A game Engine to Learn Computer Science Languages

Ángel Serrano-Laguna, Javier Torrente, Borja Manero, Baltasar Fernandez-Manjon
ISIA Department, Complutense University of Madrid, Spain
({angel.serrano | jtorrente | balta} @fdi.ucm.es; borja@sip.ucm.es).

***Abstract*—There is an increasing interest in providing Computer Science (CS) instruction to a wider sector of the population. On the one hand, it would be convenient to include CS instruction in higher education beyond engineering disciplines, since CS has become a powerful catalyzer for development of society, and therefore the need for a workforce with solid CS background is growing. On the other hand, it would be beneficial to bring CS instruction to primary and secondary education, used as a vehicle to increase interest in CS and capture talent for STEM disciplines from early stages. However, successful delivery of CS instruction to a wide audience is a challenge. Game-based learning is one of the most promising approaches at the moment, since they have the power to appeal to wider audiences.**

In this paper we identify the need to find more scalable game-based instruction paradigms that can be easily adapted to variable levels of complexity and contents related to CS. We present a flexible and scalable game architecture, and a game model to create videogames for learning CS languages, along with a game engine developed as a reference implementation. The game model focuses on level-based games where the student has to introduce short text fragments or programs to solve each of the levels. This game model is consistent to others found in the literature (Scratch, Logo, etc.) that have proven it effective, since it allow students to discover programming in a self-exploratory way, using their own intuition and learning from their mistakes. Our approach is scalable because (1) it separates the CS language used to write the programs from the game design, allowing reusing the games with different CS markup or programming languages; and (2) it provides a system of levels that allows incremental learning of CS language structures.

The approach was tested by developing "Lost in space", an educational game for learning XML. In this game, students control a spaceship, and their goal is to reach a safe point in each of the levels. They provide instructions to the ship with short programs that they write using XML-based instructions. At the beginning students can use a small set of instructions. As they master these types of instructions, new ones become available, supporting in this manner scaffolded learning. The game was tested with undergraduate students from computer science and social sciences, by comparing it with traditional instruction (i.e. lecture). Students who played the game were much more engaged than those who attended the lecture, showing a more active attitude along the whole experience and also spent more time practicing after class. Findings also suggest that the game was effective for instruction regardless of the background of the students. However, the educational gain observed with the game-based instructional approach, even effective, was not significantly higher than traditional instruction.We think that our approach is adequate to introduce CS languages in general, as well as new programming languages.

***Index terms*—Application software, educational technology, software architecture, computer programming, educational videogames**

I. Introduction

THE idea of using videogames to support learning of computer programming is not new, as it dates back almost to the origins of instructional games [1]. However, this interest seems to have reached a peak recently, with numerous advocators bringing the potential of digital game technology to the field of computer programming education. First, the popularity of digital games can attract talented people to computer science and software engineering, professions that are essential in today's economy. Second, digital games can help smooth the learning curve for novice programmers by providing a highly visual and motivating environment. Finally, learning programming since childhood can foster development of high level thinking and problem-solving skills.

As a consequence, there are numerous initiatives dedicated to facilitate using game technology for learning computer programming. Most of them target kids, although other software targets college students. The drawback of these systems is that they are hard to scale. They are usually devised for a specific target audience, educational goal and/or programming language, which makes it difficult to repurpose and reuse the software in different settings.

In this paper, we present a scalable game engine to create games for learning programming that could be used by students with different backgrounds. The game engine facilitates educational game development by providing game mechanics that are appropriate for learning programming. It is scalable because (1) it separates the programming language being learnt from the game design, allowing reusing the games with different programming languages; and (2) it provides a system of levels that allows incremental learning of programming structures. The engine has been used to develop the game *Lost in Space*. We have conducted two case studies with college students from different backgrounds (computer science and social sciences respectively) to evaluate the effectiveness of the game for learning computer programming.

