of the intelligent agent technology through examples and case studies as well as demonstrate how agents can revolutionize human-computer interaction;
- present the advantages of the agent-based approach to engineering complex software systems.

By the end of the course we expect a student to be able to (learning outcomes):

- understand the basic notions of agent systems;
- explain the difference between agents and other programs;
- understand the key concepts involved with modelling agents and multi-agent systems;
- discuss and synthesise agent solutions;
- sensibly design multi-agent systems;
- cope with key issues in implementing agent-based and multi-agent systems;
- identify tasks in information systems that present possible applications areas of the technology;
- critically analyse the expected benefits of using AMAS technology.

More particularly, the topics covered include, among others, agent architectures (logic, reactive, BDI, hybrid), communication and interaction protocols (speech acts theory, agent communication languages, knowledge communication, blackboards, Contract Net protocol, auctions, negotiation), biology-inspired agents (Ant Colony Optimization, Bio-Networking, Artificial Life), planning, learning and mobile agents, agent theories (intentional notions: information, motivation and social attitudes), multi-agent system software engineering methodologies (AAII methodology, Cassiopeia, Agent UML), Semantic Web basics etc. The recommended textbook is [20], with additional readings from [17,13] and custom made slides.

The material of the course is delivered through a series of formal lectures, summing a total of 30 hours over ten of the weeks of the semester (three hours/week). The idea is that roughly one hour per week is a demonstration session during which students may, for example, watch educational videos, tool demonstrations and be exposed to applications of AMAS, while the other two hours per week is an interactive lecture. Finally, there is also a two-hour consolidation class, in the middle of the semester and another two-hour revision class at the end of it. The students' performance is assessed through two assignments, each contributing 15% of the total mark, and a two-hour formal examination at the end of the semester, for 70% of the total mark.

As in any typical IA course, it was very important for us that through the assignments during the semester we manage to assess a number of different aspects involved with the design and development of agent-based and multi-agent systems, such as agent architectures, communication protocols and languages, and interaction issues including coordination, negotiation, auctions etc. Most importantly, however, what we aimed for is that the students gain hands-on experience and that they are being assessed from a very practical perspective (even more so since assessment of the theoretical aspects is being achieved through the formal examination). With this motive, which had been further reinforced by our previous experience and students' feedback, we decided that the assignments should involve the development of multi-agent systems in an appropriate environment that would be both easy to use, not imposing an extra burden to the students, as well as maximally rewarding.

We chose NetLogo as an environment to assign assessed coursework to our students. Although it is not designed so as to target all the aspects involved in AMAS, it meets several of the requirements listed section 1. It is easy to install and provides with a plethora of interesting examples ready to run. The language is functional and although learning another programming paradigm might put some burden, our students have been acquainted with declarative programming through their previous courses and students appear to be less "intimidated" by it, in comparison to Java or Prolog for example. Therefore, the learning curve seems to be small and students are ready to produce a decent program in a short time. Additionally, it allows the visualisation of the developed systems; this is valuable to students as it helps them both to gain a better understanding as well as get immediate feedback for their efforts.

On the other hand, since NetLogo is not specifically designed for AMAS, it suffers from not being able to provide ready-made constructs for symbolic perception, goal-oriented agents, communication and coordination. In order to compensate for the features NetLogo lacks, we decided to work towards these issues in advance and give the students an appropriate small set of extensions that have been particularly developed to facilitate a BDI-like agent architecture and inter-agent communication issues.

4. NetLogo as an Educational Tool

NetLogo is a modelling environment targeted to the simulation of multi-agent systems that involve a large number of agents. The platform aims to provide "a cross-platform multi-agent programmable modelling environment" [18].

The system offers a simple and expressive programming language and facilities for GUI creation on which custom visualizations of the studied multi-agent systems can be created with particular ease. There is an extensive set of primitives, good support for floating point mathematics, random numbers and plotting capabilities. The environment is an excellent tool for rapid prototyping and initial testing of multi-agent systems, particularly suited to systems with agents situated and operating in a restricted space, as well as an excellent animation tool of the modelled system. It also proved to be an excellent educational platform for teaching IA.

