Journal of Virtual Worlds Research

Volume 2, Number 1.

Pedagogy, Education and Innovation in 3-D Virtual Worlds.

ISSN 1941-8477

Vol. 2. No.1
"Pedagogy, Education and Innovation in 3-D Virtual Worlds"
April 2009

This issue was sponsored, in part, by the Singapore Internet Research Centre,
the Department of Radio, TV & Film at the University of Texas at Austin,
and the Texas Digital Library Consortium.

The Journal of Virtual Worlds Research is owned and published by the
Virtual Worlds Research Consortium,
a Texas non-profit corporation.
(http://vwrc.org)

Journal of Virtual Worlds Research
jvwresearch.org ISSN: 1941-8477

Vol. 2. No.1
ISSN: 1941-8477
"Pedagogy, Education and Innovation in 3-D Virtual Worlds"
April 2009

Using Second Life for Problem Based Learning in Computer Science Programming

By Micaela Esteves, Polytechnic Institute of Leiria, Portugal;
Benjamim Fonseca, Leonel Morgado, Paulo Martins ,University of Trás-os-Montes e Alto Douro, Portugal.

Abstract

A large number of students fail when beginning the study of computer programming, and withdraw from courses because of the many difficulties they face while trying to grasp the basic concepts. Programming education is typically based on putting problem-solving skills to use, by identifying a problem, developing an algorithm to tackle it, and coding that algorithm with some programming language, whose syntax and semantics must be learned. Extant research has highlighted the challenges associated with learning/teaching a programming language. However, researchers are still struggling to provide effective guidance to practitioners in this field. We believe that a better understanding of the teaching/learning process in the virtual world Second Life is a potential avenue for using this environment in classes. In this experimental research, we observed and reflected upon the problems that came up and then presented and discussed the results. We conclude with implications for future research and for practicing teachers.

Keywords: action research; project-based learning; collaborative virtual environments; learning programming; context.

Using Second Life for Problem Based Learning in Computer Science Programming

By Micaela Esteves, Polytechnic Institute of Leiria, Portugal;
Benjamim Fonseca, Leonel Morgado, Paulo Martins ,University of Trás-os-Montes e Alto Douro, Portugal.

The emergence of new technologies, such as 3D virtual worlds (e.g., Second Life and Active World), brings new opportunities for teaching and learning. In addition, these virtual worlds reinforce the wider strategic drive in education towards more personalized learning tailored to the individual learner's needs (de Freitas & Yapp, 2005; West-Burnham, 2005) and greater learner autonomy (Field, 2007).

This paper outlines a novel approach for teaching and learning a computer programming language. It uses the specific context of Second Life (SL), which is a 3-dimensional online virtual world, in computer science (CS) undergraduate courses. To teach the introductory computer programming course, our approach replaces the traditional language and development environments by the SL environment. We did it using SL's internal scripting language, *Linden Scripting Language* (LSL*)*, as a way to research if SL could be used as a platform for teaching/learning a programming language. We wanted to perceive and learn how this process of teaching and learning occurs within SL. Therefore, some questions came to mind: What are the problems for both teachers and students in using SL? Can these problems be solved and how?

With this acquired knowledge, it is possible to do a subsequent qualitative and quantitative investigation in order to determine the following:

- If SL really does improve students' comprehension of basic programming concepts;
- Understand if the effects of some SL elements, such as the visual behaviour of the objects, improves the students' performance and comprehension;
- Understand if there are advantages in this methodology in relation to others.

This approach emerges from our concern about the difficulties that novice students face when they begin to study a programming language, a fact that has been well documented in the literature (Gray, Goldberg, & Byrnes, 1993; Jenkins, 2002; Esteves & Mendes, 2004), and we as computer science teachers have observed it in our students' everyday classes. As the SL environment is similar to the games that students play daily, we intend to improve the teaching/learning programming language and change the students' thoughts about programming.

With these objectives in mind, we employed an action research methodology to analyse the teaching and learning process within Second Life. This research was developed around the last 2.5 years, and the results are presented and discussed in this paper.

This paper is organized as follows: The following section establishes the background on the phenomenon of interest – using Second Life for teaching/learning computer programming language. We follow with the methodology and design of the study. Then, the results are presented and discussed. The paper concludes with suggestions for using SL and future research.

Related work and motivation

Learning how to program a computer is a difficult task. To program a computer, students must have a good knowledge of problem-solving skills, know the programming language syntax and semantics, as well as be able to understand existing code (Linn & Clancy, 1992). At a higher level, in order to program it is also necessary to be able analyse a problem, conceive a solution, express it in a computer language code, and be able to test it for errors and correct them.

Sloane & Linn (1988) declared that programming is not just a single skill but a hierarchy of skills and that the programmer will need many of them throughout her/his career. For Dijkstra (1989), learning to program is a slow and gradual process that requires a study based on practice, and is quite different from most of its supporting disciplines (which are based more on theory), involving a great deal of reading and memorization.

Many students experience difficulties when they begin studying programming. This results in high levels of failure in introductory programming courses, typically taught in the beginning of computer science studies. In the literature, several studies can be found concerning the causes of these failures (Milne & Rowe, 2002; Gomes, Areias, Henriques, & Mendes, 2008). One of the asserted causes is the students' difficulty in knowing how to design a solution to a problem, subdivide it into simpler codeable subcomponents, and conceive hypothetical error situations for testing and finding mistakes.

Another factor is lack of motivation: Students often claim they do not like programming because they experience difficulties in understanding even the most basic concepts (Lahtinen, Mutka, & Jarvinen, 2005; Miliszewska & Tan, 2007), such as variables, data types, or memory addresses, since these abstract concepts do not have direct analogies in real life (Lahtinen et al.; Miliszewska & Tan). However, the greatest difficulty students face is not only the understanding of basic concepts, but also realizing how to apply them correctly in more complex constructs. Although some of them comprehend the syntax and semantics of a programming language, they do not know how to use it correctly to create a program (Winslow, 1996; Lahtinen et al.). It is important to integrate within the learning process the knowledge of concepts and strategies to use for solving a problem.

Students who attend programming lectures in computer science programmes have different experience levels: Some may have had previous contact with one or more programming languages, while others may be contacting it for the first time (Milne & Rowe, 2002). This discrepancy in students' knowledge renders the teaching process more complex for traditional teaching approaches because classes are diverse in skill levels.