II. Related Work

Game technology has been long used to support learning programming. One of the preferred approaches is to use activities related to game design and development. Students create simple program snippets that control characters and objects in a game environment, usually through a visual or simplified programming language. The visual condition of the results obtained facilitates getting students rapidly engaged in programming. Multiple tools have been developed around this paradigm, like GameMaker [2], Alice [3], Microsoft's Kodu

978-1-4799-3922-0/14/$31.00 ©2014 IEEE

[4] or Scratch [5]. The literature is full of experiences where this paradigm has been successfully applied. For example, De Kereki [6] reports effective use of Scratch as a motivational tool in an introductory programming course. In other example, Chen and Cheng [7] use videogame development as the core activity of their programming introduction course.

Other studies have explored the activity of playing digital games in programming courses. Our approach is similar to those. In many of these games students do not write programs, using the videogames as mere containers of theoretical contents about programming, and as a means to increase students' motivation. However, the lack of active programming is a limitation of the approach, since it is an essential activity to learn programming. For example, in [8] authors report the use of a role-playing game where quests are directly related to programming concepts. In a similar approach, the use of a game environment provided a significant increase of student motivation towards the subject [9] and student performance. In [10] authors propose two mini-games: a typing game and a fill-in-the-blank game to practice Java syntax, aiming to improve players' basic Java skills. In [11] a game is used to teach C with crosswords puzzles and duck shot games and in [12] a game is used to teach C++ concepts.

There are also educational games where students need to write little programs to move on in the game or achieve a specific goal. In [13] authors present an augmented reality game that uses cards as instructions to create shapes. In this game, players must combine several cards that represent basic instructions to create a target shape using augmented reality. In [14] authors present a 3D game where players' avatars are controlled using a subset of the Logo instructional language. Similarly, one of the mini-games proposed in [10] invites players to introduce commands to guide the main character to a target point using a simple programming language. A hybrid approach is described in [15], where students are asked to program an algorithm in order to beat a game-based challenge. The game runs this algorithm and gives the student a final score, based on the algorithm effectiveness.

The main drawback of these approaches is that the games developed are hard to reuse and to scale. Most of them cover specific languages and specific programming concepts, and makes it difficult to adapt them to cover new concepts or new programming languages to fit other target audiences.

III. Game Engine

In this section, we present the game model proposed for our approach, and the engine architecture that supports it.

A. Game Model

The engine architecture is built upon a game model that defines the main high level features for our approach. This model is defined through the following key ideas:

1) Students must code in the game.

Students must create programs to advance in the game, following a learning-by-doing approach.

2) Promote reflection, avoid time pressure.

We propose turn-based strategy and to avoid game mechanics that require fast reaction. This allows students to take their time to think out a good solution, which is good practice in programming and promotes reflection.

3) Separate input and game mechanics

The actions that are available in the game must be independent from how they are introduced in the game. This adds scalability as it allows reusing a game for learning different languages. The game defines a set of actions that users can perform in the game (e.g. move, rotate, shoot, etc.). This action set is linked to a set of structures (e.g. procedures, loops, conditions), which enables players to generate more complex behaviors in the game. An interpreter is configured to translate orders formulated in the target programming language (e.g. Java, C++, Python, etc.) onto these game sets of actions and structures.

4) Level structure.

It enables incremental introduction of new concepts and programming constructs as the student becomes more skilled, facilitating a balanced level of challenge that keeps the student engaged and prevents frustration [16].

5) A clear goal is set up for each level.

Clear goals are a desirable feature for any good video game [17], and it also facilitates writing programs.

6) Use scores to promote competition between peers and to provide a sense of progress.

Scores are used as simple metaphors to engage students.