The main entities of NetLogo are the patches, the turtles and the observer[1]. The observer simply controls the experiment, in which turtles and patches actively participate. *Patches* are stationary "agents", i.e. components of a grid on which *turtles* exist, i.e. agents that are able to move, "live" and interact. Both patches and turtles can inspect the environment around

[1] The recent version (4.0) also offers links, however this work does not involve these new entities.

them, for example detect the existence of other agents, view the state of their surrounding patches/turtles, and modify the environment. Probably the feature that most greatly enhances the modelling expressiveness of the platform is the fact that each patch and turtle can have its own user-defined variables: in the case of patches this allows modelling complex environments by including an adequate number of variables that describe it sufficiently and in the case of turtles it simply means that each agent can carry its own state.

The programming language allows the specification of the behaviour of each patch and turtle, and of the control of the execution. Monitoring and execution of the agents is controlled by the *observer* entity that "asks" each agent to perform a specific computational task.

As it has been argued elsewhere [16], we found NetLogo to be suitable for modelling multi-agent systems that we were dealing with in our course, since each NetLogo agent[2]:

- perceives on its environment and acts upon it,
- carries its own thread of control,
- is autonomous.

However, although NetLogo is an excellent tool that exposes the students to the difficulties of MAS development, it lacks build-in support for implementing communicating agents with intentions and beliefs.

5. Extending NetLogo with Libraries for Intelligent Agents

Being a platform that is primarily targeted to modelling social and natural phenomena, NetLogo supported fully the creation and study of reactive agent systems. Indeed, one of the first models that we studied was the Luc Steels Mars Explorer experiment [15] that presents many similarities to the ant colony foraging behaviour experiment included in the library models (ants can be modelled and studied as reactive agents [9,14]). However, the study of BDI-like agents (those that exhibit goal-oriented behaviour through Beliefs-Desires-Intensions) that are able to communicate with explicit symbolic message exchange was not supported. Thus, taken into consideration the fact that NetLogo fulfilled the majority of our requirements we decided to extend the platform by providing the students with one library for building simple BDI-like agents and one for FIPA-like message exchange, with their accompanying manuals.

5.1 BDI-like agents in NetLogo

Although, we could have adopted as an option to link through the JAVA interface of NetLogo an already existing BDI development platform, as for example JAM [11], this would have made the installation of the coursework platform a lot more difficult for students and would have increased significantly the learning curve, due to the complexity of such a fully fledged development environment. Thus, we decided to provide a simpler alternative to develop limited BDI-agents by providing the necessary primitives through a NetLogo library[3].

The simple BDI architecture that we have followed, follows a PRS-like [8] model, i.e. there is a set of intentions (goals) that are pushed into a stack; of course the implementation is far from delivering all the features of systems like JAM, but still can be used in implementing simple BDI agents in the NetLogo simulation platform.

[2] Actually, the excellent page that Jose Vidal (http://jmvidal.cse.sc.edu/) maintains for NetLogo models, was a starting point for our work.

[3] NetLogo being a simple platform, does not support libraries in the classic sense found in other programming environments; by the term library we refer to a set of procedures and functions (reporters) that the students are given and include in their own code

An intention consists of two parts: the *intention name* and a condition that we call *intention-done*. The former maps to a NetLogo procedure (possibly user defined) while the latter maps to boolean NetLogo reporter (function) (again possibly user defined). The semantics are the standard followed by other architectures: an agent must pursue an intention until the condition described in the intention-done part evaluates to true. For example the following intention of a luggage carrier agent working at an airport:

```
["move [23 20]" "at-gate 3"]
```

states that the agent is currently committed to moving towards the point `(23, 20)` and it will retain the intention until the reporter function "`at-gate 3`" evaluates to true. Note that the user has to specify both the procedure and the reporter, that map to the two parts of the intention.

The main concept behind the present implementation is the *intention stack* where all the intentions of the agent are stored. Agents follow the execution cycle shown below:

```
IF the intention stack is not empty THEN do:
  1. Get the top intention I from the stack
  2. Execute I
  3. If intention-done evaluates to true then remove I from stack
ELSE do nothing
```

The NetLogo implementation is rather straightforward: each intention is represented by a list of two elements, one for each part. The intention stack is a list stored in a specific turtle-own variable, which simply means that each agent has its own stack. The execution cycle is encoded in the procedure `execute-intentions`, that is called to invoke the agent's behaviour. The library also provides a set of reporters and procedures to the user (student) for adding and removing intentions on the stack, inspecting the current intentions, set time-outs as intention-done conditions, etc. For example, the following line:

```
add-intention "move [23 20]" "at-gate 3"
```

will add the corresponding intention to the stack of the calling agent.

