Using Second Life in Programming's Communities of Practice

Micaela Esteves[1], Ricardo Antunes[1], Benjamim Fonseca[2], Leonel Morgado[3], and Paulo Martins[3]

[1] Polytechnic Institute of Leiria, Ap. 4163, 2411-901, Leiria, Portugal
[2] UTAD/CITAB, Ap. 1013, 5001-801, Vila Real, Portugal
[3] UTAD/GECAD, Ap. 1013, 5001-801, Vila Real, Portugal
{micaela,antunes}@estg.ipleira.pt,
{benjaf,leonelm,pmartins}@utad.pt

Abstract. This paper presents a novel approach to teaching and learning computer programming, using the three-dimensional virtual world Second Life® to develop a programming community of practice. Our students have developed their programming projects as part of this community as an alternative way of learning. The learning of programming is a difficult process, with many students experiencing difficulties which result in high levels of failure in introductory programming courses. In this paper, we describe and analyse how this approach spurred students' motivation and interest in learning programming. We also present observations on the difficulties felt by both students and teachers in the development of projects and activities, and discuss the approaches taken to overcome those difficulties.

Keywords: Communities of practice, Collaboration, Programming learning, Virtual worlds, Second Life.

1 Introduction

Learning how to program a computer is a hard task, and a diversified set of skills must be learned for one to become a good programmer. Typically, when students initiate the study of computer programming, they usually come across several difficulties, which are then reflected in highs levels of failure in entry-level courses (commonly called "Computer Science 1/2" or "Computer Programming 1/2"). Several research efforts have sought to find the causes of this failure (*e.g.*, [1], [2]). Amongst the reasons pointed out by research are: lack of contextualization of the learning process [4]; the nature of traditional teaching method, based on lectures and specific programming language syntaxes [3], and difficulties in understanding the basic concepts of programming, such as variables, data types or memory addresses [2-3], described as abstract concepts without an equivalent representation in real life.

Compounded with these factors, we have a new generation of computer science students for whom computers have been a constant presence in their lives, an important tool, but don't feel themselves motivated to learn computer programming [5].

They often don't understand why they should write code, since there is a world of complexity to be mastered just by combining applications and settings, by fiddling with configuration files and formats. Also, typical computer environments and applications students employ as users are of a visual complexity and appeal far beyond what students typically achieve on entry-level programming courses, a factor that does not support self-motivation. On the other hand, the stereotype of a programming student as someone that is alone, programming all night long, without social contact, contributes to hinder student's personal view of programming subjects and even shed aside possible future careers related with it [5].

All the aspects mentioned above make the students feel and experience some disorientation and lose interest in learning. Although students belong to a community – the academic community – they "learn lonely and alone are tested" [4]. The vision of having the students all connected as a network node, each contributing to another's learning while building personal knowledge[4], drove us to create a programming community of practice in Second Life. The practical applications of the acquired knowledge in the community, its reflection and exchange are some of the strategies suggested by Fleury [6] and Dillenbourg [7].

In this paper, we present the result of two years of observations using Second Life as a platform for teaching and learning computer programming, with the purpose of identifying practical issues that teachers and students face in such approach, and ways to overcome those issues. It was not our goal to compare this approach to others, since we believe that any such evaluation depends on the educational methods and practices; and that the establishment of methods of practices requires educational practitioners to be aware of the practical issues that may hinder or disrupt the educational process. In the following section, we give an overview of the concept of communities of practice, and in the second section, we present the research activities and analyse the results (identified issues and approaches to overcome then). Finally, we present some conclusions based on reflection upon the results.

2 Communities of Practice

"Communities of practice are everywhere and we are generally involved in a number of them – whether that is at work, school, or in civic and leisure interests. In some groups we are core members, in other we are more at the margins", [9]. Within these communities to which we belong, people share a common interest and join each other in its pursuit, developing and learning practices and world-views in the process. The practices may reflect activities, but also social relations.