The most currently used teaching methods in programming classes are based on static materials to explain dynamic concepts with an over focus on language syntax (Sloane & Linn, 1988; Linn & Clancy, 1992; Lahtinen et al., 2005; Schulte & Bennedsen, 2006; Esteves, 2008 a). Possibly due to this, students often learn programming as being a traditional theoretical subject,

like history, based on reading rather than practicing. Therefore, they feel discouraged and do not get involved in learning, since they do not understand the concepts or achieve positive results in evaluations (Almeida et al., 2002). As Lethbridge (2007) noted, "In no other discipline is the gulf between best practice and typical practice so wide."

In addition to these difficulties, programming classes typically employ the most professional programming languages: C, C++, C#, and Java, which have extensive and complex syntaxes, rendering learning difficult for beginners (Motil & Epstein, 2000; Jenkins, 2002). Furthermore, certain issues require a higher degree of abstraction: Jenkins argues that the purpose of teaching programming is teaching students how to program and not teaching them a programming language. He also notes that the language used to teach should be one that was developed for the purpose of the LOGO language. The environment commonly used to teach programming is too complex, and better adapted to professionals and not for teaching (Gomes et al., 2007).

Since the 70s, the scientific community has studied and developed software with the purpose of helping students overcome these difficulties (Sloane & Linn, 1988). Despite this effort, the problems remain. Part of the problem is that computer programming has been introduced using programming languages that are difficult to use and with proposed activities that are not deeply connected to students' interests and passions (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008).

Recent technological developments have opened new opportunities for learning, especially the emergence of three-dimensional virtual worlds like Second Life, which is a world with a powerful visual impact that allows people to freely program behaviours into objects. As Papert (1980) noted, "A programming language is like a natural, human language in that it favours certain metaphors, images, and ways of thinking."

Making an abstract representation (mental model) into a more concrete one (external model) that facilitates problems solving has been demonstrated (Beveridge & Parkins, 1987). For example, Noyes and Garland's (2003) study of the Tower of Hanoi demonstrated that the use of an external, computer-based representation was beneficial to certain aspects of problem-solving performance in novices. Williams and Noyes (2007) concluded that computer-based representation has the potential to play a vital role in the development of problem-solving abilities, although it would be very appropriate at early stages of development.

Andries van Dam (2005) purports that visualization and visual tools in learning help students to better understand the concepts because physical, spatial, or visual representations are easier to retain and manipulate. Having an immediate display of the results of an action, students can find out at once if their idea is right or wrong. If it is wrong, they have to correct and rethink their solutions, and consequently this will stimulate their critical thinking (Kiili, 2006; Shneiderman, 1983). Using visualization techniques in the classroom can give students more options for exploring information – data, texts, or even virtual objects and spaces – and can help them gain a deeper understanding of concepts or better analyze conflicting or controversial information (Naps et al., 2003, Schweitzer & Brown, 2007; Amershi, Carenini, Conati, Mackworth, & Poole, 2008).

Some research has shown that simple visualization is not enough to support successful learning (Naps et al., 2003; Hundhausen & Brown, 2008), since students may look at dynamic visualizations without understanding the context or deeper meaning. Visualizations have been shown to be more effective if they engage the learner, rather than let the viewer watch passively (Pierson & Rodger, 1998; Naps et al. 2003; Schweitzer & Brown, 2007). For example, in an algorithm visualization, students benefited from constructing and presenting their own visualizations not only because this exercise increased their motivation and level of interest in the algorithm, but also because it stimulated meaningful discussions about them (Hundhausen & Brown).

Environments such as ALICE (Dann, 2000), JELIOT (Ben-Bassat Levy, 2003), BlueJ (Kölling Quig, Patterson, & Rosenberg, 2003), and RAPTOR (Carlisle, Wilson, Humphries, & Hadfield, 2005) have been used to teach imperative programming in undergraduate introductory computer science courses. All these environments generate concrete visual representations of a program. However, ALICE is a 3D interactive graphics programming environment for Windows (Cooper, Dann, & Pausch, 2000). It is also an object oriented by writing simple scripts in which its users can control their object's appearance and behaviour. The benefits of using it allows students to be involved and at the same time have the ability to develop an intuitive understanding of basic concepts in a visual feedback environment (Dann, Cooper, & Pausch, 2001).

In the literature review, collaborative programming has also been proposed as a way of improving students' learning. Research supports that collaboration is an effective pedagogical feature for introductory programming, and that pair programming, in particular, is appropriate for learning how to program (Guzdial et al., 1996; Menchaca, Balladares, Quintero, & Carreto, 2005).

Collaborative environments can offer important support to students in their activities for learning programming. According to Newman et al. (1989), collaboration in problem solving provides not only an appropriate activity but also promotes reflection, a mechanism that enhances the learning process. Students that work in groups need to communicate, argue, and give opinions to the other group members, encouraging the kind of reflection that leads to learning.

Although 3D virtual worlds are relatively new, they have already been used as pedagogical media (Dickey, 2003). Constructivist and constructionist learning approaches, in particular, may recognise the potential in these environments because they provide for educators an accessible means of creating a rich and compelling 3D context for situating learning and communicative tools to support discourse and collaboration.

These results influenced some of the opinions we had when we considered the use of the 3D virtual world SL as an environment for teaching/learning a programming language. Before we proposed utilization of SL to our students and colleagues, we felt the need to do a study with the aim of exploring the viability of this environment. In the next section, we will present the research that we developed.

Research methodology

The aim of this study is to determine whether SL can be used as a platform for teaching/learning imperative programming language and in what way. For that reason, we employed the action research methodology, since it is best for research interactive processes: It allows the study of a dynamic problem by introducing controlled changes, and is frequently used in the research of teaching processes (Lewun, 1946; Zuber-Skerritt, 2000).

The action research methodology is a cyclical process that incorporates four steps: **planning**, **action**, **observation,** and **reflection** upon the results. **Planning** defines how the process under research will take place, and the observation and context variables; **action** is the development of the process itself, and concurrently the researcher records the **observation** data; the observation results are **reflected** upon, and variables or context are changed according to that result. This completes one cycle of the action research process, and feeds a new phase of planning to initiate the next cycle. When there is not enough research literature on the field of study, as is the case with the use of multi-user virtual worlds to teach programming, it is necessary, before the first research cycle, to make a pre-exploratory experience in order to identify the basic problems and feed the first planning (Lessard-hébert, 1994). The research process is never completed, but a plateau is reached when the reflection at the end of a cycle deems that the amount of collected knowledge on the process is significant (Zuber-Skerritt, 2000).