To further support the BDI architecture, facilities for managing beliefs were also created. Although, the latter was not really necessary, since it is rather simple to store any information on a related turtle variable, we have designed a set of procedures and reporters that would form an abstraction layer that facilitates the students to manage agent beliefs, without getting into too many details about how to program in the NetLogo language.

A belief consists of two elements: the *type* and the *content*. The former declares the type of the belief, i.e. indicates a "class" that the belief belongs to. Examples could include any string, e.g. "`position`" "`agent`" etc. Types facilitate belief management, since they allow to check for example whether a belief of a specific type exists or the removal of multiple beliefs at once. The content on the other hand, is the specific information stored in the belief. It can be any NetLogo structure (integer, string, list, etc.). Obviously, there might be multiple beliefs of the same type with a different content, however two beliefs of the same type and content cannot be added. For instance, `["agent" 5]` and `["location" [3 7]]` are examples of beliefs that the agent can have.

Belief management is done through a set of reporters and procedures that allow the creation, removal, checking of the agent's beliefs. For example, the following line:

```
add-belief create-belief "plane-at" [23 15]
```

will include a belief of type "`plane-at`" with content "`[23 15]`" in the agent's beliefs. In the current implementation, all agent beliefs are stored in an agent own variable named `beliefs`.

5.2 FIPA-like Message Passing

The ability to exchange symbolic messages is rather important in a course that includes agent communication and interaction protocols, such as the contract net protocol. Thus, it was necessary for us to somehow enhance NetLogo with explicit message communication primitives. Messages closely follow the FIPA ACL (the Agent Communication Language of the Foundation for Intelligent Physical Agents) message format, i.e. are lists of the form:

```
[<performative> sender:<sender> receiver:<receiver>
  content: <content>..]
```

For example, the following message was send by agent (turtle) `5` to agent `3`, its content is "`plane-at 23 15`" and the message performative (FIPA) is "inform".

```
["inform" "sender:5" "receiver:3" "content:" "plane-at 23 15"]
```

A message may only include the above fields (performative, sender, receiver, content), omitting others such as the ontology field, for example, used by FIPA, thus assuming that all agents use the same ontology. The library, however, allows the creation and addition of any custom field that may be considered necessary.
As can be seen from the above, agents are uniquely characterized by an ID (number) that is in fact an integer value automatically assigned at the time of their creation by NetLogo ("`who`" NetLogo variable). We have adopted this naming scheme since it greatly facilitates the development of the message passing facilities. Message passing is asynchronous. A set of reporters and providers allow easily creating/sending/receiving/processing messages between NetLogo agents. For example the code below (assuming that the calling agent is 8):

```
let  somemsg create-message "inform"
set  somemsg add-receiver 5 somemsg
set  somemsg add-content "plane-at 23 15" somemsg
send somemsg
```

will send to agent `5` the message that follows:

```
["inform" "sender:8" "receiver:5" "content:" "plane-at 23 15"]
```

Of course, both libraries (BDI and FIPA) take advantage of functional features of the NetLogo programming language. The exact message will be send by simply issuing the following command:

```
send add-content "plane-at 23 15"
     add-receiver 5 create-message "inform"
```

It should be noted that there are also primitives that allow broadcasting a message to a "class" (breed) of NetLogo agents.

Incoming messages for each agent are stored in a variable named `incoming-queue`, which is in fact a list. This is a "user-defined" variable that each agent must have in order to be able to communicate. Sending a message to an agent simply means adding the message to its `incoming-queue` list; it does not require an explicit receive command to be invoked on the receivers side. At any time the agent has the ability to obtain the messages from its queue using the reporters and procedures provided.

Both libraries, FIPA and BDI, were fully implemented in the NetLogo language: thus, their "installation" was trivial, since students had only to include the given library code in their models. Error checking and debugging facilities were kept to a minimum to avoid having efficiency issues.

We have used both libraries for a number of years in our classes: students have found them easy to use and did manage to implement multi agent systems that involve communicating BDI agents under an interaction protocol. The interested reader may find the libraries, brief manuals and examples, at [1].

