Teaching and learning logic programming in virtual worlds using interactive microworld representations

Spyros Vosinakis, George Anastassakis, Panayiotis Koutsabasis

Spyros Vosinakis and Panayiotis Koutsabasis are assistant professors at the Department of Product and Systems Design Engineering, University of the Aegean, Greece.

George Anastassakis is postdoctoral researcher at the Department of Informatics, University of Piraeus, Greece.

Abstract

Logic Programming (LP) follows the declarative programming paradigm which novice students often find hard to grasp. The limited availability of LP visual teaching aids can lead to low motivation for learning. In this paper we present a platform for teaching Prolog programming in Virtual Worlds that enables the visual interpretation and verification of program results in a straightforward fashion and requires students to adopt a collaborative problem-solving approach. Results from student-centred evaluation are encouraging regarding group learning performance and user experience. The paper contributes to current practice of teaching and learning LP by proposing a metaphor and a system that makes use of toy world examples visualized in a shared 3D environment.
Practitioner notes

What is already known about this topic

Logic programming (LP) is an essential skill for Artificial Intelligence; novice students often find it hard to grasp.

Virtual Worlds (VWs) have positive impact on the learning of fundamental procedural programming concepts. There have not been any approaches for representing logic programs in VWs until now.

What this paper adds

The proposed system (MeLoISE) enables the representation of intuitive LP scenarios as interactive microworlds in a VW. MeLoISE allows for a collaborative problem-solving approach to teaching and learning LP. The pilot application of MeLoISE showed that students had very satisfactory performance and user experience.

Implications for practice and/or policy

LP teaching and learning can take advantage of MeLoISE to visualize a problem space and let users interact with its contents. Academic curricula can further explore the use of VWs for teaching LP, especially in introductory courses.

Introduction and background

Logic Programming (LP) is a declarative programming paradigm based on mathematical logic, in which knowledge is expressed as facts and inference rules. Logic programs encode objects and relationships among them in a particular problem domain. The Prolog language is the dominant implementation of the paradigm and is offered by most Computer Science academic curricula. LP teaching and learning methods and approaches rest on toy-world or micro-world examples that are often employed in many introductory courses (Yang & Joy, 2007). For example, the majority of examples contained in a popular book for teaching Prolog (Bratko, 2000) involve simplified worlds such as geometric objects, electric circuits, traveling in a network of connected cities, etc. The very nature of LP allows for an intuitive mapping between a logic program and an abstract or fictional world.

When students are first introduced to Prolog, they find it hard to create, understand and explain Prolog programs. Such difficulties are not exclusively grounded on the language's syntax and semantics, which are fundamentally different to those of imperative and procedural languages, but also on the fact that Prolog has a procedural execution model (Hong, 2004). Furthermore, observational studies of novice Prolog learners (Taylor, 1990) reveal that students can easily make logical mistakes when attempting to formulate Prolog clauses based on real world descriptions, because they tend to confuse the formal language of Prolog with
natural language constructs. This prohibits students from proceeding to and mastering more advanced topics and may lead to decreased motivation for learning. As a consequence, novice Prolog learners often need support to better construct and comprehend logic programs.

Similar issues of basic understanding have been effectively addressed in ‘traditional’ programming paradigms (e.g. procedural and object-oriented) through visual tools that help students grasp novel programming concepts and increase their motivation. Some of these tools, such as Greenfoot (Kölling, 2010) and Program your Robot (Kazimoglu, et al. 2012), enable learners to proactively program the behaviour of various actors, introducing interactivity in the learning process. Others, like Alice (Cooper, et al. 2000) and Meadow (Anderson and McLoughlin, 2007), present students with 3D game-like environments aiming to further facilitate the learning process via familiar representations.

