Building a Scalable Game Engine to Teach Computer Science Languages

Ángel Serrano-Laguna, Javier Torrente, Borja Manero Iglesias, and Baltasar Fernández-Manjón, *Senior Member, IEEE*

***Abstract*— Every day, more people are interested in learning computer science (CS), either to improve their skill set to apply for new jobs or just for personal growth. The sector of the population looking for instruction on these subjects has increased and diversified. We need new tools that appeal to this wider audience, and game-based learning is one of the most promising approaches at the moment. There is a need for more scalable game-based instruction paradigms that can be easily adapted to different levels of complexity and content related to CS (different programming languages, different programming paradigms, and so on). Throughout this paper, we present a flexible and scalable architecture to create videogames for learning CS languages. The architecture is based on the idea that students control the game using small pieces of text written in some CS language. The keys of the scalability of our approach are: 1) it separates the CS language used to write the programs from the game design and 2) the game model provides a system of levels that allows incremental learning of CS language structures. As validation and implementation of our approach, we developed Lost in Space, an educational videogame to teach the XML markup language. In this game, the player travels through several levels, guiding a spaceship by introducing small pieces of XML in a text console. Players can move and rotate the ship among other power-ups that get unblocked as they advances in the game. The game was tested with undergraduate students from CS and social sciences, by comparing it with traditional instruction (i.e., a teacher with a slides presentation). Students who played the game were much more engaged than those who attended the lecture, showing a more active attitude throughout 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, although 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, and seems to be more appealing to new comers than traditional instruction.**

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

Manuscript received June 4, 2015; revised August 30, 2015; accepted September 1, 2015. Date of current version November 4, 2015. This work was supported in part by the Regional Government of Madrid under Grant eMadrid S2013/ICE-2715, in part by the Complutense University of Madrid under Grant GR3/14-921340, in part by the Ministry of Education, Culture and Sport, Spain, under Grant TIN2013-46149-C2-1-R and Grant FPU AP2012-4310, and in part by the European Commission under Grant SEGAN 519332-LLP-1-2011-1-PT-KA3-KA3NW and Grant H2020-RAGE 644187.

The authors are with the ISIA Department, Complutense University of Madrid, Madrid 28040, Spain (e-mail: angel.serrano@fdi.ucm.es; jtorrente@fdi.ucm.es; balta@fdi.ucm.es; borja@sip.ucm.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/RITA.2015.2486386

I. Introduction

VIDEOGAMES have been used as successful educational tools in many different fields and topics: engineering and problem solving [1], learning foreign languages [2], health [3] and even theatre [4]. The idea of using videogames to support the learning of computer programming is not new, as it dates back almost to the origins of instructional games [5]. 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, and new educational programs trying to introduce programming in early educational stages [6].

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 facilitating the use of 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, and 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 flexible and scalable game engine to create games for learning programming that could be used by students with different backgrounds, as an extended version of the paper published in [7]. 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 for reuse of 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,* which supports the learning of several programming languages. We have conducted two case studies with college students from different backgrounds to evaluate the effectiveness of the game for learning computer programming. The first case study involved a group of computer science students, with previous knowledge of programming. The second case study was conducted with social sciences students, with no programming background.

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 [8], [9]. For example, in [10] researchers use the state-machine nature of board games to simplify the introduction of some programming concepts.

In other approaches, students create simple program snippets that control characters and objects in a game environment, usually through a visual or simplified programming language. These snippets can have several purposes: beat a level in a game, create interactive content or animate a simple scene with several characters. The visual condition of the results obtained facilitates the feedback and gets students rapidly engaged in programming.

Multiple tools have been developed around this paradigm, such as GameMaker [11], Alice [12], Greenfoot [13], Microsoft's Kodu [14], Scratch [15] and other self-built systems [16]. The literature is full of experiences where this paradigm has been successfully applied. For example, de Kereki [17] reports effective use of Scratch as a motivational tool in an introductory programming course. In other example, Chen and Cheng [18] 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, which is an approach more similar to ours. In many of these games students do not write programs: the videogames are used as mere containers of theoretical content about programming, in which the only goal is to pursue an increase in students' motivation and interest for the activity. However, the lack of active programming is a limitation of the approach, since writing programs is an essential activity to learn programming.