6. Practical Assignments using NetLogo

Using the programming facilities described in the previous sections, we have designed a number of practical assignments for the IA course. Each year, there are two practicals handed to the students: the first aims to allow students to have a gentle introduction to NetLogo and involves developing a reactive agent system. The second involves BDI agents that cooperate via message exchange and under the Contract Net interaction protocol. We have always tried to have close to the real world scenarios, so that the students' interest and motivation is higher. In one academic semester, both practical handouts refer to the same scenario, so that students see different aspects of applying AMAS technology to the same problem.

6.1 Practical #1: rescue units scenario

One of the scenarios that we have used, is the *rescue units* scenario, in which agents operate in a disaster area to efficiently locate and rescue victims. The rescue procedure is rather simple: a rescue unit locates the civilian in need (victim) and provides oxygen and water to the victims, so that they can be sustained in life until transportation arrives. Thus the rescue team consists of rescue-unit agents, i.e. autonomous vehicles that can move around the disaster area, locate (detect) any civilians in danger (victims) and temporarily rescue them. In the scenario there also exists a base station, installed in a central location, that provides refuelling and renewal of medical supplies (oxygen, water, etc.) services.

Rescue units have a very limited set of sensors, including a sensor for detecting civilians in danger, which operates at a very close range, an obstacle detection sensor (to avoid obstacles and other rescue units), sensors detecting low fuel levels etc. Agents also have limited abilities; they can move around in the disaster area, provide immediate attention to the victims, move towards the base, since the latter transmits a signal they can follow, etc. Agents do not have message exchange capabilities and are to follow the reactive architecture. The similarities of the above scenario with Luc Steels Mars explorer are obvious. This scenario is found in the first handout of the semester and upon completion of the practical we expect the students to:

- understand in depth the reactive agent architecture, its advantages and disadvantages,
- design a simple reactive agent to perform a task,
- build a simple prototype of a reactive agent system in NetLogo,
- evaluate the design choices made based on simulation results.

6.2 Requirements and assessment

In order to allow students to concentrate on the above, we provide all the sensors and agent abilities implemented in NetLogo, as well as an initial setup of the experiment environment; what we ask is the agent architecture, any enhancement that could help to increase the efficiency of the system and experimental results. Students are assessed according to the following criteria:

- Correctness, originality and justification of the proposed agent architectures;
- Implementation and code documentation;
- Analysis and presentation of experimental results;
- Presentation of the report (clarity, structure).

6.3 Practical #2: extended rescue units scenario

The same scenario, augmented appropriately, acts as the basis of the second practical, which is far more demanding. In this case, there are also ambulance agents that are autonomous transportation units, able to collect the rescued civilians and bring them to the base. However, an ambulance can save a victim only if it has been discovered by a rescue unit. In this extended scenario, all agents have the ability to exchange explicit symbolic messages and follow BDI or hybrid architectures. Overall, agents are far more complex but since students have already been exposed to the NetLogo platform, they are now asked to contribute a bit more code. For this second assignment students are asked to implement a cooperation protocol between rescue-units and ambulances, specify the FIPA-ACL messages needed to be exchanged under the cooperation protocol and implement everything they propose in NetLogo, using the libraries mentioned in the previous sections.