More recently, general-purpose Virtual World (VW) platforms such as Second Life and OpenSim have enabled the development of representationally-rich and highly-interactive teaching tools, in which students and instructors are embodied as avatars and cooperate towards learning goals. In some cases (Esteves, et al. 2011; Rico, et al. 2011) the learning sessions are carried out using the VW’s own scripting language, whilst in others the VW has been enhanced with specific tools and extensions targeting introductory programming, e.g. SLurtles (Girvan, et al. 2013). The introduction of such tools denotes a trend to employ, and benefit from, constructivist methods, including self-guided, problem-based and collaborative learning. Furthermore, the multi-user nature of VWs can support group or pair programming, which is promoted by concurrent thinking and practice of professional software development (Beck, 2000).

A number Prolog teaching and learning tools have been developed so far, but they lack the aforementioned characteristics found in VWs. Some tools focus on the systematic use of programming techniques, templates and structures for guided programming and error analysis (Mondshein, 2010, Hong, 2004), while others focus on the comprehension of the transition between the two paradigms by mapping declarative features to familiar imperative constructs (Kumar, 2002). In some cases, hands-on experimentation and testing is available, which seems to have a positive impact on the learning outcome (Dahl, et al. 2010). Prolog teaching tools are purely diagrammatical, offer limited interactivity, and do not support collaborative problem solving. Up to now, there are no VW-based approaches for learning LP.

We argue that VWs lend themselves directly to the task. Logic programs essentially consist of definitions of objects and relationships among them, and the simplified or fictional micro-worlds used in introductory courses can be visualized as 3D environments in an intuitive and straightforward fashion. With the added benefits of real-time interactivity and multi-user support, VW platforms can be enhanced to support LP teaching and learning.

In this paper we present and evaluate a VW-based approach for teaching introductory Prolog programming courses that makes use of the micro-world metaphor typically used in LP courses.
A representation metaphor for logic programs in Virtual Worlds

In our proposed paradigm, the problem world is visualized as a 3D environment which students and instructors are not only able to explore, but can construct new representations in it, modify the existing settings and also participate in the problem environment themselves through their avatars (Figure 1). In contrast to a typical LP teaching session, where the problem definition is static and students have to manually encode the problem in Prolog, in our approach the status of the environment is automatically translated to appropriate LP code, thus ensuring consistency between visualization and logical representation. Students and instructors can instantly modify or re-define the problem by manipulating the world’s contents and can work together towards solutions by adding their own code and employing several system-provided functions regarding spatial and other relations among objects in the environment.

Figure 1. Left: typical paradigm for learning LP. Right: the proposed learning paradigm.

The representation metaphor is based on the following principles: a) be generic, allowing for a wide variety of problems to be represented and b) afford a simple, unambiguous mapping to a logic program and vice versa. Based on these principles, we decided to convert most of the generic, visually recognizable aspects of a 3D world into respective LP constructs. The elements of the environment are assigned a unique name and declared as objects or avatars in the logic program depending on their type. Spatial and structural relations and the visual properties of the objects and avatars are also represented in the world description (Figure 2). The relations include direction (front of, left of, right of, behind), distance (near, far), placement (on, above, below, between), collision, avatar-object interactions (sitting, holding, wearing) and structure (part of). The visual properties supported are colour, size and shape type.
This mapping produces a knowledge base that describes the problem world either entirely or with the manual addition of a few extra rules for problem-specific representations. Simple micro-worlds such as blocks world, grid-based environments and graphs can easily be created in the VW and automatically mapped to logic descriptions. More importantly, real-world or fictional stories involving humans, places and objects can be also supported with a few additional rules. Such scenarios are truly challenging to develop in the traditional LP learning paradigm.

**Description of the learning platform**

MeLoISE (Meaningful Interpretations of Simulated Environments) is a platform that implements the proposed representation metaphor and can be used for collaborative learning and practicing LP while immersed in a VW. It consists of components (Figure 3) that enable the interconnection between a Prolog engine and multiple educational scenarios running in a shared 3D environment. In each scenario, users are asked to identify, understand and solve LP problems presented as constructed spaces, by writing and testing appropriate code in Prolog. Users can collaborate throughout the various stages of the learning activities by communicating, co-constructing the environment, sharing code, introducing their avatars in the problem scenario, and testing their solutions together.