These communities may have a formal or informal organization (formal communities of practice being those with regular meetings with predefined work, informal ones all others, including those that may not even see themselves as a community). Typically, communities are organized around some particular area of knowledge / activity that provide members a sense of joint enterprise and identity [9].

As stated by Wenger [8], there are three elements involved in defining a community of practices (CP). One is the domain: the community must have a subject to talk about. The second is a community of people that interact and thus facilitate the development of relationships regarding the domain. A Web page is not a CP, or if there are seventy managers that never talk with each other, they are not a CP, even if they have the same functions. There must be a community of people, a sharing and construction of knowledge. The third element is the set of practices (the “practice”): the community must have a practice and not just a common interest that people share. They learn together how to do the things they do (or want to do). And that learning involves participation in the community. A participation that refers not just to local events of engagement in certain activities with certain people, but to a more encompassing process of being an active participant in the practices of social communities and constructing identities in relation to these communities [8].

According to Wenger [8], a community of practice is a good way to promote learning and good practices, not only because it develops knowledge in a living and experimental way, but also because it helps participants reach solutions to possible problems, with significant connections leading individuals to higher creative levels than they could reach on their own [9]. A typical community is made up by different levels of participation: central, active and peripheral. Initially, people join communities and learn at the periphery. As they become more competent they move more to the “centre” of the community. According to Wenger [8], in order for a community of practice to be successful it needs to motivate the participants’ involvement at the different levels, establishing the dialogue between the internal and external perspectives of the community. The participation of external elements in a community is extremely useful for the development of practices in that community as well as the integration of the community itself in other groups.

3 Developed Activity

The main objective of this study is to find out if and how could SL be used as a platform for teaching / learning the imperative programming language paradigm that is commonly taught in college level computer science courses. For this purpose, we have employed action research methodology. For this purpose, we create a community of practice for teaching and learning computer programming in the Second Life virtual world (SL), we provided to students elective alternative assignments on some compulsory college-level subjects. This took place at two Portuguese Higher Education institutions: the University of Trás-os-Montes e Alto Douro (UTAD) and the Higher School of Engineering and Management of the Polytechnic Institute of Leiria (ESTG). The subjects’ main aim is to allowing students to develop semester-long projects, to improve programming skills.

3.1 Methodology

In this study we have employed action research (AR) methodology, a cyclical process approach that incorporates the four-step processes of planning, action, observing and reflecting on results generated from a particular project or body of work [12].

The first action research cycle started by planning a model for teaching introductory programming concepts in SL. To that end, it was necessary to make a pre-exploratory research and pre-observation with the goal of identifying problems and planning actions [13]. Initial plans were formulated, and actions for their prosecution were devised and implemented. While the action (teaching-learning) took place, results were monitored for reflecting later on.

The data collected for the reflection step of the action research methodology was based on daily session reports, classroom images and questionnaires. The reports, written down by the teacher-researcher at the end of each session, describe what happened during the class, indicating all the critical incidents and its implications. Classroom images (screenshots) have been taken in order to assist the teacher to review the lesson when necessary, such as when a critical incident had happened. Questionnaires with open questions concerning the learning / teaching method were presented to students at the beginning, middle and end of the process, to provide further information on the learning process. These elements are used as a tool to adjust and improve the learning / teaching approach.

The final step in the first research cycle was reflection upon the outcomes and based on them planning the next cycle. This goes on until the reflection of a cycle showed that the problem was then solved or level of knowledge achieved is fixed. At this point the study was concluded and a report was produced.

3.2 Programming Environment

The programming environment was SL itself, not any offline editor. SL is a persistent on-line 3D virtual world conceived by Philip Rosedale in 1991 and is publicly available since 2003 [10]. It allows large numbers of users to connect, interact and collaborate simultaneously at the same time and in the same (virtual) space. Figure 1 shows a typical programming session in this research: we can see 6 avatars on black rugs (students programming) and two other - teachers' avatars.