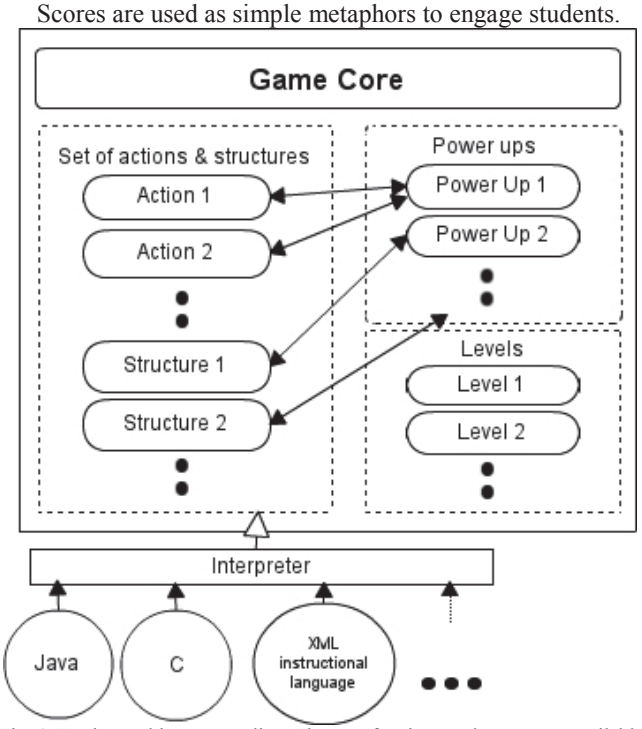

Fig. 1. Engine architecture outline. The set of actions and structures available is directly related to the power-ups unlocked in the game levels. An interpreter translates the programs to the set of actions and structures.

B. Engine Architecture

In this section, we present the engine architecture that supports the model presented in the previous section. Fig. 1 shows the main components of the architecture. These

components are:

1) Game core

The game core provides basic functionality for the game. It includes, among other components, a system of game rules, a physics engine and a rendering engine.

2) Set of actions and structures

This component contains the set of finite actions and control structures that players can use within the game.

3) Interpreter

The interpreter translates the programming code introduced by the students into game actions and structures, making the programming language to interact with the game exchangeable. For every new language that needs to be introduced, it is required to create a new interpreter able to translate it.

4) Levels

Every level is composed of logic blocks. Every block has its own logic and behavior within the game (defined in the game core). To make them extensible, levels are defined apart, in a format understandable by the game engine. Levels can be created or adapted to meet any specific needs and to adjust the duration of the game.

Levels are short, easy at first, and their complexity increase through the game. The game contains power-ups, each of them representing an action or a structure of the defined set. Ideally, every new power-up allows the player to perform a new action that is required to beat the current level.

On the player side, these power-ups unlock new programming structures/elements that the player use (and learn) by writing new programs.

This mechanism is scalable, allowing designers to add new power-ups and puzzles every time the game needs to be expanded to add new concepts or programming structures.

All levels are defined in an easy-to-read format that is technology agnostic. For example, *Lost in Space* uses the XML format for its level definition (Fig. 2).

```
<phase>
  <wall x="0" y="0" height="9" width="5" />
  <wall x="4" y="0" width="9" height="3" />
  <wall x="6" y="2" width="7" height="8" />
  <player x="5" y="4" />
  <exit x="5" y="6" />
</phase>
```

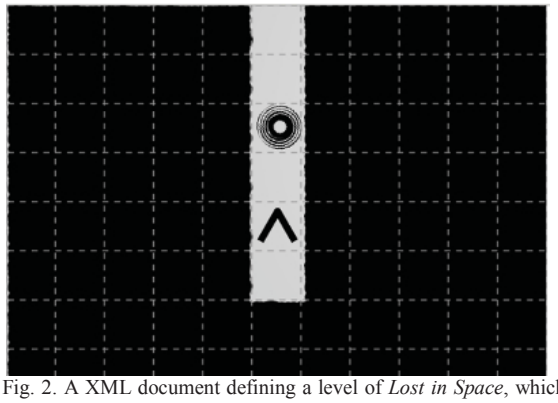

Fig. 2. A XML document defining a level of *Lost in Space*, which is shown below.

Each tag represents one of the blocks present in the level grid. The example defines three types of game elements: walls (3), player (1) and exit (1). The behavior and logic of each block is defined in the game core.

IV. Lost in Space

Lost in Space[1] is built upon the game model and architecture presented in section III. The game was developed to introduce new programming languages to students with different backgrounds. Fig. 3 shows a screenshot of the game.

The game screen is divided in two parts. The left side contains two elements: the code interpreter text area (bottom), where players must introduce their code snippets; and a help window (top) where syntax clues about the unlocked powers collected are shown.