The architecture of MeLoISE follows a three-tier client-server model based on the OpenSim platform. Users can join the VW using a Second Life-compatible browser that connects to the OpenSim server. The VW includes one or more Interface Units, which are special objects that define the range and contents of each educational scenario. Each Interface Unit connects to a Knowledge Base server to submit the current status of the environment and to receive replies to user queries. The Knowledge Base server stores a simplified
geometric representation of each educational scenario, generates the appropriate Prolog facts and processes queries using an embedded Prolog engine. A detailed presentation of the architecture of MeLoISe can be found in Vosinakis et al. (2014).

![Diagram of MeLoISe architecture](image)

*Figure 3. The MeLoISe architecture.*

We illustrate the usage and functionality of the implemented environment with an example problem: Assume you are looking at a cafeteria, which includes chairs, tables, coffee cups and customers. All chairs are arranged around tables, some customers are sitting on chairs, and there are coffee cups on some tables. The goal is to write a logic program that can infer whether two customers are having a coffee together, i.e. at least two people are sitting at the same table and at least one coffee cup is placed on it and return the names of any such couple.

The preparation of an educational scenario with MeLoISe includes the definition, design and construction of a Scenario Space based on the logic problem. A Scenario Space is a segment of the VW that visually represents the micro-world and can be generated by placing the Interface Unit, in the VW along with the 3D entities (objects) of the given problem. In the selected example, the instructor uploads the models of a chair, a table and a coffee cup. Each of these three objects is assigned a name, e.g. ‘c’, ‘t’ and ‘cup’ and a description written in a special format, which is ‘$chair’, ‘$table’ and ‘$coffee’ respectively. This description is used by MeLoISe to automatically insert facts denoting each individual object’s type. The instructor or the students can duplicate these objects and build instances of the cafeteria world in each Scenario Space by arranging chairs near the
tables they belong to, and by placing coffee cups on some of the tables. They may also add further decorative elements (e.g. floor, walls, paintings) to make the environment more appealing (Figure 4a).

Each Scenario Space is automatically registered in the knowledge base and appropriate facts are generated. By default, all avatars and objects bearing a type description (the chairs, tables and cups in our example) are included as Prolog constructs. Despite having the same name in the 3D environment, each registered entity is assigned a unique name in Prolog. E.g. the chairs’ names will become ‘c’, ‘c_1’, ‘c_2’, etc. Thus, all collaborating students participating in the same scenario space will be assigned a name too, based on their avatar’s name, e.g. ‘john_smith’. If a student sends the command ‘kb registered’ to the interface object, the assigned names of all objects and avatars become visible in the 3D environment (Figure 4b).

The facts generated automatically by MeLoISE cover almost every aspect of the problem world description of our example. The chairs, tables, coffee cups and customers that participate in our environment have been declared with facts such as the following, which have been generated for all participating entities and describe their type:

- chair(c).
- table(t).
- coffee(cup).
- avatar(john_smith).

Furthermore, the embedded spatial relation on(X, Y) can be used to represent whether a coffee cup is on a table or not. E.g. if the object cup1 is on the table t_2 in the VW, the clause on(cup1, t_2) will be held true in Prolog. The only missing relation is the ‘belongs’ relation, which determines whether a chair belongs to a table. However, this can be easily derived by adding a rule that takes advantage of the embedded spatial relationship near(X, Y):

- belongs(Chair, Table) :- chair(Chair), table(Table), near(Table, Chair).

Students can add and test similar rules as a first step towards solving the problem. To add their own code, they have to open and edit a shared file included in the Interface Unit. When the file changes, the new code is publicly displayed in a shared board for the whole group to see (Figure 4c). Then, they can issue queries to the Prolog engine in a way similar to Prolog’s development environment. E.g. they could write: ‘kb query belongs(X, Y)’ to test their rule. If their rule is correct, the system will reply with the appropriate ‘belongs’ relations between chairs and table. The reply is sent as a text message in the public chat (Figure 4d). Students can, then, reflect on the reply and decide if their rule is correct or needs modifications.