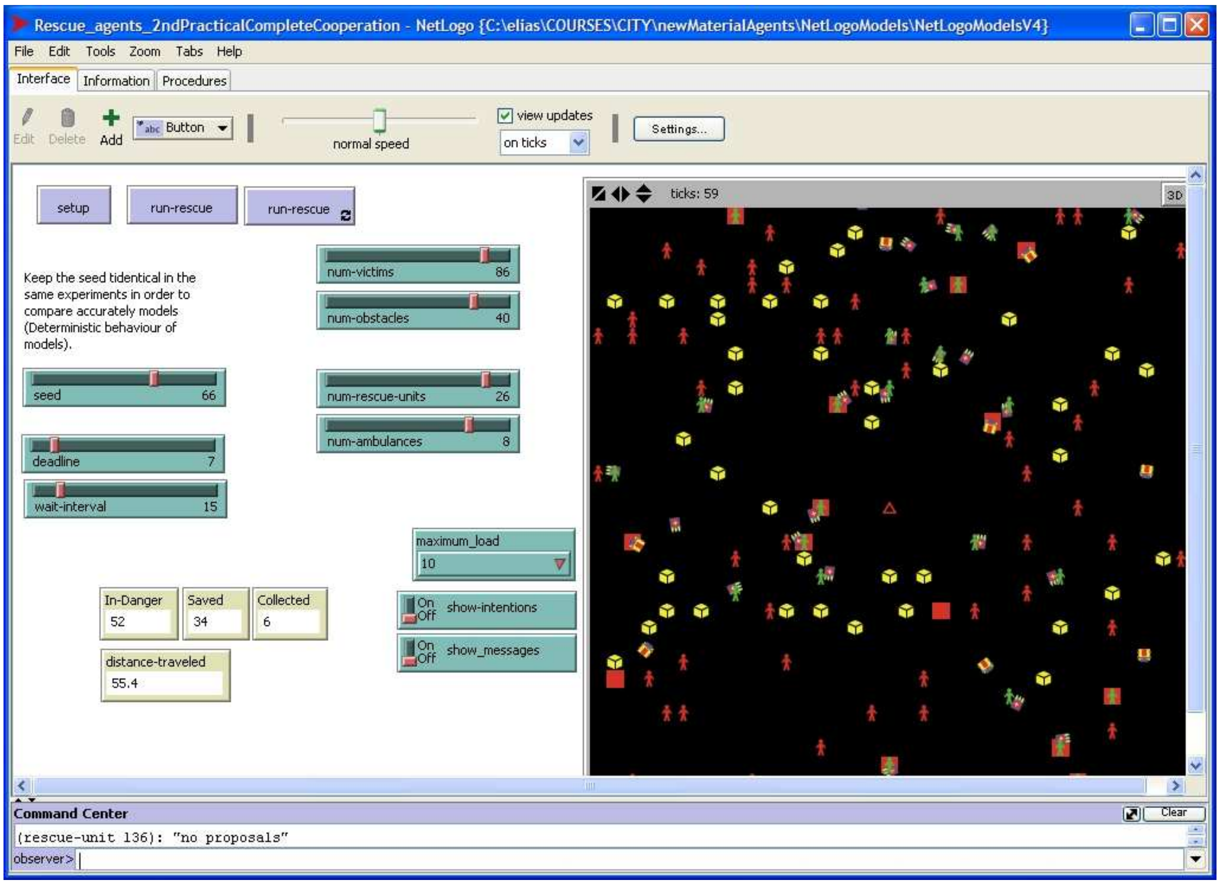
The expected learning outcomes of the second practical are that the students:

- understand in depth the issues and the difficulties involved when building a multi-agent system, such as agent communication languages, interactions protocols, language used etc.,
- use an existing library to construct FIPA ACL-like messages and implement an interaction protocol,
- propose a suitable agent architecture to perform a problem solving task,
- build a simple prototype of a multi-agent system in NetLogo,
- evaluate the design choices made based on simulation results.

Students are assessed according to the following criteria:

- Correctness, originality and justification of the proposed agent architectures;
- Correctness and justification of the cooperation protocols proposed;
- Implementation and code documentation;
- Analysis and presentation of experimental results;
- Presentation of the report (clarity, structure).

It should be noted that the code provided to the students, allows them to modify various parameters of the experiment, i.e. the number of rescue units, the number of ambulances, initial fuel, number of obstacles, as well as measure various efficiency criteria, such as total time, total distance travelled by all agents, etc. Figure 1 shows the complete environment of the rescue unit scenario.



Rescue_agents_2ndPracticalCompleteCooperation - NetLogo {C:\elias\COURSES\CITY\newMaterialAgents\NetLogoModels\NetLogoModelsV4}

normal speed

view updates ✓

on ticks

ticks: 59

Keep the seed tidentical in the same experiments in order to compare accurately models (Deterministic behaviour of models).