Several examples exist in the research using this approach. In [19] authors report the use of a role-playing game where quests are directly related to programming concepts (e.g., answering questions about some programming language feature). In a similar approach, the use of a game environment provided a significant increase in student motivation towards the subject [20] and student performance. In [21] 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 [22] a game is used to teach C with crosswords puzzles and duck shot games. Finally, in [23] 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 [24] 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 [25] 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 [21] 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 [26], 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. In the following sections, we present our game engine which tries to tackle some of these drawbacks.

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: Writing code is essential to learn programming, thus, students must create programs to overcome the different levels in the game. This enables a learning-by-doing approach.

2) Promote Reflection, Avoid Time Pressure: We propose a slow paced gameplay, based on a turn-based strategy where the time has no influence on the final score. We want to avoid game mechanics that require fast reaction of the players. 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 simple enough so that a valid syntactic construction, representing the action, can be built with as many programming languages as possible. This adds scalability as it allows for the reuse of 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 programming 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 [27].

5) A Clear Goal Is Set Up for Each Level: Clear goals are a desirable feature for any good video game [28], and it also facilitates writing programs. Puzzles and levels will be designed in such a way that multiple solutions are possible, so players can find creative solutions to them.

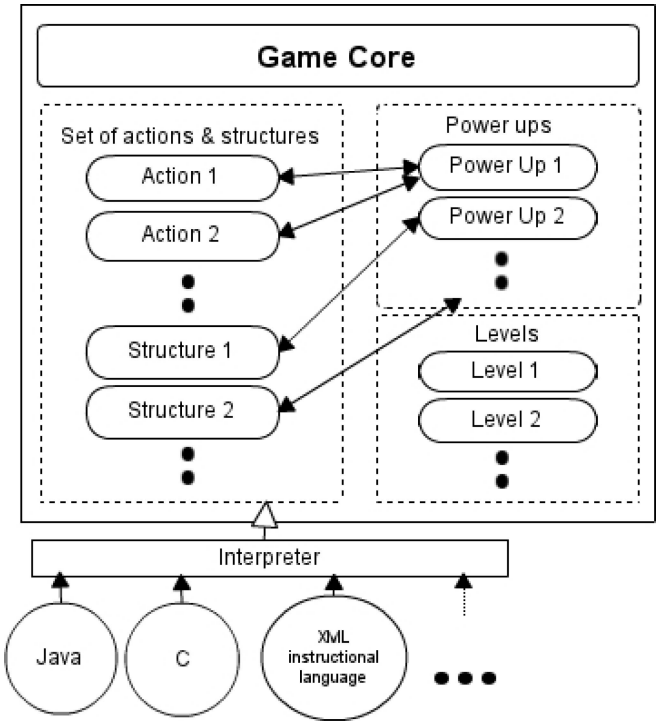

Fig. 1. Engine architecture outline. The set of actions and structures available is directly related to the power-ups unlocked in the game levels. An interpreter translates the programs to the set of actions and structures.

6) Use Scores to Promote Competition Between Peers and to Provide a Sense of Progress: Scores are used as simple metaphors to engage students and at the same time to reflect user performance. Scores help users to compare their results to others.

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. Ideally, the game core should be "fixed," i.e. it should not vary when adding new programming languages.

2) Set of Actions and Structures: This component contains the set of finite actions and control structures that players can use within the game. This set can be extended to fulfill new needs derived from the inclusion of new programming languages.

3) Interpreter: The interpreter translates the programming code typed 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 separately, in a proprietary descriptive 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 (e.g. it can be interesting to have several versions of the game with different durations, to fit one-hour lectures, two-hour lectures, etc).

```
<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>
```

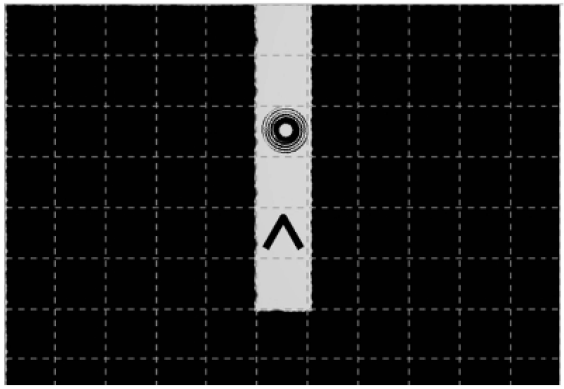

Fig. 2. A XML document defining a level of *Lost in Space*. The output level is shown below.