Research questions

The long-term research question is "Does SL present conditions to be used as a platform for teaching and learning an imperative computer programming language?" Beyond the basic technical analysis, this question can be formulated in a different way: "What are the problems for both teachers and students in using SL? Can these problems be solved and how?"

To answer these questions, we examined two dimensions of this project – the learning experiences of the students and the teaching experiences of online teachers.

The pre-exploratory research took place during the second semester of the academic year 2006/2007, the first and second cycles of the action research during the first semester of the year 2007/2008, and the third and fourth cycles in the second semester of the same year (2007/2008).

This report presents in detail the data collected from the third and fourth cycles, and focuses on the analysis and discussion of the final results obtained through this research. It then briefly describes the previous cycles, which are derived from a study of using SL during 2 years.

Data collection

We used several data sources to facilitate triangulation and reduce bias in our analysis.

The data collected for the reflection step was based on daily session reports made by the teacher-researcher; classroom images that helped the teacher to remember the important key points; questionnaires with open-ended questions, at the beginning, middle, and end of the process, concerning the learning/teaching methods and the students' opinions about the project

they had developed and in learning in this environment. This entire source of information provided us further information on this learning process and the students' motivation. The primary source of data was the communication between teacher-researcher and students, which was recorded and saved for each session as an essential tool to help the teacher to remember the context of students' uncertainty and opinions.

Participants

Our participants were computer science students from the University of Trás-os-Montes e Alto Douro (UTAD), in Vila Real, Portugal, and from the Higher School of Technology and Management (ESTG – Portuguese-language acronym) of the Polytechnic Institute of Leiria, Portugal. These students took part in the research process, while developing elective alternative assignments on the following compulsory subjects: Laboratório de Informática I of the 1st curricular year (Lab I); Laboratório de Informática II (Lab II) and III (Lab III) of the 2nd curricular year (at UTAD); and Projecto I (at ESTG). In all cases, these subjects had as the main goal to allow students to develop a semester-long project in order to improve their programming skills. Other students of the same subjects were developing assignments using other programming languages and environments. In both cases, there were no prescribed lectures.

We had three different types of students: a group of beginners (Group A); a group with some knowledge of programming (Groups B1 and B2); and one other with experience in programming (Group C).

The students from Group A, at UTAD, were enrolled in the 2nd semester of the 1st curriculum year. Although they were at the initial stage of learning how to program, they had been exposed in the previous semester to its introductory aspects in about 30% of two subjects (see Table I). The project made in SL was the students' first contact with a programming project (not just a class exercise).

In Groups B1 and B2, the students were more advanced in learning how to program: Some were at the UTAD, where they were enrolled in the 2nd year, 1st semester; they had already studied introductory aspects of the C programming language in the previous semester and developed a command-line project in C, and while participating in this research, they were also taking a course on object-oriented programming in C++. Others were at the ESTG, where they were enrolled in a post-secondary technical course (CET) and had previously studied C programming (see Table I). In both cases, the project developed in SL was their first programming project (i.e., not just a class exercise).

In Group C, the students were from UTAD, enrolled in the 2nd year, 2nd semester, and thus at a more advanced stage of learning how to program: they had completed courses in C and C++ programming, and developed a command-line project in C++ (see Table I). Although these students still required the teacher's support, they had some autonomy in using and studying programming.

Table I
No. of students experienced in programming languages.

Group	Curriculum year	Number of students	Prog. Languages already studied	Prog. Languages being studied	School
A	1st	5		C	UTAD
B1	2nd	10	C	C++	UTAD
B2	Post-secondary	6	C	None	ESTG
C	2nd	4	C, C++	C#	UTAD
D	1st	9		C	UTAD

In this study, 5 students from Group A and 4 from Group C, both from UTAD, participated in the exploratory research. In the first and second action-research cycles, the students were only of Group B (10 students from UTAD and 6 from ESTG). In the third and fourth cycles, there were 9 students from Group D.

Materials and tasks

All the participants worked with Pentium IV computers running the Windows XP operating system, with 1 GB of RAM and somewhat limited bandwidth. We proposed a project for students to develop, i.e., we specified what kind of objects should be devised, including features and behaviour, and students had to construct their objects following the specifications.

The teacher had the challenge of presenting the students with a project that would trigger their learning process, by defining a problem that was adapted to their knowledge level. The project presented to students in the pre-exploratory experience (and also in the third and fourth cycles) had tremendous visual impact. The students had to build an object, like a dog or a robot (in the later cycles: a car and a motor-racing track), and develop a program with the aim of simulating the behaviour that each object had to execute. Thus, the students got immediate feedback regarding the correctness of their program by simply looking at the behaviour of the object. For example, the robot should follow its owner's orders.

So, for the first research cycle, we decided that one of the variables to be analysed would be the students' reaction when the project did not take advantage of visual feedback. Therefore, this cycle had two parallel tracks: one similar to the pre-exploratory phase, with visual behaviours (Group B2), and another without visual behaviours (Group B1). The non-visual project consisted only of data manipulation: The students had to implement a pharmacy sale store in which they controlled the entrance/exit of medical drugs. The visual project involved the development of a dog. For the second cycle, we introduced a visual feedback component in the pharmacy project and a data manipulation task on the other, with the aim to observe the students' behaviour in these different sequences: from abstract to visual and from visual to abstract.

Table II presents a summary of the projects developed in each cycle.

Table II
Groups of students that took part in the experience with projects developed.

	Pre-phase	1st cycle	2nd cycle	3rd cycle	4th cycle
Groups	A, C	B1, B2	B1, B2	D	D
Project	Dog Robot	Dog Pharmacy store	Dog with data manipulation Pharmacy store with visual feedback	Car	Car

Programming environment

The programming environment was SL itself, not any offline editor. SL is a persistent online 3D virtual world conceived by Philip Rosedale in 1991, and it has been publicly available since 2003 (Linden Research, 2007). It allows large numbers of users to connect, interact, and collaborate simultaneously in time and (virtual) space.

People are represented inside SL by an avatar and through it they can interact within the environment, including talking, walking, flying, and building.