num-victims 86

num-obstacles 40

seed 66

num-rescue-units 26

num-ambulances 8

deadline 7

wait-interval 15

maximum_load 10

In-Danger 52

Saved 34

Collected 6

distance-traveled 55.4

show-intentions On

show_messages On

Command Center

```
(rescue-unit 136): "no proposals"
observer>
```

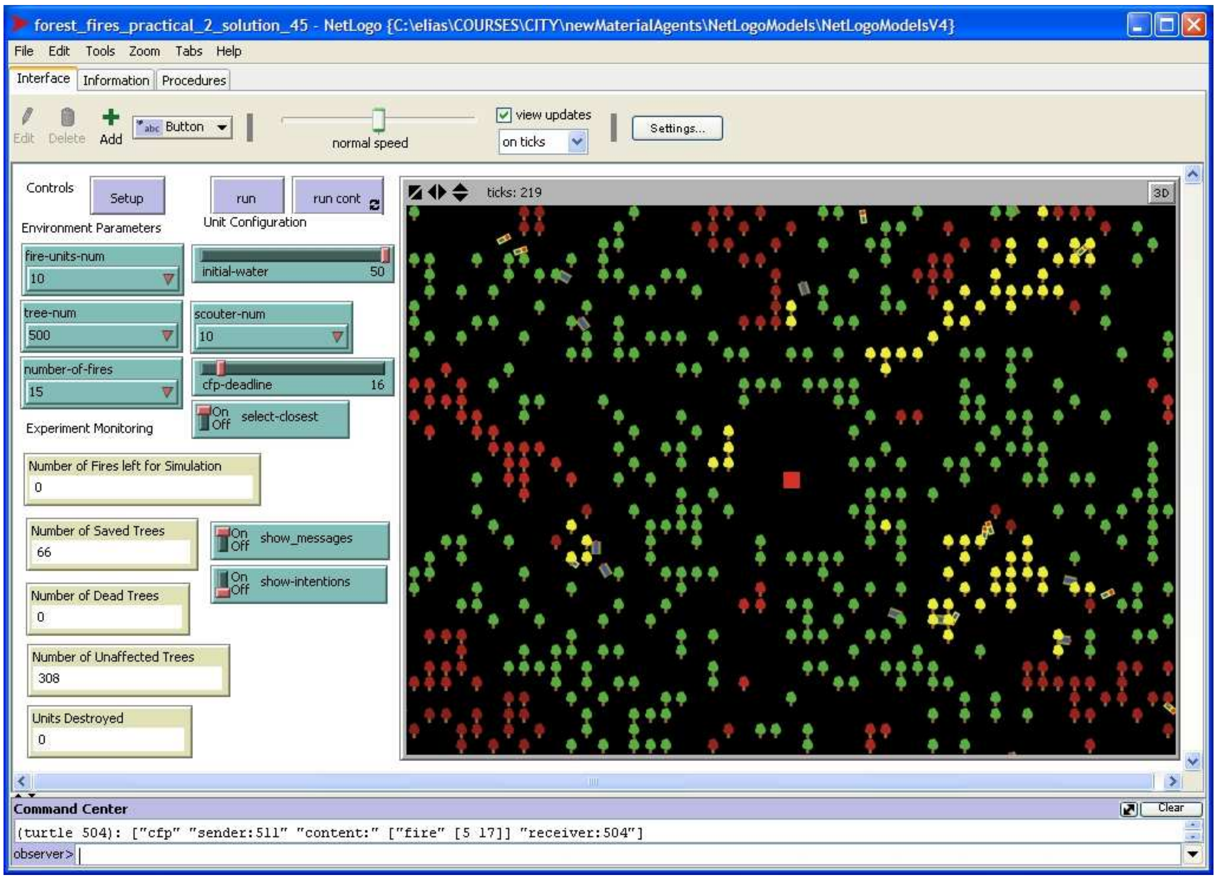
Figure 1 NetLogo ScreenShot of the Rescue Unit Scenario

6.4 Other similar scenarios

A scenario similar to the above one that has been presented to students in another academic year, involved *forest fires*, keeping the same spirit and structure as in the rescue scenario, but in a different setting: for the first practical, students had to implement reactive agents that patrol a forest and extinguish small spots of fire before they spread. For the second practical, there are scouters that detect fire spots in the forest and ground units that move to the specific location and extinguish the fire, after a fire has been detected by a scouter.

In this model, fire spots (i) appear randomly during the execution of the experiment, and (ii) are spreading in adjacent trees over time, if they are not put out, i.e. there is a close to reality fire spread model against which the agents were competing. Various parameters can be set as, for example, the number of fire spots that randomly appear, the density of the forest, the number of scouter and ground units, the water supply of each unit, etc. The learning outcomes and assessment criteria have been as stated above and the students were again given the environment setup, sensors and actions of the participating agents. Figure 2 shows a screenshot of the forest fires environment.