Levels are short, easy at first, and their complexity increases 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, and all of them are evenly spread throughout all the levels in the game.

On the player side, these power-ups unlock new programming structures/elements that the players 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).

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 present new programming languages to students from different fields and different background knowledge. Fig. 3 shows a screenshot of the game, with the main elements highlighted.

[1]Available at *http://bit.ly/lostinspacexml* at date August 20, 2015. Source code available in *https://github.com/anserran/lostinspace.*

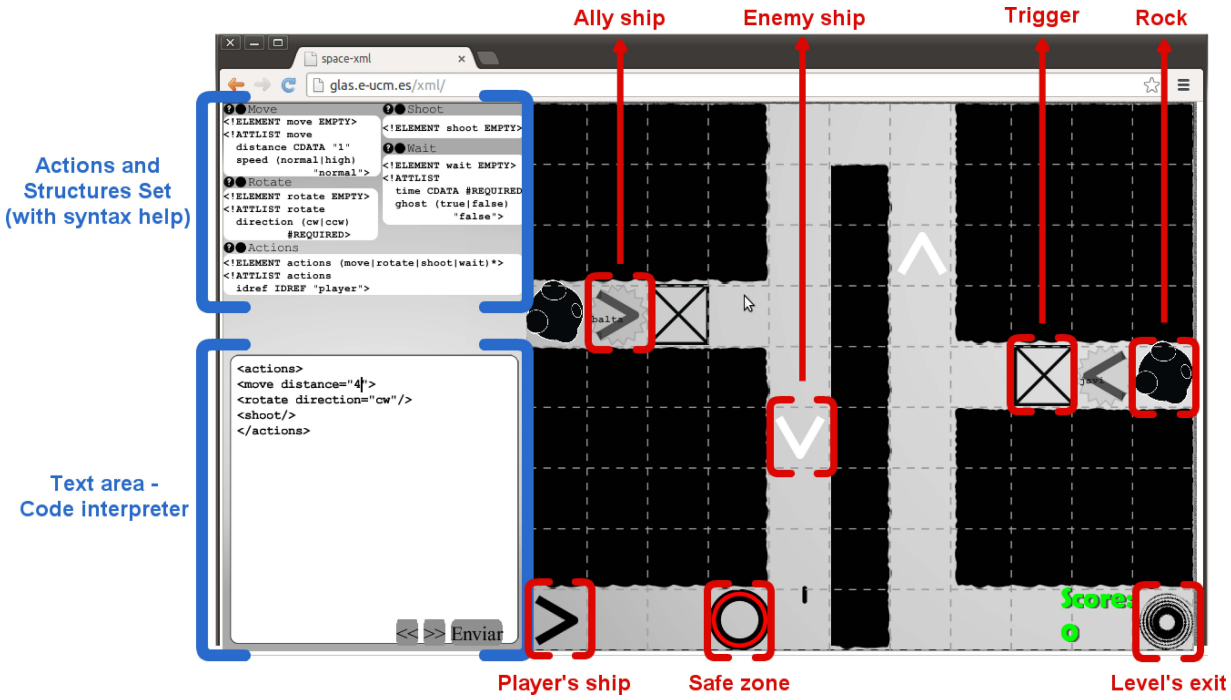

Fig. 3. *Lost in Space* screenshot running on a web browser. The game screen is divided into two parts. On the left part, a text area to type the code, and above, the available set of structures and actions. On the right, the current level, formed by different game elements.

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 and use them to help beat the level) or *enemies* (they will try to destroy the player's spaceship and their behavior cannot be controlled by the user), 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, open walls, 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 has several atomic operations available. These operations affect the main ship and the allied ships, and are unlocked in the course of the game, every time a new power-up is collected. In the last level the player can use a total of 5 instructions:

- *Move*: translates the spaceship a number of units (squares in the grid) in the direction it is facing.
- *Rotate*: changes the direction of the spaceship 90 degrees, clockwise or counterclockwise.
- *Shoot*: shoot a missile that can destroy enemies and rocks.
- *Wait:* inserts a waiting time between two actions.
- *Disappear*: makes the spaceship invincible for a short period of time.