Figure 1 shows a typical programming session during this research: We see 6 avatars on black rugs (students programming) and 2 teachers' avatars.

Figure 1: Typical programming session.

SL programming is done with the scripting language LSL, which has C-style syntax and keywords. Three-dimensional objects created in SL can receive several scripts that are executed concurrently. Each script has its own state machine: Program flow is sequential but structured by triggering events and responding to them (through either environment interactions or programmatic components), in addition to common methods from imperative/procedural programming, such as procedures and flow-control primitives. The programmer defines the states of each state-machine and how/when to switch state. The language's programming

libraries include functions for communication with external servers: sending and receiving e-mail, XML-RPC, and HTTP requests and responses.

SL enables synchronous collaboration among students because the system permits two or more avatars to edit the same object and share the same code while programming it. Figure 2 presents two avatars editing the same object (a car): The left window shows the car's content, which is a script that is opened by double-clicking. The script code (see Fig. 3) collects the name of all cars nearby and shows them to the owner's car (function *listProdut).*

Figure 2: Two avatars sharing an object.

```
File  Edit  Help
list Prod;
key gQueryID;

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

   for(i=0;i<listLength;i++)
        llOwnerSay("\nProdutos=" + llList2String(catal,i));

}

default
{
    state_entry()
    {
        llSetText( "CAIXA DE GESTAO", < 0.0, 0.2, 0.0 >, 1 );
        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 id, string msg){

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

    }
```

Figure 3: The code they share.

Asynchronous collaboration is also supported because the SL world is a persistent one. Students and teachers may access and leave in-world objects and messages to the other members (group and private messages are supported). When a user logs in all his/her messages are shown,

and he/she can see all the objects left in this world by others (and edit them, if adequate permissions have been set).

SL enables two types of writing communication: a public and private channel. All the communication that occurs through the public channel can be seen by everyone who is around, whereas in the private channel the communication is just done between two people.

Procedure

The project was presented to all the students in the first session; afterwards some students volunteered to participate in this research. From this point on, they formed groups of two elements and developed their projects inside SL, collaborating with each other. The teachers and students met in-world once a week, for about 2 hours, to keep track of students' progress, exchange ideas, and make suggestions. Face-to-face meeting did not take place, because the teachers-researchers were in Leiria and the students in Vila Real, 270 km apart. Once a month, we met to talk about the project in Vila Real.

On all the occasions the students had some difficulty with the code they had implemented and shared that difficulty with the teacher, so that they could observe it together and at the same time find out what was wrong and then follow the teachers' indications/instructions. In this way, the students could correct the code and continue working.

At the beginning of a lab session, the teacher explained the concepts that students had had doubts about and prompted them to find out the solution to their own problems.

Outcomes

Along this journey, we had always two concerns in mind, the teacher (teaching process) and the students' points of view (learning process).

Pre-exploratory and 1st cycle

According to the teacher's perspective, in the pre-exploratory cycle, it was necessary to improve several issues, namely,

- Communication amongst the participants – In this cycle, the public channel was used. This made communication between the teacher and students complicated since all the messages appeared on the screen at the same time and consequently it was difficult to understand who was saying what;
- The arrangement of the students' avatars in the classroom – As the students were dispersed in the classroom, the teacher might lose the notion of the global class; and
- Identify the students' difficulties during the self-study outside the weekly guidance session – Outside the weekly guidance session, the students continued developing their work. For better guidance, it would be useful to have a mechanism that could inform the teacher, by email or another outside system, about what the students had done throughout the week, the difficulties they had experienced, and attempts to overcome them.

In the following cycle, the first two issues, mentioned above, were partially resolved.

- From the students' perspective, some difficulties were felt in using the SL interface. The ones who were in the beginning level of the study (Group A) encountered some obstacles in understanding the LSL semantics and dealing with the compilation errors. The other ones, Group C, considered the functions to use for implementing a kind of behaviour a hard task.
- Students from both groups (A, C) liked to learn how to program inside SL. Some of them began to develop objects and programs for other SL residents.

During the first cycle, our concerns were to understand the reasons for the students' difficulties and at the same time improve the teaching process itself. We decided to introduce a different type of project with the goal to observe if the students' reaction to SL was the same as their previous colleagues. One project was programming motions and behaviours and the other a more traditional text data processing.

During this cycle of reflections, we concluded that those students who were only doing the project with data manipulation did not like it and were frustrated and stressed; they confirmed this when an inquiry was made. The other group of students showed opposite behaviour as they were engaged, motivated, and enjoyed themselves in developing the project (Esteves, 2008 a; Esteves et al., 2008 b).

2nd cycle plan

Teacher – The teacher should write, inside the students' code, a comment addressing the student's uncertainty, especially concerning compilation and execution errors. For the communication and spatial arrangements of the students' avatars, we maintained the same methodology used in the previous cycle.

Project – Some visual components were introduced in the pharmacy store project, with the aim to change the students' behaviour and data manipulation in the other project.

The results of the 2nd cycle showed:

The students' perspective:

Project – The alteration made in the project did not change the students' behaviour from Group B1 (pharmacy store project). Group B2's experience remained unchanged; they were motivated and engaged despite mentioning that this modification made the project more difficult. In spite of creating an algorithm in the beginning, the students showed some difficulties as far as understanding what they were supposed to do throughout the project.

Selecting adequate library functions for object control – The students showed some awkwardness in understanding what the predefined functions did. They constantly said, "I do not understand what this function does, and how can I use it?"

SL interface – The uncertainty experienced in using the SL interface persisted. Some students mentioned that they had trouble understanding the teacher's explanation, especially about the use of SL interface.

Compilation and execution errors – There was a decline in the repetition of the same mistakes.

The teacher's perspective:

Communication – The use of a private channel to address the students' uncertainties worked well. The students said they appreciated this mode of communication because they felt they had a private teacher and could discuss their doubts without feeling embarrassed about lagging in relation to others.

Discovering the students' difficulties during self-study – The students' use of a note card to express their uncertainties when they were outside the weekly guidance session was ineffective. They preferred to send an email or a private message inside SL. The teacher spent many hours helping the students inside SL and outside the class. The students said they had all the support they needed from the teacher and that it would be hard for that to happen in a normal class. Though it was extra effort for the teacher, she recognized that it was helpful for the students: "Sometimes, students struggled with little things and a push might help them to overcome the difficulty."