The right side shows the current level. The goal for each level is simple: drive the player's spaceship to the current level exit (a wormhole), eluding any obstacles that may be laid out between them. These include spaceships that can be *allied* (the player can write instructions to control them) or *enemies* (their behavior cannot be controlled), obstacles (*rocks* and *walls*), *safe zones* (where the player cannot be hit by enemy fire) and *triggers*, which release actions in the game (e.g. movement of obstacles, shooting, etc.). If the player's spaceship is hit by a shot or collides with an obstacle or another ship, it is destroyed and the level starts again.

To complete the levels, the player counts with several atomic operations. These operations affect the main ship and the allied ships, and are unlocked in the course of the game. In the last level the player can use a total of 5 instructions: *move*, *rotate*, *shoot*, *wait* and *disappear* (to avoid collisions).

The game flow goes as follows: the player writes a program and submits it. The code is analyzed and interpreted by the game. If the syntax is correct, it generates a set of actions that are executed in the game. Otherwise it reports the error back to the player. As the player advances in the game, power-ups are unlocked, appearing on the top left side of the screen.

Table I provides some examples of programs to interact with the game. As the table shows, the game supports two target programming languages: Java, and an XML-based programming language. Although XML is not a programming language but a markup language, it provides a well-defined syntax which allows building programming or declarative languages on top of it. For example, some technologies like *Ant* or *Maven* use XML-based languages to define procedural behaviors.

TABLE I
EXAMPLES INPUT PROGRAMS FOR THE GAME

Effect in the game	Java	XML
Move ship 4 spaces	`ship.move(4);`	`<move distance="4"/>`
Make ally shoot	`if (ship.getId().equals("ally"){ ship.shoot(); }`	`<actions idref="ally"> <shoot/> </actions>`
Shoot 4 times	`for (int i = 0; i<4; i++){ shoot(); }`	`<actions repeat="4"> <shoot/> </actions>`

[1] Available at *http://gleaner.e-ucm.es/xml/index.html* at date April 10, 2014. Source code available in *https://github.com/anserran/lostinspace.*

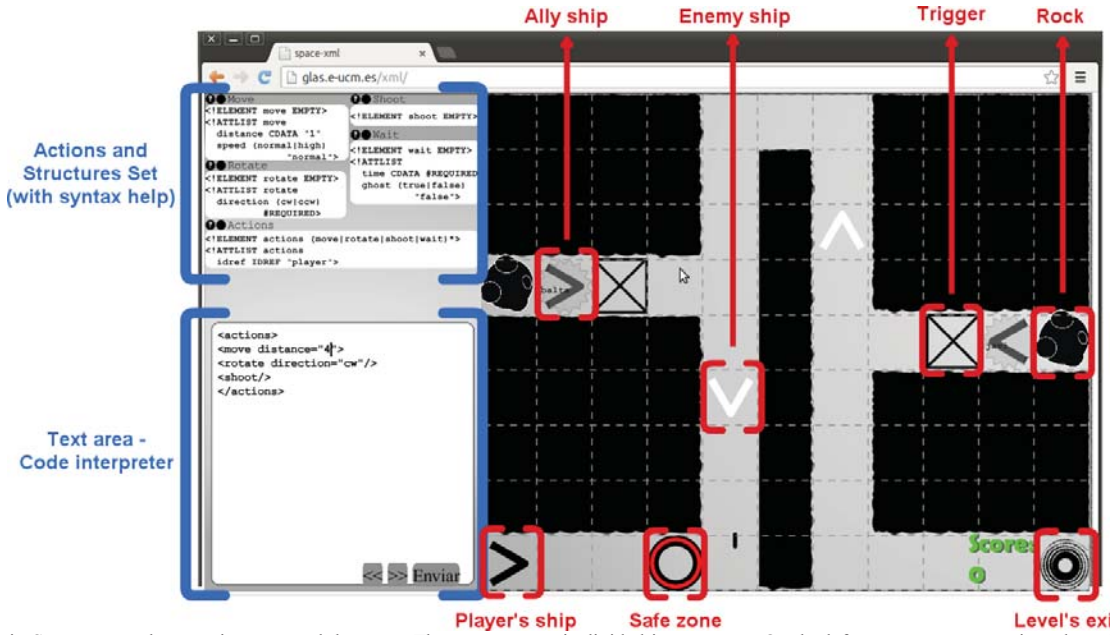

Fig. 3. Lost in Space screenshot running on a web browser. The game screen is divided in two parts. On the left part, a text area to introduce the code, and above, the available set of structures and actions. On the right, the current level, formed by different game elements.