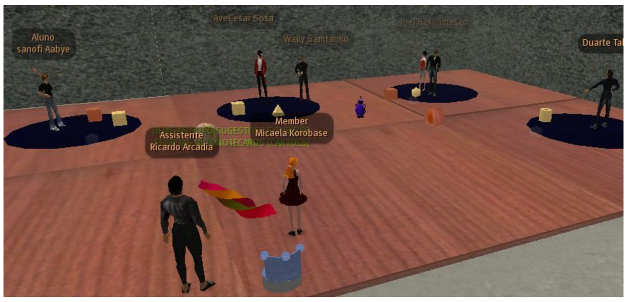
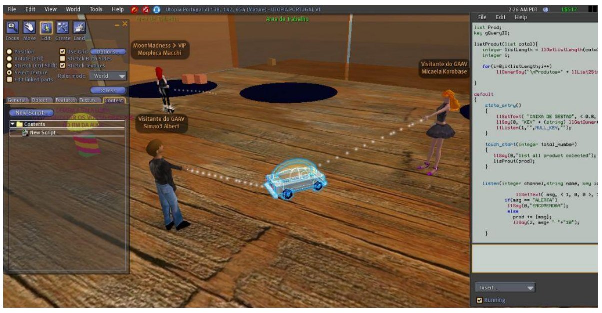


Fig. 1. Typical programming session

SL programming is currently done with a scripting language named *Linden Scripting Language* (LSL), which has C-style syntax and keywords. 3D objects created in SL can receive several scripts that are executed concurrently. Each script has its own state machine: program flow is sequential, using common methods from imperative programming, such as procedures and flow-control primitives, and structured in the traditional way, via function definition and function calls, but also by triggering events and responding to them (events can be raised either by environment interactions such as object collision or programmatic components such as requesting a server-based service). The programmer defines the states of each state-machine and explicitly specifies when to switch state. The language's programming libraries include functions and events both for SL-based results programming and for communication with external servers: sending and receiving e-mail, accepting XML remote procedure calls, and handling HTTP requests and responses.

SL enables synchronous collaboration among students because the system permits two or more avatars to edit the same object and include their own scripts, which act concurrently on the object (and may exchange messages). Also, it is possible to share scripts, so that students can access and edit the same piece of code while programming it. By default, only the creator of an object or script has full access to it. Thus, to share an object or script it's necessary for the creator of the object to explicitly set its permissions adequately. Figure 2 presents two avatars editing the same object (a car): the left window shows the car's contents, one being a script that is opened by double-clicking.



```
list Prod;
key gQueryID;

listProdut(list catal){
   integer listLength = llGetListLength(cata
   integer i;

   for(i=0;i<listLength;i++)
        llOwnerSay("\nProdutos=" + llList2St
}

default
{
    state_entry()
    {
        llSetText( "CAIXA DE GESTAO", < 0.0,
        llSay(0, "KEY" + (string) llGetOwner
        llListen(1,"",NULL_KEY,"");
    }

    touch_start(integer total_number)
    {
       llSay(0,"list all product colected");
       lisProut(prod);
    }


   listen(integer channel,string name, key i

                llSetText( msg, < 1, 0, 0 >,
            if(msg == "ALERTA")
             llSay(0,"ENCOMENDAR");
             else
               prod += [msg];
               llSay(2, msg+ " "+"10");

    }
```

Fig. 2. Two avatars sharing an object

One particular aspect of script sharing is that although several avatars may read and change it, saving the script overwrites the current version. Initially, this is not really a problem, because scripts are shared by the teacher and there is only a student editing the script. But as the community evolves, students are able to contribute more often and in larger numbers, and so coordination among participants is required. Chat channels can be used to coordinate who is accessing and changing the script.

Asynchronous collaboration is also supported because the SL world is persistent. Students and teachers may access and leave in-world objects and messages to the other members (group messages and privates messages are supported). When a user logs in all his/her messages are shown, and he/she can see any objects left in the world by others (and edit them, if adequate permissions have been set).