The gameplay 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, modifying the state of the player's spaceship. Otherwise it reports the error back to the player. As the player advances in the game, power-ups appear at certain levels. Once collected, power-ups are unlocked, and appear at the power-ups bar, on the top left side of the screen.

TABLE I
EXAMPLES OF 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>`

Table I provides some examples of program snippets 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 a markup language and not a programming 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.

The game has thirteen levels, and its estimated completion time is 50-60 minutes, since its original use was for a one-hour lecture. The game architecture easily allows 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 the 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. In this case study, we used the game to completely replace a lecture about the XML markup language with half of the students of a classroom. The goal of this case study was to compare game-based instruction using *Lost in Space* with traditional instruction based on lectures (slides presentation with a teacher). 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 and evenly distributed into either the experimental group or the control group–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 given 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. They only received some basic instructions to access the web page where the game was hosted. Students playing the game were told to upload a screen capture with the final score to the course learning management system (Moodle) as a way to encourage competition.

Identical pre and posttests were conducted to compare obtained scores in both groups. Each test had two exercises scoring from 0 to 10 each (20 is the maximum total final score for the test).

In the first exercise students were asked to identify syntactic errors in an XML document. Marking a correct error increased the total score in the exercise. Marking a non-existing error decreased the score.

In the second exercise students were asked to write a small XML document conformant to a given Document Type Definition (DTD). In order to score the exercise, three things where evaluated: 1) the ability of the student constructing valid pieces of XML, 2) the compliance of the document with the DTD, and 3) the completeness of the written document, using as reference the document the exercise asked for.

Tests were anonymous –a unique untraceable code was used to pair pre and post tests 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.

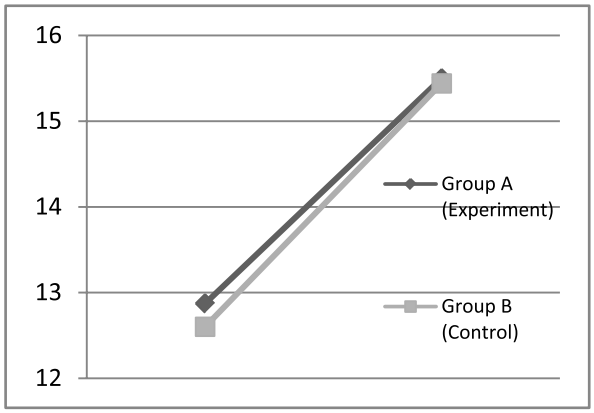

Fig. 4. Score results from Group A and Group B from pretest and posttest.

TABLE II
Results From Case Study I

Exercise	Pre / Post	Group A	Group B
Q1 (over 10)	Pretest	7.38 ± 1.56	7.12 ± 1.81
	Posttest	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)	Pretest	5.49 ± 2.30	5.47 ± 2.66
	Posttest	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)	Pretest	12.87 ± 3.05	12.60 ± 4.08
	Posttest	15.50 ± 2.87	15.44 ± 3.57
	Total gain	2.62 ± 2.91 **(+13.1 %)**	2.83 ± 2.99 (+14.5 %)

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.

In the pretest, students scored 12.87 and 12.60 on average in groups A and B respectively; this difference was not statistically significant using an unpaired T-Test ($p = 0.84$) and an Independent-Samples Mann-Whitney U-Test ($p = 0.968$). Therefore we conclude that both groups had equivalent initial knowledge of the subject.

Both groups scored higher in the posttest. 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 pretests were found to be statistically significant in 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 posttests 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 to be 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 $p = 0.706$).

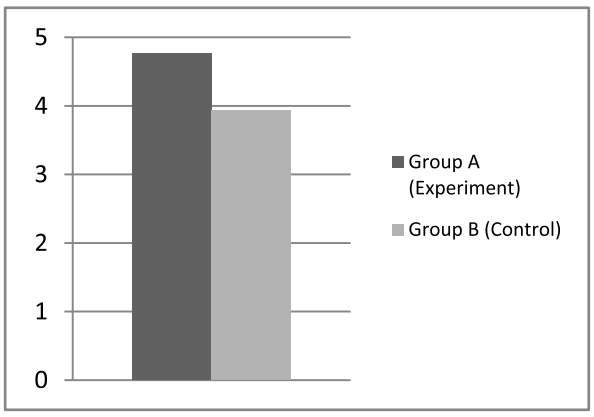

Fig. 5. Self-reported satisfaction form Group A and Group B from posttest.