The game has thirteen levels, and its estimated completion time is 50-60 minutes. The game architecture easily allows for adding new levels and mechanics, as well as new programming languages. To add new levels, it is only necessary to create the XML documents defining the new levels. To add support for a new programming language, for example, Python, it is only necessary to develop a new interpreter able to parse Python code.

V. Case Study I

In this section we describe the first experience using *Lost in Space* to learn XML syntax in Computer Science settings, taking advantage of the XML-based programming language the game supports. The goal of this case study was to compare game-based instruction using *Lost in Space* with traditional instruction based on lectures. The null and alternative hypotheses are defined as follows:

- H0: The effect of instruction is the same regardless of the approach (game-based vs. traditional) used.
- HA: The effect of instruction is different depending on the approach used.

A. Methods, Participants and Settings

Students were randomly assigned to either experimental or control groups (A and B respectively). Students in the experimental group (3 females and 14 males) attended a gameplay session with the game *Lost in Space* while students in the control group (3 females and 11 males) attended a lecture supported by a PowerPoint presentation and driven by an experienced teacher. Both sessions lasted one hour and covered the same contents about XML and programming. Students in the experimental group did not get explicit guidance by any instructor, apart from basic instructions to access the game. As an extra to encourage competition, students playing the game were told to upload a screen capture with the final score to the e-learning system (Moodle).

Identical pre and post tests were conducted to compare obtained scores. Each test had two exercises scoring from 0 to 10 each (20 is the maximum total score for the test). In the first exercise students were asked to identify syntactic errors in an XML document. In the second exercise students were asked to write a small XML document conformant to a given Document Type Definition (DTD).

Tests were anonymous (a unique untraceable code was used to pair pre and post test for each student), and students had 15 minutes to complete each test. At the end of the session, students in both groups were also asked to rate the educational experience using a 5-point Likert scale.

B. Results

Fig. 4 compares results of pre and post tests for groups A (experimental) and B (control). Table II shows a more detailed summary of the final results, broken down by questions.

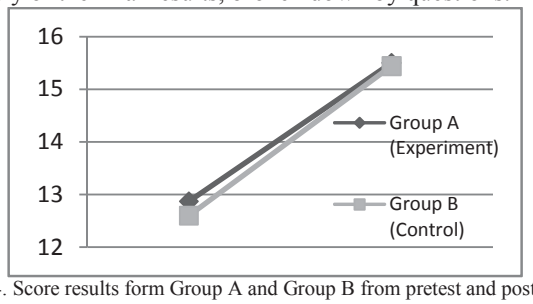

Fig. 4. Score results form Group A and Group B from pretest and posttest

In the pre test, students scored 12.87 and 12.60 on average on groups A and B respectively, being this difference not statistically significant after running an unpaired T-Test ($p=0.84$) and a Independent-Samples Mann-Whitney U-Test ($p=0.968$). Therefore we conclude that both groups had equivalent initial knowledge on the subject.

TABLE II
RESULTS FROM CASE STUDY I

Exercise	Pre / Post	Group A	Group B
Q1 (over 10)	Pre test	7.38 ± 1.56	7.12 ± 1.81
	Post test	8.29 ± 1.17	8.35 ± 1.63
	Gain (Post-Pre)	0.91 ± 1.30 **(+9.1%)**	1.23 ± 1.96 (+12.3%)
Q2 (over 10)	Pre test	5.49 ± 2.30	5.47 ± 2.66
	Post test	7.20 ± 2.26	7.09 ± 2.16
	Gain (Post-Pre)	1.71 ± 2.49 **(+17.1 %)**	1.61 ± 1.65 (+16.15 %)
Total =Q1+Q2 (over 20)	Pre test	12.87 ± 3.05	12.60 ± 4.08
	Post test	15.50 ± 2.87	15.44 ± 3.57
	Total gain	2.62 ± 2.91 **(+13.1 %)**	2.83 ± 2.99 (+14.5 %)

Both groups scored higher in the post test. Group A registered a score increase (post-pre) of 13.1% and group B an increase of 14.5% on average. Differences between post and pre tests were found statistically significant on both groups after running a paired T-Test for each group (p=0.002 for group A, p=0.003 for group B) and a related-samples Wilcoxon Signed Rank test (0.002 and 0.009 respectively).