3.3 Community Structure

Teachers were the community coordinators, so they defined the projects to be developed, encouraged and motivated the periphery students through the exchange of opinions between members of the community, as well as sharing experiences of active participants that were once in periphery.

The community had meetings about two hours per week in Second Life (SL), where they developed their programming work and kept track of community's progress, exchanged ideas and made suggestions. Face-to-face meeting took place only once a month, because the teachers were in Leiria and the students in Vila Real, 270 km apart: once a month there was a meeting at Vila Real to discuss the projects and the details of on-line cooperation.

The first students that participated in the community (2nd term of 2006/2007), 50% had little experience in both programming and SL, so they were in periphery level. The remaining 50% already had some experience in programming, although this was their first contact with SL, and thus while at the periphery of the "programming in SL" community they are already active members of our community if seen just from a programming perspective. The teacher's task was to motivate students at periphery to reach the active stage [10]. At the beginning of the subsequent term, 80% of students were at the periphery and the others within the active level.

3.4 Analysis of Results

It is possible to distinguish two phases in these activities: the first consisted in building objects with the modelling tools of SL, and is devoid of programming (robots, trains and dogs); and the second one consisted in the development of programs in LSL, to provide behaviours to the objects created previously.

During the first phase there weren't significant disparities between students at different levels of participation, the difficulties felt by both students were identical. For example: how to link objects with each other, how to make a copy or to line up objects. This is consistent with the previously-mentioned fact that while some students were at the periphery and others within the active level, regarding programming expertise, all began at the periphery regarding SL use.

In the second phase, some differences were observed among students. Students from the active level didn't have great difficulties in understanding how LSL works. Although they had already worked with event-based programming in other courses, these students weren't familiar with the concept of state machines or their programming. The major difficulty they faced consisted in selecting which library functions and events to use, and how to use them to implement specific functionalities. The teachers guided them, by showing alternative ways of creating identical object behaviours, so that they could ponder which would be more adequate.

Students at the periphery, programming-wise, weren't used to self study or autonomous computer programming development, so closer guidance from the teachers was needed. It began with simple examples that students would experiment with and modify. Whenever they had difficulties in understanding the examples, some explanations were provided for that specific part of the code. This way, students could understand what these small programs could do and the goal of each one. Based on the personal experience of the programming teachers and the scientific literature in this field, it is known that this level of understanding is difficult to reach when students are learning to program using traditional environments, such as C command-line compilers, where the students generally feel great difficulty in understanding the programming objective [2].

A particular important aspect in programming learning is the students' reaction to compilation errors [11], which are inevitable in the learning process. Students from active level corrected the compilation errors whenever they happened without the teachers' help, whereas the periphery students found themselves without knowing the reason why they occurred or how to correct them. When students had some difficulty about the code they had implemented, they shared it with the teachers, so they could observe and at the same time find out what was wrong and follow the teachers' indications/instructions. This way, the students corrected the code and went on.

Execution errors occurred more often with active-level students. These students tested more programs by their own initiative and noticed more frequently that these didn't execute as they expected. It was observed that the students were not less motivated because of this; on the contrary, they corrected the programs and tested them until they behaved has they wanted them to.

One of the projects set forth by teachers to the community consisted only in data manipulation and very little graphic interaction (one of SL's differentiating factors). This resulted in difficulty for teachers to motivate periphery students to strive to reach the active level. In order to go overcome this, community leaders had to involve active students from the previous semester and external elements, in order to motivate and increase the activity inside the community. In order to assess the work done by students, it was observed that it was difficult to manage the attribution by students of access privileges for teachers to their scripts, leading to situations that rendered assessment impossible without contacting the student and requesting correction of wrong privileges (for example, when a student would send the teacher an object without conferring the necessary permissions to access the scripts).