Naturally, the same assignment "template" can be used in a variety of other scenarios such as taxis transporting passengers in a city, vacuum cleaning a floor (which has been actually used in another academic year), logistics problems, etc. The above presented coursework handouts may also be found at [1].



fire-units-num 10
initial-water 50
tree-num 500
scouter-num 10
number-of-fires 15
cfp-deadline 16
select-closest On
Number of Fires left for Simulation 0
Number of Saved Trees 66
show_messages On
show-intentions On
Number of Dead Trees 0
Number of Unaffected Trees 308
Units Destroyed 0

```
(turtle 504): ["cfp" "sender:511" "content:" ["fire" [5 17]] "receiver:504"]
observer>
```

Figure 2 NetLogo ScreenShot of the Forest Fires Scenario.

7. Discussion and Conclusions

The overall impression that we gain from students is that they enjoy the practical aspect of the course. This is reported in the formal evaluation of the course at the end of the semester. The overall student satisfaction increased in comparison to the early years of the introduction of the course in which the practical component was restricted only to design issues. The interest factor of the course has also increased. In addition to student formal feedback, it is also clear to us that the overall student performance in final examinations is increased due to better and deeper understanding of the issues around AMAS. The final examination contains only theoretical questions and design exercises (no programming involved) but the students present better solutions to the problems posed. Formal analysis of the data is an issue which we need to pursue in the coming years.

We have distributed a questionnaire to our final year students asking about their opinion on NetLogo and its use in the coursework assessment as well as whether the platform facilitated better understanding of the theoretical issues. The results which are listed in Table 1 demonstrate that our initial objectives about using NetLogo as a tool for teaching AMAS were met. In particular, the vast majority of students agrees that use of another programming language would not be preferable. In addition, students realise that the libraries provided for BDI and FIPA-ACL facilitated the development of a fully functional NetLogo code for the two assignments. Further collection of data is necessary in order to have a full validation of our approach, since our research involves only one academic year.

In informal discussions we had with students, they are happy with the coursework assignments. They appreciate the fact that a partial program and an initial setup of the environment is given to them in advance. Although, at first they express their concern about yet another programming language to deal with, their worries were relaxed when they had their first encounter with NetLogo. They really liked the sense of seeing and experimenting with the virtual world they developed with minimum programming effort. Thus, they had more time to design and implement the "real thing", i.e. the agent systems requested from them. The NetLogo libraries and the associated manuals we provided, facilitate their concentration on developing the systems for the given scenarios to a great extent. Finally, they appreciated the fact that the scenarios given seemed realistic enough to attract their interest.

Table 1 Students opinion on NetLogo.

	Strongly Disagree or Disagree (%)	**Neutral (%)**	**Strongly Agree or Agree (%)**
NetLogo was easy to install	0	0	**100**
NetLogo's visual environment helped to better understand agents behaviour	0	6.2	**93.8**
NetLogo demos helped to better understand the agent theory	6.2	31.3	**62.5**
NetLogo practicals increased my interest in the unit	0	18.8	**81.2**
NetLogo practicals helped me to better understand the agent theory	0	18.8	**81.2**
NetLogo language was easy to learn	0	12.5	**87.5**
Previous encounter with logic prog. helped me in learning NetLogo language	12.5	**56.2**	31.3
Practical#1: Initial code given helped me in developing the solution	0	6.2	**93.8**
Practical#2: Libraries for BDI and FIPA-ACL helped me in developing the solution	7.1	21.4	**71.4**
I would prefer to use another language (e.g. Java) for the practicals.	**62.5**	25	12.5
I would prefer to use another language (e.g. Prolog) for the practicals.	**68.7**	31.3	0
I would prefer to use actual robots for the practicals.	25	18.8	**56.2**
NetLogo practicals were fun!!!	12.5	31.3	**56.2**

We believe that the teaching objectives for this course are met through the proposed approach. All learning outcomes are assessed and in particular we have managed to accomplish the learning outcomes related to hands-on experience. It is also somehow important to us that a relatively good percentage of our graduates seek a postgraduate programme in the area of AI and AMAS.

Through the two libraries described in this paper, we managed to enable students to develop something more than a reactive agent. The libraries are by no means complete or fully fledged to meet the complete BDI and FIPA-ACL requirements. Future work is directed towards such extensions. For example, NetLogo offers a JAVA interface, through which a link to complete BDI packages, such as JAM, is possible. Although we consider it not suitable for educational purposes, as argued above, the idea could offer new opportunities for using the platform for research purposes. In addition, we are considering a rather ambitious development of an extendible/customisable game platform in NetLogo, in which educators will be able to setup game environment and rules, such as tank battles, RoboCup, etc., that will offer the possibility to apply the platform in an even simpler manner in the context of a course. Towards this direction, we have already implemented a prototype, but the work is not fully completed yet.