Differences in the post tests across groups were found not statistically significant after running an unpaired T-Test (p=0.960) and a Mann-Whitney U-Test (p=0.858), indicating that both groups ended up with a similar knowledge level. As a consequence, we fail to reject the null hypothesis.

We also observe that the score increase was higher in the second exercise (writing an XML document) than in the first one (finding errors in an XML document) for both groups. However, in none of the groups this difference was found statistically significant (Group A: T-Test p=0.237, Related-Samples Wilcoxon Signed Rank Test p=0.344; Group B: T-Test p=0.530, Wilcoxon Signed Rank Test p=0.706).

There are also differences in the self-reported satisfaction. It is high in both groups (medians above or equals to 4), although it is higher in group A (game group), with more than 76% of top score (5/5) responses.

C. Discussion

Results show that students' scores increased both after traditional instruction and game-based instruction. Post test scores were equally high for both (near 15 over 20 on avg.), showing little not significant differences, which suggests that both approaches are appropriate for learning XML.

The two approaches were similar in content (i.e., in both sessions the educational content covered was similar); however, teaching approaches were different. While in the instruction, teacher explicitly presented the XML foundations –using a PowerPoint presentation–, in the game, XML content was implicit, i.e., the game did not present any formal content to the player with text or any other direct explanation: the students build their own knowledge through active play.

The average effect of instruction (13-14%) is not very high. However, this may be the consequence of the reduced exposure to instruction (<50 minutes) and the high pre test scores of the participants (around 12.5 points over 20).

There was no difference in the pre test score between groups, which allows us to discard any potential bias introduced during the randomization process.

Finally, students in the experimental group were more satisfied with the experience than the students in the control group. This is not because the satisfaction in the control group was low but a consequence of the outstanding satisfaction rates achieved in the experimental group. This finding is consistent with researchers' observations during the sessions, who noticed deep engagement in students in the experimental group. Researchers observed abnormally frequent interaction between peers, who vigorously competed to get the highest possible score. Students also kept playing after the class.

The different engagement observed in the two groups may suggest that overall *Lost in Space* was a better instructional approach, although it did not yield better results than traditional instruction in terms of knowledge acquisition.

VI. CASE STUDY II

We designed a second case study to explore if the effects of using the game for instruction with a different student population are consistent to those observed in the previous experiment. This will help us discuss on the scalability of the game model proposed by analyzing the size of the potential target audience that could use the game to learn programming.

In the second study we replicated the gameplay session (instruction delivered to group A) described in section VI with college students enrolled in a social sciences degree, who had no previous programming background (group C) and therefore were expected to obtain a significant lower pre-test score. The null and alternative hypotheses are described as follows:

- H0: The score increase factors (post-pre) of the two game-based instruction groups (A vs C) with different programming backgrounds are equal.
- HA: The score increase factors are not equal.

A. Method, Participants and Settings

Group C was composed by 13 students (5 males and 8 females) from a Degree in Information and Documentation (social sciences), who had not received previous instruction on computer programming and had no technical background.

We replicated the experimental design described in section VI, using the score increase (obtained as the difference in score obtained between pre and post test) as the independent variable that estimates the knowledge gain about XML syntax.

B. Results

Figure 5 shows a high-level view of the results for group C compared to group A, while Table III provides insight on these results. These data indicate that the initial knowledge of students is lower than in groups A and B, who had computer programming background, as initially expected. The difference between pre-test scores was found statistically significant after running an unpaired T-Test (p=0.001) and an Independent-Samples Mann-Whitney U-Test (p=0.002).