One of the difficulties felt in the community development was the lack of a common space for impromptu presentation of schematic ideas and reflections: a "blackboard" as it was. Another issue we came across was the absence of a mechanism that would inform the teachers, by email or some other non-SL system, what the students had achieved throughout the week, i.e., what was reached, what had caused more delays, which difficulties had been felt, and which attempts had been made to try and overcome them.

4 Conclusions

In this paper we presented a study that has been conducted using the action research methodology. In this study we created a programming community with the aim to explore the viability of using SL as platform for teaching and learning a computer programming language.

This study is not finished yet but we can conclude that:

- SL has characteristics that make it a platform suitable for teaching / learning a computer programming language but it is necessary to use it in association with another platform where the teacher can supply students with teaching materials.
- Students learning how to program by programming physical interactions in SL (e.g., making a dog follow you and obey your voice command) are typically motivated. Students who focused primarily on non-visible techniques such as data structures and string processing, benefiting from the environment just for enhanced context and not as a source of feedback for programming behavior, did not seem to exhibit any motivational advantage over students who employ a

traditional console-oriented (text-only) approach. Thus, teachers must pay special attention when conceiving students' assignments, particularly if the students are novice in programming because they need projects that stimulate their imagination.

References

1. Gray, W.D., Goldberg, N.C., Byrnes, S.A.: Novices and programming: Merely a difficult subject (why?) or a means to mastering metacognitive skills? Review of the book Studying the Novice Programmer. Journal of Educational Research on Computers, 131–140 (1993)
2. Miliszewska, I., Tan, G.: Befriending Computer Programming: A Proposed Approach to Teaching Introductory Programming. Journal of Issues in Informing Science & Information Technology 4, 277–289 (2007)
3. Lahtinen, E., Mutka, K.A., Jarvinen, H.M.: A Study of the difficulties of novice programmers. In: Proceedings of the 10th annual SIGSCE conference on Innovation and technology in computer science education (ITICSE 2005), Monte da Caparica, Portugal, June 27-29, 2005, pp. 14–18. ACM Press, New York (2005)
4. Figueiredo, A.D., Afonso, A.P.: Managing Learning in Virtual Settings: the Role of Context. Information Science Publishing (2006)
5. Lethbridge, C., Diaz-Herrera, J., LeBlanc, Jr., Thompson, B.: mproving software practice through education: Challenges and future trends. In: Future of Software Engineering (FOSE apos;2007), pp. 12–28 (May 2007)
6. Fleury, M., Oliveira Junior, M.: Gestão do Conhecimento Estratégico – Integrando Aprendizagem, Conhecimento e Competências. Editora Atlas, São Paulo (2001)
7. Dillenbourg, P.: Learning In The New Millennium: Building New Education Strategies For Schools. In: Workshop on Virtual Learning Environments (2000), `http://tecfa.unige.ch/tecfa/publicat/dil-papers-2/Dil.7.5.18.pdf`
8. Wenger, E.C., Snyder, W.M., McDermott, R.: Cultivating communities of practice: a practitioner's guide to building knowledge organizations. Harvard Business School Press Book (2002)
9. Lave, J., Wenger, E.: Situated Learning: Legitimate Peripheral Participation. Cambridge University Press, Cambridge (1991)
10. Esteves, M., Antunes, R., Morgado, L., Martins, P., Fonseca, B.: Contextualização da aprendizagem da Programação: Estudo Exploratório no Second Life. In: Proceedings of IADIS Ibero-Americana WWW/Internet 2007, Vila Real, Portugal, Outubro, 7–8 (2007)
11. Esteves, M., Mendes, A.: A Simulation Tool to Help Learning of Object Oriented Programming Basics. In: Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference, Savannah, Georgia, USA, October, 20–23 (2004)
12. Lessard-hébert, M., Goyette, G., Boutin, G.: Investigação Qualitativa: Fundamentos e Práticas, Lisboa, Instituto Piaget (1994)
13. Dick, B.: A beginner's guide to action research (2008), `http://www.scu.edu.au/schools/gcm/ar/arp/guide.html`