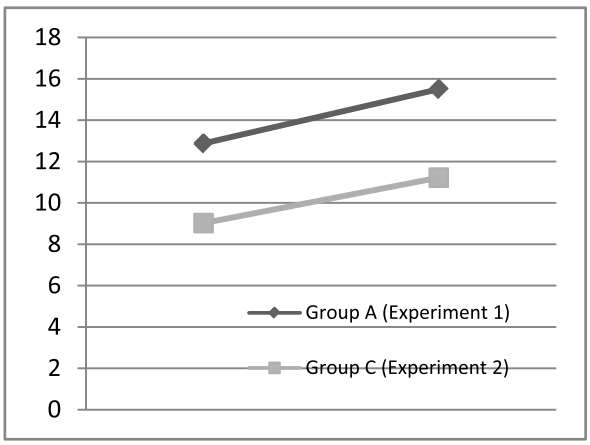

Fig. 6. Score results form Group A and Group B from pretest and posttest.

There are also differences in self-reported satisfaction. Figure 5 shows the results for Group A and Group B. It is high in both groups (Group A = 4.76 and Group B = 3.92), 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. Posttest scores were equally high for both (near 15 over 20 on avg.), showing few significant differences, which suggests that both approaches are appropriate for teaching/learning XML.

The two approaches were similar in content (i.e. in both sessions the educational content covered was similar, in fact, game content was designed around the slides presentation); however, teaching approaches were different. During instruction, the teacher explicitly presented the XML foundations –using a PowerPoint presentation–, while during 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 two causes: the reduced exposure to instruction (less than 50 minutes) and the high pretest scores of the participants (around 12.5 points over 20).

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

Finally, the 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.

If we consider all factors, 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 whether the effects of using the game for instruction –knowledge increase and high motivation– with a different student population are consistent with 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 significantly lower pretest 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 made up of 13 students (5 males and 8 females) from a Degree in Information and Documentation (social sciences), who had not received previous instruction in 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 pretest and posttest) as the independent variable that estimates the knowledge gain of XML markup language.

B. Results

Figure 6 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 pretest scores was found statistically significant after

TABLE III
RESULTS FROM CASE STUDY II

Exercise	Pre / Post	Group C	Group A
Q1 (over 10)	Pretest	6.10 ± 1.76	7.38 ± 1.56
	Posttest	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)	Pretest	2.91 ± 2.03	5.49 ± 2.30
	Posttest	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)	Pretest	9.02 ± 2.63	12.87 ± 3.05
	Posttest	11.22 ± 3.73	15.50 ± 2.87
	Total gain	2.19 ± 2.91 **(+10.95 %)**	2.62 ± 2.91 **(+13.1 %)**

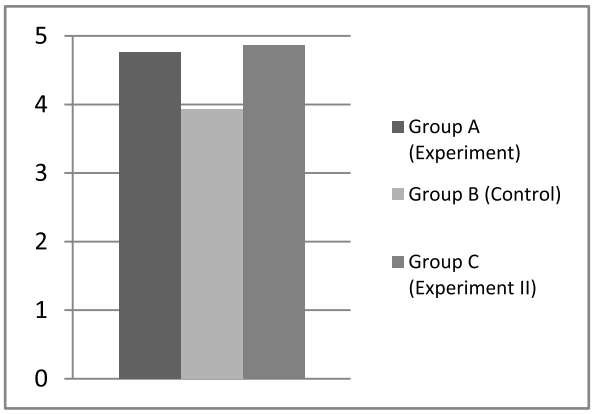

Fig. 7. Self-reported satisfaction from Group A, B and C.

running an unpaired T-Test ($p = 0.001$) and an Independent-Samples Mann-Whitney U-Test ($p = 0.002$).

Students' score was higher in the posttest than in the pretest, 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 23.2% on average.

Finally, Figure 7 shows the average self-reported satisfaction of Group C, along with the average satisfaction of Group A and Group B. Students valued the experience similarly to Group A, with an average score of 4.86 (over 5).

C. Discussion

Group C started from a lower level than Group A, which is likely 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, i.e. they needed to understand the concept of markup/programming language, and that all of them have a syntax in its foundation. 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 be explained by the programming background of group A, which 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.