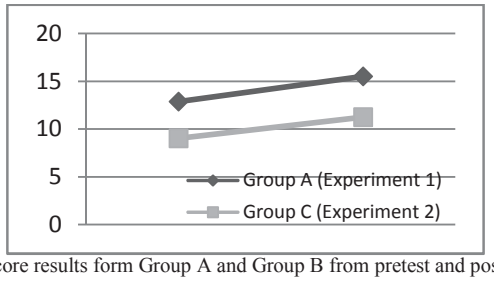

Fig. 5. Score results form Group A and Group B from pretest and posttest

Students' score was higher in the post test than in the pre-test, showing an average increment of 10.95%. This difference was found statistically significant after running a paired-samples T-Test (p=0.004) and a Related-Samples Wilcoxon Signed Rank Test (p=0.011), which suggests that the game was also effective for group C.

Compared to group A, both groups showed similar total score increments, being slightly higher for group A (2.62 ± 2.91) than for group C (2.19 ± 2.91 for group C). This difference was not found statistically significant after running an unpaired T-Test (p=0.650) and an Independent-Samples Mann-Whitney U-Test (p=0.662), allowing us to retain the null hypothesis. It suggests that the effect of game-based instruction was similar for all students regardless their previous computer programming background.

However, results are not totally equivalent in both groups, as differences across exercises are significant for group C. In this group, students did not improve their score for the first exercise (-1.3% increase on average), while their performance in the second exercise increased a 23.2% on average.

Finally, students valued the experience similarly to group A, with an average score of 4.86 (over 5).

TABLE III
RESULTS FROM CASE STUDY II

Exercise	Pre / Post	Group C	Group A
Q1 (over 10)	Pre test	6.10 ± 1.76	7.38 ± 1.56
	Post test	5.98 ± 1.84	8.29 ± 1.17
	Gain (Post-Pre)	-0.13 ± 1.30 **(-1.3%)**	0.91 ± 1.30 **(+9.1%)**
Q2 (over 10)	Pre test	2.91 ± 2.03	5.49 ± 2.30
	Post test	5.24 ± 2.51	7.20 ± 2.26
	Gain (Post-Pre)	2.32 ± 2.22 **(+23.2 %)**	1.71 ± 2.49 **(+17.1 %)**
Total =Q1+Q2 (over 20)	Pre test	9.02 ± 2.63	12.87 ± 3.05
	Post test	11.22 ± 3.73	15.50 ± 2.87
	Total gain	2.19 ± 2.91 **(+10.95 %)**	2.62 ± 2.91 **(+13.1 %)**

C. Discussion

Group C started from a lower level than Group A, which can be a consequence of their lack of programming background. However, data suggest that students in group C increased their knowledge after playing the game in a similar way to group A. However, an interesting finding that deserves further discussion is that students in group C only improved scores for the second exercise.

Both exercises had different mechanics. In the first exercise students had to identify syntactic and semantic errors in a fragment of an XML document. To complete this exercise, they had to *be aware* of the syntactic and semantic rules to form valid XML documents. In the second exercise students had to write a short XML document. In this case, students needed to have the procedural skills to write XML documents. While students have to apply syntactic and semantic rules to reach a valid solution, they do not need to be aware of what these rules look like - they just need to apply them.

That may explain the programming background of group A allowed them to infer the syntactic and semantic rules behind XML after practicing with the game, even if these rules were never explicitly presented to them. In contrast, students in group C were not able to make that inference on their own, probably as a consequence of their lack of computer programming background.

VII. CONCLUSIONS

In this paper we discussed how educational videogames can help addressing some of the challenges related to computer programming instruction. Building upon the extensive literature on this topic, we identified the need to find more scalable game-based instruction paradigms, given the increasing interest in providing computer programming instruction to a wider sector of the population, including not only computer science students but also college students of different disciplines and even kids. As a response, we propose a flexible game architecture, supported by an extensible game engine and game model, to generate fun and entertainment games that can be extended to cover different languages and suit diverse target audiences.