Reflection

Although the students have already had contact with a C programming language, part of the B1 Group presented many problems understanding the basic concepts, namely, repetition of instructions – when to use them and how; function utilization – they did not know how to use the function definition, i.e., how to call a function; difficulty in structuring their thoughts because they did not understand what they were being asked to do. Thus, they did not correctly do the algorithm. One point to extol on was that these students did not recognise their own weaknesses. They said, "I know the C language; I do not have difficulties in C." However, they did not demonstrate that knowledge when they were developing the project. In relation to these difficulties, Winslow (1996) mentioned that novice programmers neglect strategies, are limited to the general knowledge of the subject and that knowledge is fragile. Fragile knowledge is described as something that the student knows but fails to use when necessary.

The project did not help the students, since it was only based on data manipulation, despite all the alterations made. The teacher could not overcome these difficulties associated with the deficiency the students had on the C language, and neither changed their opinion about programming.

The B2 Group's project had visual impact; although they showed the same difficulties in language and in understanding the project, throughout the project development, they overcame part of their difficulties, had a pleasant time, and mainly changed their own opinions about learning a programming language.

One of our concerns as CS teachers is the superficial level of learning that novice students show and their difficulty in understanding the project (in developing the algorithm). In

order to address this, we implemented a project based learning (PBL) methodology approach in the next cycle. Once within PBL, the problem acts as the catalyst that initiates the learning process (Duch, 2001). Furthermore, effective problems should engage the students' interest and motivate them to probe for deeper understanding of the concepts being introduced (Duch).

After several interactions, we realized that the students' main difficulty in relation to predefined functions was the English language. For this reason, we decided to translate the principal function, with an example of the application, to the Portuguese language for the next cycle.

Despite the fact that the class was online, few students experienced difficulty understanding the teacher's explanation of the use of the SL interface. Hearing the instructions is easier than reading it, as the interface has many options.

In relation to compilation and execution errors, due to the fact the teacher wrote a comment in the students' code every time they experienced difficulties helped them remember the solution to the problem and avoid repetition of the mistake.

According to the teacher's perspective, the use of a private channel to address the students' uncertainties is an important act, although it also increases the teacher's work. Moreover, a deep connection was created between the teacher and the pupil, and that is not easy to achieve in traditional classes. It was complicated for the teacher to manage all the students' requests and comment on their codes at the same time (see Fig. 4). It is essential to find a way that the teacher can give an immediate response to the students during the lectures, so that the students do not have to wait so long for the teacher's response to their questions.

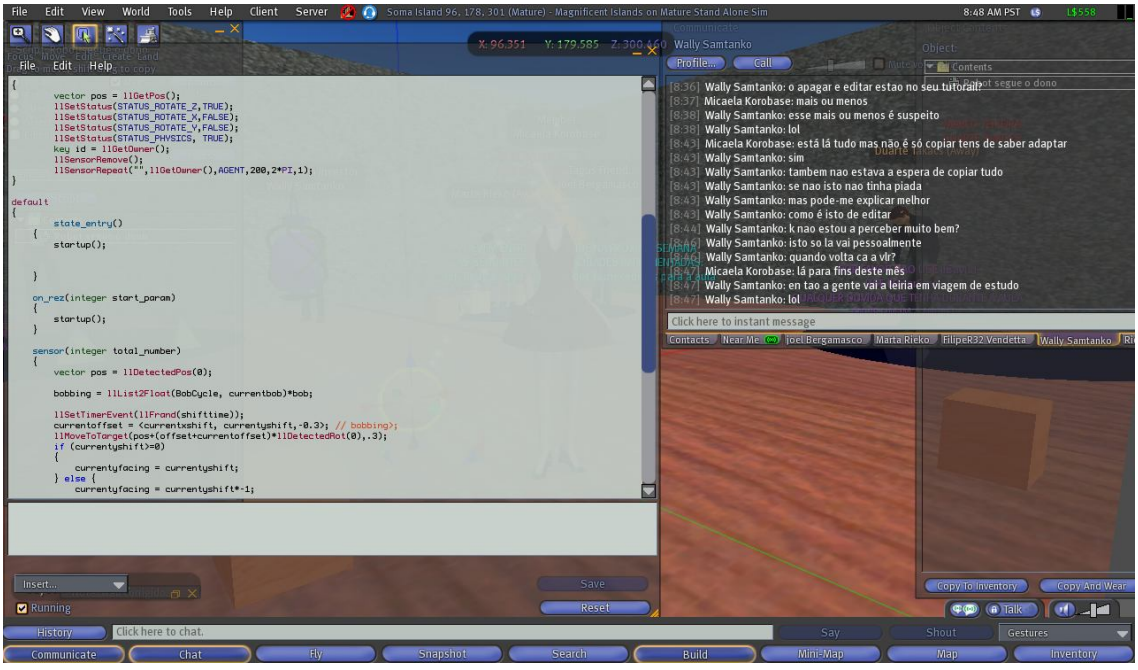

Figure 4: The teacher's work during the lessons.

3rd cycle plan

For teacher:

The teacher should be present in the first class to explain the SL interface to avoid misunderstanding and recurrence of uncertainty about the SL itself.

To communicate, we used the same methodology; however, the teacher should have written a few sentences such as "wait a moment, I'm talking to your colleague …" or "there is not a semicolon at the end of the instruction …" to copy and paste every time it was necessary. In this way, we would have been able to see if the teacher's work decreased.

Class:

In the classroom, we maintained examples of objects using simple programs so that the students might consult the project developed by other students.

During this cycle, we decided to use the methodology project-based learning. The project was the starting point of the students' learning process.

The students discussed the project among themselves and presented a report to their teacher 2 weeks after the project had begun. In this report, they explained what the problem was, the events presented, and how they thought it could be resolved, i.e., the algorithm. In the next session within SL, the students and teacher examined several solutions presented according to the problem, exposing the strengths and weaknesses of each solution during the phase of brainstorming.

The goal here was to make sure that they had gained a better, deeper, and more detailed understanding of the mechanisms or processes underlying the problem. Thus, the teacher challenged the students to clarify their own ideas, encouraging them to elaborate on the subject matter, questioning ideas, looking for inconsistencies, and considering alternatives. By doing so, the teacher helped the students organize their own knowledge, resolve their misconceptions, and discover that which is not so well understood. From this point, the students began to develop their project studying the language LSL, with the aim to solve the problem.