References

1 ‘Extending NetLogo to support BDI-like architecture and FIPA ACL-like message passing: Libraries’ manuals and examples’.http://eos.uom.gr/~iliass/projects/NetLogo.
2 M. D. Beer and R. Hill, ‘Teaching multi-agent systems in a UK new university’, in Proceedings of 1st AAMAS Workshop on Teaching Multi-AgentSystems, (2004).
3 M. D. Beer and R. Hill, ‘Multi-agent systems and the wider artificial intelligence computing curriculum’, in Proceedings of the 1st UK Workshop on Artificial Intelligence in Education, (2005).
4 S. Behnke, J. Muller, and M. Schreiber, ‘Playing Soccer with RoboSapien’, in RoboCup-2005: Robot Soccer World Cup IX, volume 4020 of Lecture Notes in Artificial Intelligence, 36–48, Springer, (2006).
5 R. H. Bordini, ‘A recent experience in teachingmulti-agent systems using Jason’, in Proceedings of the 2nd AAMASWorkshop on Teaching Multi-Agent Systems, (2005).
6 M. Fasli and M. Michalakopoulos, ‘Designing and implementing e-market games’, in Proceedings of the IEEE Symposium on Computational Intelligence in Games, pp. 44–50. IEEE Press, (2005).
7 E. Ferme and L. Gaspar, ‘RCX+PROLOG: A platform to use Lego Mindstorms^{TM} robots in artificial intelligence courses’, In Proceedings of the 3rd UK Workshop on AI in Education, (2007).
8 M. P. Georgeff and A. L. Lansky, ‘Reactive reasoning and planning’, in Proceedings of the AAAI Conference on Artificial Intelligence (AAAI’87), pp. 677–682, (1987).
9 M. Gheorghe, I. Stamatopoulou,M. Holcombe, and P. Kefalas, ‘Modelling dynamically organised colonies of bio-entities’, in Unconventional Programming Paradigms: International Workshop, (UPP’04), Le Mont Saint Michel, France, September 15-17, 2004, Revised Selected and Invited Papers, eds., J.-P. Banatre, P. Fradet, J.-L. Giavitto, and O. Michel, volume 3566 of Lecture Notes in Computer Science, 207–224, Springer-Verlag, (2005).
10 H. Hara, K. Sugawara, and T. Kinoshita, ‘Design of TAF for training agent-based framework’, in Proceedings of 1st AAMAS Workshop on Teaching Multi-Agent Systems, (2004).
11 M. J. Huber, ‘JAM: a BDI-theoretic mobile agent architecture’, in Proceedings of the 3rd Annual Conference on Autonomous Agents, New York, NY, USA, (1999). ACM.
12 Joint ACM/IEEE Task Force on Computing Curricula, ‘Computing curricula 2001’, ACM Journal of Educational Resources in Computing, 1(3), (2001).
13 S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 2002.
14 I. Stamatopoulou, I. Sakellariou, P. Kefalas, and G. Eleftherakis, ‘Formal modelling for in-silico experiments with social insect colonies’, in Current Trends in Informatics, eds., T. Papatheodorou, D. Christodoulakis, and N. Karanikolas, volume B of Proceedings of the 11th Panhellenic Conference in Informatics (PCI’07), May 18-20, Patras, Greece, pp. 79–89, (2007).
15 L. Steels, ‘Cooperation between distributed agents through self-organisation’, in Towards a New Frontier of Applications, Proceedings of the IEEE International Workshop on Intelligent Robots and Systems (IROS’90), pp. 8–14, (1990).
16 J. M. Vidal, P. Buhler, and H. Goradia, ‘The past and future of multiagent systems’, Iin Proceedings of 1st AAMAS Workshop on Teaching Multi-Agent Systems, (2004).
17 G.Weiss,ed. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MITPress, 1999.
18 U.Wilensky. Netlogo. http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer- based Modelling. Northwestern University, Evanston, IL.,1999.
19 A.B.Williams, ‘Teaching multi-agent systems using AI and software technology’, in Proceedings of the 1st AAMAS Workshop on Teaching Multi-Agent Systems, (2004).
20 M.Wooldridge, An Introduction to Multi Agent Systems, John Wiley & Sons, 2002.