Finally, students have to formulate a second, more complex rule to solve the problem. Again, they can work together on their code and test it. An appropriate solution for this example would be the following:

- having_a_coffee(Person1, Person2) :- sitting_on(Person1, Chair1), sitting_on(Person2, Chair2), Person1 \= Person2, belongs(Chair1, Table), belongs(Chair2, Table),
coffee(Coffee), on(Coffee, Table).

Students can test their code against several variations of the cafeteria instance, by having their avatars sit on chairs or by moving coffee cups between tables. If they detect mismatches between Prolog’s reply to their queries and expected results, they can re-work on their code and possibly experiment with different arrangements of the environment towards understanding why their rule has been misinterpreted.

Figure 4: Snapshots of the cafeteria scenario. a) problem world configuration, b) registered names of objects, c) code board, d) a user query and Prolog’s reply.

Pilot study and assessment

Setting and participants

MeLoISE has been evaluated in two studies that took place within a Masters (MSc) Logic Programming course at the Informatics Department, Piraeus University, Greece. The first study took place in January 2014 and involved 14 students in four groups; the results are documented in (Vosinakis et al, 2014). The second study replicated the first; it took place in December 2014 and involved 23 students in six groups. All students were familiar with Prolog, but only some (11 out of 37) had previous experience with VWs; to balance group skills, those students were evenly distributed among groups. In this paper we present the results of both studies treated as one, because the conditions between the studies were identical: a mirror of the environment was used; the
method was identical; all students were attending the same course, the study occurred at the same time interval within course evolution with the same instructor; they were carried out in the same computer lab. All 37 students (age: 25.75; st.dev.: 2.71) present an intended user group for the MeLoISE platform.

Measures

The measures for this study were group performance and collaborative user experience. Group performance was measured by the instructor of the course (scale [0,10]) in terms of the quality of code listings delivered at the end of the scenarios. We consider group performance satisfactory, if it is above 65%; fair, if within [40%-65%]; and not satisfactory if below 40%.

Collaborative user experience was measured with self-reporting measures, based on questionnaire responses. The questionnaire was self-developed and included fifteen questions in total - five for each dimension of our study: perceived learning effectiveness (specific to LP), collaboration and user experience. The items about collaboration and user experience are concerned with issues that are ubiquitous in respective work; for example, regarding user experience, Bargas-Avila and Hornbæk (2011) in their review have referred to all identified aspects (usability, usefulness, presence, engagement, visual appeal). All statements (illustrated in Figures 8, 9, 10) were rated with a five-point Likert scale (strongly disagree (1) to strongly agree (5)). Overall, we have used descriptive statistics to outline a qualitative picture of student performance and experience.

Procedure

The procedure included:

- Tutorial (30’) about the basics of interacting in the VW.
- MeLoISE tutorial (45’) about the user interface and functionality with simple examples.
- First problem scenario (45’): ‘Cafeteria’. The aim was to answer the queries: (a) if a chair belongs to a table; (b) if a table is taken; (c) if two users are having a drink together.
- Second problem scenario (45’): ‘Object recognition’ (Figure 5). The aim was to recognize composite objects (made up with primitive shapes): (a) if three spheres are at a pile; (b) if six primitive objects compose a snowman. This was a slightly more difficult scenario.
- Questionnaire and wrap-up (15’).
Data collection and analysis

The data collection methods and analysis included:

- Observation of student actions occurred in the computer lab (by the instructor of the course) and the VW (by the other two researchers). Notes and verbal comments were recorded on the fly, since that multiple groups were observed simultaneously.

- Concurrent probing. We posed questions to groups when they seemed to slow down or stuck to help them find their direction to a solution.

- Questionnaire responses. These were analysed according to guidelines in user experience studies (Albert and Tullis, 2013) including grouping responses into the top-2 box (positive responses) and bottom-2 box (negative responses) (Figures 7, 8, 9). For each question users could optionally write comments to justify their ratings (Tables 1, 2, 3).