It was necessary to use the *moodle* as a repository of contents, such as the LSL functions and examples in the Portuguese language.

Another teacher followed the teacher research instructions and used them to teach in another class, and we considered this as the fourth cycle. Its goal was to compare the results.

Project:

The project presented to the students must have visual feedback and data manipulation. In this way, the project consisted of developing a car with x litres of petrol. The car should always follow the motor-racing track, although its configuration could be changed any time. As the car moves, it consumes petrol. So, the car owner needs send it to the petrol station to refill, if not, when it stops, the owner is fined.

The results of the 3rd and 4th cycles showed:

The students' perspective:

Project – The students mentioned that they liked the project, although it was difficult because it had many components to interact with. During this cycle, we observed that the students were motivated and engaged in developing their projects, and the students confirmed this in the questionnaire.

Methodology – They liked the methodology used. It was the first time they had discussed the solution presented for the project within group. They said "I like it and it helps me to see the mistakes of my solution before I implement it," and "It helps me to understand the project better and think about the problems that I could not see before."

Selecting adequate library functions for object control – There was an improvement in understanding LSL function from the students and how to use it.

SL interface – As the project developed, students' uncertainties disappeared.

Compilation and execution errors – The same result as that in the previous cycle was obtained.

The teacher's perspective:

Communication – The use of pre-defined phrases helped avoid stress during the class. During the brainstorm phase, the teacher missed having a blackboard, where the various solutions and their advantages/disadvantages could have been written. This difficulty was overcome with the teacher's ability to copy and paste text of the solutions discussed when necessary.

Discovering the students' difficulties during self study – The students did not use the doubts object, as their other colleagues preferred to send an email or wait for the class to discuss the project with their teacher and partners.

Reflection

With the teacher's presence in the first class to explain the SL interface, the students' uncertainties disappeared as the project developed.

Utilization of the private channel to individually address the students' uncertainties worked well. They had the same reaction as their colleagues.

The LSL function and examples in Portuguese helped students to better understand the LSL. As these students were at the beginning of their programming language studies, it was important to see their reactions to compilation and execution errors.

- Compilation errors: The methodology used by the teacher, which involved writing a comment in the code explaining how and why the errors occurred, helped the students avoid and understand the mistakes.

- Execution errors: Execution errors occurred when the object presented a behaviour different from the one the students had expected; for example, the car turned left when it should have turned right, they questioned why this happened and tried to resolve the problem themselves. They asked for the teacher's help when they could not fix it. This comports with Williams' (2007) belief regarding the role visualization plays in teaching, which is that computer-based representation has the potential to play a vital role in the development of problem solving abilities.

From the teacher's point of view, the use of pre-defined phrases ready to be copied and pasted when needed helped avoid stress during the class. In this way, the teacher was able to give an immediate reply to the students.

The use of PBL methodology worked well, since the students understood the project objectives at the beginning and correctly structured their ideas. This avoided misconceptions along the project development, as it noted by O'Kelly (2005). During the brainstorm phase, the teacher missed having a blackboard where problems, students' ideas or hypotheses, and questions about the project could be written, ensuring discussion. This problem was overcome by the teacher and students copying and saving the discussion. This written record helps the students keep track of their problem solving and provides a focus for reflection.

Discovering the students' difficulties during self-study, this group, like their colleagues, preferred to send an email or wait for class to discuss the project with their teacher and partners.

Discussion

In this study, we aimed to collect baseline data on teaching and learning to program in SL. Our main concern in this research was to formulate a framework for future SL usage as a platform for teaching/learning a programming language in a qualitative and quantitative way, in order to determine if students learn better here than in traditional environments.

According to the students, particular factors motivated them in choosing this way of learning. The majority mentioned that they wanted to try a new form of learning, although they had a bad experience in learning a programming language. Others wanted to try the program, once other colleagues had tried it and liked it. A few did not have a specific reason for choosing this way of learning.

Concerning the students' process of learning, we can conclude that some important points should be taken into account. First, the type of project is very important, since in this environment everything moves and interacts with people, and it is not understandable to develop a project which does not use these environment characteristics. As Duch (2001) mentioned, effective problems should engage the students' interest and motivate them to probe for deeper understanding of the concepts being introduced.

In this research, we verified that the use of a non-visual project is a mistake and results in a loss of time for both the students and teacher. The students who focused primarily on non-visible techniques, such as data structures and string processing, benefit from the environment just for enhanced context and not as a source of feedback for programming behaviour, did not seem to exhibit any motivational advantage over students who employed a traditional console-

oriented (text-only) approach. The students did not learn and struggle themselves and did not understand why they had to do that kind of project inside an environment where everything moves, has colours, etc. The teacher spent her time trying to motivate the students, without results. At the end of the project, some of these students' feelings about programming remained unchanged, i.e., they still did not like programming.

Those students (B2 group) who had a project with a visual behaviour, although they showed the same difficulties on language, they overcame part of them. They believed they had spent a pleasant time and changed their opinions about learning a programming language.

As Barret (2005) argues, learning demands both the fun of playing with ideas and the hardness of refining and reworking ideas, and both complementary parts are needed for learning. The fun transpired to be what Papert (1996) termed "hard fun"; in that it is both challenging and interesting, and this implies "hard."

The teacher often thought students understood her explanations, and a few minutes later she realized they had not. Utilization of the PBL avoids these problems, allocating more responsibility to the students. This could be observed in the last cycle.

Ten percent of the students were so enthusiastic about building that they avoided/forgot about the programming of the objects. Sometimes, the teacher saw that students were so enthusiastic in building but at the same time quiet as they did not put any doubts about the project. In these instances, the teacher must be vigilant in preventing this from occurring.

At the end of the project, the students commented that the principal drawback was the use of the programming language (LSL) used in SL rather than C. As in the other lectures they need C or C#, the study of LSL forces them to study another language that is not necessary during their course. This argumentation is not relevant as the objective of an introductory language, as this one, is to understand the concepts (variables, functions, lists, etc.), as noted by Jenkins (2002).

The teacher's work was more intense and stressful than in a traditional class because she had to prepare her lessons in advance, writing out everything she wanted to teach. To do that, the teacher has to predict the students' potential difficulties and the eventual questions that could arise. Thus, the teacher should be prepared the address these difficulties and questions in order to give rapid feedback the students.