We developed a game using this game engine (*Lost in Space*), which was used for XML instruction in two different college settings (computer science and social sciences respectively). In these experiences the game was well accepted by the students, and we also observed that they deeply engaged in gameplay. Moreover, data collected suggest that they learned with this kind of game-based instruction in a similar way to traditional instructional methods regardless of their background. However, a potential limitation of the approach for students with no computer programming background that was identified. Data collected suggest that they were not able to infer the syntax of the language on their own, as syntactic rules were never explicitly provided to the students. However, students with programming background were able to make that inference. Instructors willing to use this approach should take this finding into consideration and design a strategy to help students to construct the explicit representation of the knowledge acquired, using debriefing sessions or closer tutoring, for example.

We think that our approach is adequate to introduce computer programming in general as well as new programming languages. And, in some cases, it can be used as a complement, but it also can be used as a whole educational resource.

ACKNOWLEDGMENT

Special thanks to Ricardo García-Mata from the UCM statistical service for his assistance with the data analysis and beta testers from the e-UCM group who helped us to improve the game.

REFERENCES

[1] T. W. Malone and M. R. Lepper, "Making learning fun: A taxonomy of intrinsic motivations for learning," R. E. Snow and M. J. Farr, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 223–253.

[2] J. Robertson and J. Good, "Story creation in virtual game worlds," *Communications of the ACM*, vol. 48, no. 1, pp. 61–65, 2005.

[3] A. E. Rais, S. Sulaiman, and S. M. Syed-Mohamad, "Game-based approach and its feasibility to support the learning of object-oriented concepts and programming," in *2011 Malaysian Conference in Software Engineering*, 2011, pp. 307–312.

[4] M. B. MacLaurin, "The design of kodu: a tiny visual programming language for children on the Xbox 360," *ACM Sigplan Notices*, vol. 46, no. 1, pp. 241–245, 2011.

[5] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[6] I. F. de Kereki, "Scratch: Applications in Computer Science 1," in *2008 38th Frontiers in Education Conference*, 2008, pp. T3B–7–T3B–11.

[7] W.-K. Chen and Y. C. Cheng, "Teaching Object-Oriented Programming Laboratory With Computer Game Programming," *IEEE Transactions on Education*, vol. 50, no. 3, pp. 197–203, Aug. 2007.

[8] M. Chang, "Web-Based Multiplayer Online Role Playing Game (MORPG) for Assessing Students' Java Programming Knowledge and Skills," in *2010 Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning*, 2010, pp. 103–107.

[9] P. Sancho, J. Torrente, and B. Fernández-Manjón, "Do Multi-User Virtual Environments Really Enhance Student's Motivation in Engineering Education?" FIE Conference, San Antonio, TX, USA, 2009.

[10] T. Mitamura, Y. Suzuki, and T. Oohori, "Serious games for learning programming languages," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2012, pp. 1812–1817.

[11] R. Ibrahim, J. Semarak, K. Lumpur, and A. Jaafar, "Using educational games in learning introductory programming: A pilot study on students' perceptions," in *2010 International Symposium on Information Technology*, 2010, pp. 1–5.

[12] S. H. A. Hamid and L. Y. Fung, "Learn Programming by Using Mobile Edutainment Game Approach," in *2007 First IEEE International Workshop on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL'07)*, 2007, pp. 170–172.

[13] N. Masso and L. Grace, "Shapemaker: A game-based introduction to programming," in *2011 16th International Conference on Computer Games (CGAMES)*, 2011, pp. 168–171.

[14] I. Paliokas, C. Arapidis, and M. Mpimpitsos, "PlayLOGO 3D: A 3D Interactive Video Game for Early Programming Education: Let LOGO Be a Game," in *2011 Third International Conference on Games and Virtual Worlds for Serious Applications*, 2011, pp. 24–31.

[15] H. C. Jiau, J. C. Chen, and K.-F. Ssu, "Enhancing Self-Motivation in Learning Programming Using Game-Based Simulation and Metrics," *IEEE Transactions on Education*, vol. 52, no. 4, pp. 555–562, Nov. 2009.

[16] J. Chen, "Flow in games (and everything else)," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.

[17] J. P. Gee, "What video games have to teach us about learning and literacy," *Computers in Entertainment*, vol. 1, no. 1, p. 20, Oct. 2003.