- Log files and code listings. Respective data were recorded in: (a) chat logs (student communications and commands to the KB), (b) Prolog code, (c) server logs.

Results

Group performance

Satisfactory performance (i.e. >65%) was identified in half of groups for both scenarios (Figure 6), while most groups that had satisfactory performance performed really well: in 8/10 instances they performed over 80% (Figure 7). For the first scenario, one group performed fairly and four not satisfactory; this was reversed for the second scenario. Furthermore, 7/10 groups improved their performance in the second scenario. The average group performance was 57% for the first problem scenario and 67% for the second. Despite the second scenario was slightly more difficult, students improved their learning performance.
Given that group performance improved over time and that all groups improved their performance in the second scenario, we find these results encouraging. During the wrap-up, a considerable number of students reported that they would require more time to absorb the basics of this course and to get familiar with the environment. After reviewing their logs, we saw that they indeed came up with several naïve errors (like syntax errors when inserting or querying to the KB) at the beginning that held them off for some time.

![Figure 5. Overall group performance considered as satisfactory, fair or not satisfactory.](image)

Collaborative learning experience

In terms of perceived learning effectiveness, the large majority of participants were at the top-2 box (Figure 8). Users reported on several aspects of the system (MeLoISE, the setup, OpenSim), which contributed to learning effectiveness (Table 1); the ‘setting’ of the learning activity contributed most (19%). In this respect, a
user reported that “some facts about the world were there... this was very useful for me to make queries and understand them (in relation to the setup of the place)”, while another said that “you can visually see the results of the code you're writing. This makes you understand the code.” In addition, some users could not easily distinguish between elements of the whole environment (Everything: 11.9%) saying that “the environment gave me an idea of the programming of an open world”, and “I would prefer this environment than SWI Prolog”. Regarding interactivity, only a few users (7.1%) referred to this explicitly, however most other elements of the system are themselves interactive.

Figure 7. Upper/bottom-2 boxes of user responses about (perceived) learning effectiveness.

<table>
<thead>
<tr>
<th>Learning Effectiveness</th>
<th>Number of Users</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>I understood the given problem scenario(s)</td>
<td>32</td>
<td>19.0%</td>
</tr>
<tr>
<td>I was able to identify the necessary knowledge</td>
<td>31</td>
<td>14.3%</td>
</tr>
<tr>
<td>I was able to insert knowledge to the KB</td>
<td>34</td>
<td>11.9%</td>
</tr>
<tr>
<td>I was able to assess the validity of KB contents</td>
<td>27</td>
<td>9.5%</td>
</tr>
<tr>
<td>I was able to debug the KB</td>
<td>23</td>
<td>4.8%</td>
</tr>
</tbody>
</table>

Table 1. Aspects of the system that contributed to learning effectiveness.

In terms of collaboration, again the large majority of participants were at the top-2 box (Figure 9). Several aspects of the system contributed to collaboration (Table 2), with most notable that of the chat (36.4%) and the
code board (18.2%), i.e. a surface on which the user-submitted code was displayed in-world. The chat was used for issuing commands to the MeLoISE system and getting responses, and users reported on its value in terms of “allowing talking/whispering”, “I could see others talking, as well as the code responses while working”. The code board was referred by many users, especially because it enabled work in parallel: “the code board enabled one of us to write the code, while the others were collaborating and exploring”.

![Figure 8. Upper/bottom-2 boxes of user responses about collaboration.](image)

<table>
<thead>
<tr>
<th>Collaboration</th>
<th>Number of Users</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chat</td>
<td>20</td>
<td>36.4%</td>
</tr>
<tr>
<td>Code board (in-world code display)</td>
<td>10</td>
<td>18.2%</td>
</tr>
<tr>
<td>Avatars</td>
<td>9</td>
<td>16.4%</td>
</tr>
<tr>
<td>Mini map</td>
<td>7</td>
<td>12.7%</td>
</tr>
<tr>
<td>Other</td>
<td>4</td>
<td>7.3%</td>
</tr>
<tr>
<td>Voice communication</td>
<td>3</td>
<td>5.5%</td>
</tr>
<tr>
<td>&quot;The UI&quot;</td>
<td>2</td>
<td>3.6%</td>
</tr>
<tr>
<td>All aspects mentioned</td>
<td>55</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 2. Aspects of the system that contributed to collaboration.