The teacher also had to prepare the classes' supplies and some visual materials (programming objects behaviour). During the class, the teacher helped the students with ongoing work and answered their questions.

Communication between the teacher and students, both in public or private channel, was informal. That is a "rule" inside SL. The teacher was seen by the students as a colleague, a member of the community with more knowledge, with whom they could talk to and discuss their uncertainties. An interesting fact occurred during the first cycle: The teacher inadvertently wrote her explanation to a student in Caps Lock, and after a few minutes the student complained "Are you angry with me? It is not my fault that I do not know how to do this!" This fact is clearly an example that in writing communication it is important to be careful to avoid misunderstanding.

In sum, to use SL for teaching/learning a program language, the problem presented must have a strong visual impact, be interesting to the students, and be complex enough. The communication system should use the public channel to give general explanations to the students and the private channel to address their uncertainties. The use of PBL methodology is recommended.

Conclusions

The aim of this study was to analyse whether SL presented conditions that could be used as a platform for teaching and learning a programming language. In this paper, we have presented the study performed and discussed the results.

We have concluded that using Second Life as a platform for teaching and learning a programming language could benefit novice students. However, it is necessary to be mindful of the type of project presented, i.e., it must meet the students' interests.

We think SL is a very good way to introduce a programming language because the LSL is not as complex as C. Students develop projects they like and work inside an environment they appreciate. They are exposed to an international active programmer community, where their work can be recognized, and this is difficult to achieve in traditional classes.

As future work and research to be developed, we point out that it would be helpful and extremely relevant if the teacher could follow the student's progress/efforts and help him/her in a more effective and direct way. To achieve this, it would be necessary to develop an automatic mechanism that follows the students within the SL world, answering their questions when the teacher is not present, while simultaneously recording this record and sending it back to the teacher.

Furthermore, this study adopts an optimistic view of the impact of the experience with virtual settings and related activities. We acknowledge that there may be negative effects if the experience is unpleasant, unrewarded, or under-developed. For instance, a negative experience may reduce an individual's virtual self-efficacy, inhibit his/her performance, and limit his/her ability to transfer knowledge effectively in virtual work. Further exploration in this direction may be productive.

Bibliography

Almeida, E. S., Costa, E. B., Braga, J. D. H., Silva, K. S., Paes, R. B., & Almeida, A. M. (2002). AMBAP: Um Ambiente de Apoio ao Aprendizado de Programação. *In X Workshop sobre Educação em Computação, Florianópolis. Anais do WEI 2002/ SBC2002.*

Amershi, S., Carenini, G., Conati, C., Mackworth, A. K., & Poole, D. (2008, January). Pedagogy and usability in interactive algorithm visualizations: Designing and evaluating CIspace. *Interact. Comput. 20*(1), 64–96.

Barrett, T. (2005). Who said learning couldn't be enjoyable, playful and fun? – The voice of PBL students. In Esa Poikela & Sari Poikela (Eds.), *PBL in Context – Bridging Work and Education* (pp. 159–175). Finland,Tampere University Press.

Ben-Bassat Levy, R., Ben-Ari, M., & Uronen, P. (2003). The Jeliot 2000 program animation system. *Computers & Education 40*, 1–15.

Beveridge, M., & Parkins, E. (1987). Visual representation in analogical problem solving. *Memory & Cognition 15*, 230–237.

Carlisle, M. C., Wilson, T. A., Humphries, J. W., and Hadfield, S. M. 2005. RAPTOR: a visual programming environment for teaching algorithmic problem solving. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA, February 23 - 27, 2005). SIGCSE '05. ACM, New York, NY, 176-180.

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3-D tool for introductory programming concepts. In J. G. Meinke (Ed.), *Proceedings of the Fifth Annual CCSC Northeastern Conference on the Journal of Computing in Small Colleges*, Ramapo College of New Jersey, Mahwah, New Jersey, United States (pp. 107–116). Consortium for Computing Sciences in Colleges, USA.

van Dam, A., (2005, September/October). Visualization research problems in next-generation educational software. *IEEE Computer Graphics and Applications 25*(5), 88–92.

Dann, W., Cooper, S., & Pausch, R. (2000). Making the connection: Programming with animated small world. In *Proceedings of the ITiCSE* (pp. 41–44). ACM Press: New York.

Dann, W., Cooper, S., & Pausch, R. (2001). Using visualization to teach novices recursion. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 109–112). Canterbury, England, ACM.

de Freitas, S., & Yapp, C. (2005). *Personalizing learning in the 21st century.* Stafford, UK: Network Education Press.

Dickey, M. D. (2003) Teaching in 3D: Pedagogical affordances and constraints of 3D virtual worlds for synchronous distance learning. *Distance Education 24*(1), 105–121.

Dijkstra, E. W. (1989). On the cruelty of really teaching computing science. *Communications of the ACM 32*(12), 1398–1404.

Duch, B. (2001). Writing Problems for Deeper Understanding. In B. Duch, S. E. Groh, & D. E. Allen (Eds.) *The power of problem-based learning: A practical "how to" for teaching undergraduate courses in any discipline* (pp. 47-53). Sterling, Virginia, 2001, Stylus Publishing.

Esteves, M. & Mendes, A. (2004). A simulation tool to help learning of object oriented programming basic. In *Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference*, October, 20–23, 2004, Savannah, Georgia, USA, (pp. 7-12). IEEE: Washington, DC.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2008), a. Contextualization of programming learning: A virtual environment study. In *Proceedings of the 38th ASEE/IEEE Frontiers in Education Conference*, October 22–25, 2008, Saratoga Springs, NY (pp. 17–22). IEEE: Washington, DC.

Esteves, M., Antunes, R., Morgado, L., Martins, P., & Fonseca, B. (2008), b. Using Second Life in programming's communities of practice. In Proceedings *14th Collaboration Researcher' International Workshop on Groupware, CRIWG 2008*, Omaha, Nebraska, September 14–18 2008, *Lecture Notes in Computer Science.* Springer, US.

Field, J. (2007). Looking outwards, not inwards. *ELT Journal 61*(1), 30–38.

Figueiredo, A. D. and Afonso, A. P. (2006). *Managing learning in virtual settings: The role of context.* In António Dias Figueiredo (Eds.). Information Science Publishing (Ideal Group), 701 E. Chocolate Avenue, STE. 200, Hershey, PA 17033, USA.

Guzdial, M., Kolodner, J., Hmelo, C., Narayanan, H., Carlson, D., Rappin, N., et al. (1996). Computer support for learning through complex problem solving. *Communications of the ACM 39*(4), 43-45.

Gomes, A., Areias, C. M., Henriques, J., & Mendes, A. (2008, Julho). Aprendizagem de programação de computadores: Dificuldades e ferramentas de suporte. *Revista Portuguesa de Pedagogia 42*(2), 161–179.

Gomes, A., Mendes, A. J. (2007). An environment to improve programming education. In B. Rachev, A. Smrikarov, and D. Dimov (Eds.), *Proceedings of the 2007 International Conference on Computer Systems and Technologies*, Bulgaria, June 14–15, 2007, ACM: New York.

Gray, W. D., Goldberg, N. C., & Byrnes, S. A. (1993). 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*, 9(1), 131–140.

Hundhausen, C. & Brown, J. (2007). An experimental study of the impact of visual semantic feedback on novice programming. *Journal of Visual Language and Computing 18*, 537–559.

Hundhausen, C. D. & Brown, J. L. (2008, January). Designing, visualizing, and discussing algorithms within a CS 1 studio experience: An empirical study. *Comput. Educ*. *50*(1), 301–326.

Jenkins, T. (2002). On the difficulty of learning to program. In *Proceedings of 3rd Annual LTSN_ICS Conference,* Loughborough University, United Kingdom, August 27–29, 2002 (pp. 53–58). The Higher Education Academy: York, United Kingdom.

Kiili, K. (2005). Digital game-based learning: Towards an experiential gaming model. *Internet and Higher Education 8*, 13–24.

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The Blue J system and its pedagogy. *Journal of Computer Science Education 13*, 249–269.

Lahtinen, E., Mutka, K. A., & Jarvinen, H. M., (2005). 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 (pp. 14–18). ACM Press: New York

Lethbridge, C., Diaz-Herrera, J., LeBlanc, R., Jr., & Thompson, B. (2007, May). Improving software practice through education: Challenges and future trends. In *2007 Future of Software Engineering* (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 12-28.

Lessard-Hébert, M., Goyette, G., & Boutin, G. (1994). Investigação qualitativa: Fundamentos e práticas. Lisboa: Instituto Piaget.

Lewun, K. (1946). Action research and minority problems. *Journal of Social Issues 2*(4), 34–36.

Linden Research (2007). What is Second Life? Retrieved November 18, 2007, from http://secondlife.com/whatis/

Linn, M. C. & Clancy, M. J. (1992). Can experts' explanations helps students develop program design skills? *International Journal of Man-Machine Studies*. 36, 4 (Apr. 1992), (pp. 511-551).

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice: Urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education,* Portland, OR, USA, March 12-15, 2008. SIGCSE '08 (pp. 367–371). New York: ACM. .

Menchaca, R., Balladares, L., Quintero, R., & Carreto, C. 2005. Software engineering, HCI techniques and Java technologies joined to develop web-based 3D-collaborative virtual environments. In *Proceedings of the 2005 Latin American Conference on Human-Computer interaction*, Cuernavaca, Mexico, October 23–26, 2005. *CLIHC '05 124*. New York: ACM.

Miliszewska, I. & Tan, G. (2007). Befriending computer programming: A proposed approach to teaching introductory programming. *Journal of Issues in Informing Science & Information Technology 4*, 277–289.

Milne, I. & Rowe, G. (2002). Difficulties in learning and teaching programming – views of students and tutors. *Education and Information Technologies 7*(1), 55–66.

Motil, J. & Epstein, D. (2000). JJ: A language designed for beginners (less is more). Retrieved July 16, 2008, from http://www.ecs.csun.edu/jmotil/TeachingWithJJ.pdf

Naps, T. L., Röbling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., et al. (2003). Exploring the role of visualization and engagement in computer science education. *Annual Joint Conference Integrating Technology into Computer Science Education*. Aarhus, Denmark (pp. 131–152).

Newman, D., Goldman, S. V., Brienne, D., Jackson, I. & Magzamen, S. (1989) Computer mediation of collaborative science investigations. *Journal of Educational Computing Research 5*(2), 151–166.

Noyes, J. M., & Garland, K. J. (2003). Solving the Tower of Hanoi: Does mode of presentation matter? *Computers in Human Behavior 19*, 579–592.

O'Kelly, J. and Gibson, J.P. (2005) PBL: Year One Analysis—Interpretation and Validation, PBL International Conference, PBL *In Context – Bridging work and Education*, Finland, June 9th –

Pierson, W. C. & Rodger, S. H. (1998). Web-based animation of data structures using JAWAA. *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education* 30 (1), 267–271.

Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? In *Proceedings of the Second international Workshop on Computing Education Research,* Canterbury, United Kingdom, September 09–10, 2006 (pp. 17–28). ICER '06. ACM: New York.

Schweitzer, D. & Brown, W. (2007). Interactive visualization for the active learning classroom. *SIGCSE Technical Symposium on Computer Science Education*. Covington, KY (pp. 208–212).

Papert, S. (1980) *Mindstorms: Children, computers, and powerful ideas.* Basic Books, Inc.: New York.

Papert, S. (1996). *The connected family: Bridging the digital generation gap.* Atlanta, Georgia, Longstreet Press.

Shneiderman, B., (1983). Direct manipulation: a step beyond programming languages. *EEE Computer 16*, 57–69.

Sloane, K. D. & Linn, M. C. (1988). Instructional conditions in Pascal programming classes. In R. E. Mayer (Ed.), *Teaching and learning computer programming* (pp. 207–235). Hillsdale, NJ: Lawrence Erlbaum Associates.

West-Burnham, J. (2005). Understanding personalization. In J. West-Burnham & M. Coates (Eds.), *Personalizing learning*. Stafford UK: Network Educational Press.

Williams, D. J. and Noyes, J. M. (2007). Effect of experience and mode of presentation on problem solving. *Computers in Human Behavior* 23(1), 258-274.

Winslow, L. E. (1996). Programming pedagogy – A psychological overview. *SIGCSE Bulletin 28*, 17–22.

Zuber-Skerritt, O. (2000). A generic model for action learning and research programs in Graduate Management Theses. In *Action learning, action research and process management: Theory, Practice, Praxis,* Action Research Unit, *Faculty of Education*, Griffith University, Brisbane.