In terms of the user experience, again the large majority of participants were at the top-2 box (positive responses) (Figure 10). Users highlighted (Table 3) the interactivity of the environment (22%) and avatar representations (19%). Users reported that “it was a motivating setup and environment”, as well as “most of our (team's) difficulties were about the syntax of Prolog, not the environment.”
Negative aspects of the user experience were reported about the graphics of the environment (mediocre graphics: 12.2%): “when the avatar sits, sometimes it doesn’t look natural”, “the models were low-poly but they had quite nice animations”, “although we are used in better quality of graphics, it serves educational purposed well.” Also, a few “slow loading times” were reported, mainly at the beginning of the session, when all users struggled to login to the environment, and when they massively moved from the areas of 1st scenario to that of the 2nd.

**Discussion and conclusions**

We argue that MeLoISE provides affordances for constructivist approaches to LP teaching. It supports social interaction and collaboration, as students are immersed in a shared problem space, and can work together towards a solution. Furthermore, it enables free experimentation and reflection by exploring and manipulating a concrete problem space, changing initial conditions and comparing Prolog’s replies with expected results. Finally, it allows for more active involvement in the learning process, where experienced students can design
their own problem environments in the VW and learn through creative experimentation. These characteristics are valuable for applying instructional methods based on constructivism, like problem-based learning (Vosinakis and Koutsabasis, 2012) or constructionism (Girvan et al., 2013), which have already been used successfully in VWs.

Almost all students participating in the study found the system useful for teaching and learning Prolog. This is consistent with the results of a previous study on Prolog teaching strategies (Yang & Joy, 2007), in which the use of concrete, real-world paradigms is one of the two most popular approaches for both learners and instructors. Previous investigations of automated environments for LP teaching and learning have relied on visualizations of knowledge, syntactical elements and inference procedures (e.g. Mondshein et al. 2010; Hendriks et al. 2010). However, such visual elements are of rather technical nature (graphs, box diagrams, etc.); therefore they present the structure and execution of the program rather than a concrete representation of its meaning. As Naps et al. (2002) argue, visual programming environments are more effective if they let students interact with programming constructs and create their own programs by manipulating visual objects. This kind of interactivity is afforded by MeLoISE, since the logic program is partially constructed from the objects that comprise the scenario space.

MeLoISE presents an innovative approach and system for teaching LP in VWs based on a dedicated implementation interfacing the OpenSim environment with a Prolog engine. The approach exploits the metaphor of micro-worlds which is typical in LP teaching as well as the multi-user, entertaining nature of VWs. Based on the pilot study results, the platform was accepted enthusiastically by students who responded positively to all dimensions of the collaborative user experience. Regarding performance, student groups performed well to the given problems, while their performance improved over time. Constraints of our evaluation include that we looked into group performance only (and not atomic performance) and we determined learning effectiveness largely on participant perceptions rather than a more systematic approach (that would include a post-test).

Further investigation of learning effects will be pursued with studies that compare the traditional approach to LP learning with the MeLoISE platform in terms of both individual and group performance. Further work also includes: adaptations for K-12 education about introductory logic programming; investigations of learning effectiveness between students of different levels of programming competencies; more extensive learning exercises or projects in which students both construct problem scenarios and write the appropriate code to answer queries; automatic assessment of student solutions.

**Statements on open data, ethics and conflict of interest**

The data of this research can be accessed under the ODC-BY 1.0 license after contacting the corresponding author. According to the ODC-BY 1.0 you are free to share, create or adapt the data as long as you attribute its use by citing this manuscript. This research complies with the ethical guidelines for educational research proposed by BERA (British Education Research Association). The authors have no affiliations with or
involvement in any organization or entity with any financial interest in the subject or materials discussed in this manuscript.

References