Finally, group C reported the highest satisfaction of the three groups involved in the experiment. According to the observations made by the instructors present during the experiment, the reasons for this difference are the following: 1) group C did not have any similar past experience; and 2) their low exposure to videogames in their daily life increased the 'surprise effect' in the session, which lead to greater satisfaction, in contrast to group A, which was probably more exposed to videogames due to their CS background.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we discussed how educational videogames can help address some of the challenges related to computer programming instruction, and we made an extensive review on how videogames and learning programming are coupled together in the literature.

Building upon the extensive literature on this topic, we identified the need to find more scalable and flexible game-based instruction paradigms, given the increasing interest in providing computer programming instruction to a wider sector of the population, a population that not only includes 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, engaging and entertaining games that can be extended to cover different languages, suit diverse target audiences and fill different classroom timespans.

We developed a game using this game engine (*Lost in Space*), which was used for XML markup language 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 was identified.

The data collected suggests that students from social science were not able to infer the syntax of the language on their own, as they do not have any previous programming background and syntactic rules were never explicitly provided to them. 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.

However, it seems reasonable to assert that, whatever the background of the target audience, our approach can be a rewarding and entertaining way to learn and teach programming.

We think that our approach is adequate to introduce computer programming in general as well as new programming languages in particular. And, even if it was initially designed as an additional strategy to support learning, it can also be used as the main educational resource in some contexts.

Finally, we would like to point out that this approach can have several extra advantages if we also consider the game engine as an assessment tool.

We deployed *Lost in Space* in a web server with some basic user tracking. This helped us discover helped us discover that some of the students kept playing several days after the experiment was over. This gave us the idea that a more advanced tracking of the gameplay of each player can help us to assess and improve the learning process [29].

The building blocks of the game engine –mainly levels and power-ups– mark clear milestones that we can track and assess to have a deeper insight of how players are learning programming. This will also help us to improve the game engine and game model.

ACKNOWLEDGMENTS

The authors would like to thank Ricardo García-Mata from the UCM statistical service for his assistance with the data analysis. They would also like to thank all beta testers from the e-UCM group who helped them to improve and polish the game, and to all the students that participated in the experiments.

REFERENCES

[1] J. A. Betz, "Computer games: Increase learning in an interactive multidisciplinary environment," *J. Edu. Technol. Syst.*, vol. 24, no. 2, pp. 195–205, 1996.

[2] W. L. Johnson, N. Wang, and S. Wu, "Experience with serious games for learning foreign languages and cultures," in *Proc. SimTecT Conf.*, 2007, pp. 1–7.

[3] T. Baranowski, R. Buday, D. I. Thompson, and J. Baranowski, "Playing for real: Video games and stories for health-related behavior change," *Amer. J. Preventive Med.*, vol. 34, no. 1, pp. 74–82, 2008.

[4] B. Manero, C. Fernández-Vara, and B. Fernández-Manjón, "E-learning a escena: De La Dama Boba a Juego Serio," *IEEE Revista Iberoamericana Tecnologias Aprendizaje*, vol. 1, no. 1, pp. 51–58, 2013.

[5] 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, USA: Lawrence Erlbaum, 1987, pp. 223–253.

[6] P. Doerschuk, J. Liu, and J. Mann, "An INSPIRED game programming academy for high school students," in *Proc. Frontiers Edu. Conf.*, Oct. 2012, pp. 1–6.

[7] Á. Serrano-Laguna, J. Torrente, B. Manero, and B. Fernandez-Manjon, "A game engine to learn computer science languages," in *Proc. IEEE Frontiers Edu. Conf.*, Oct. 2014, pp. 1–8.

[8] K. Doss, V. Juarez, D. Vincent, P. Doerschuk, and J. Liu, "Work in progress—A survey of popular game creation platforms used for computing education," in *Proc. IEEE Frontiers Edu. Conf.*, Oct. 2011, pp. F1H-1–F1H-2.

[9] T. Barik, M. Everett, R. E. Cardona-Rivera, D. L. Roberts, and E. F. Gehringer, "A community college blended learning classroom experience through artificial intelligence in games," in *Proc. IEEE Frontiers Edu. Conf. (FIE)*, Oct. 2013, pp. 1525–1531.

[10] J. Heliotis, I. Bezáková, and S. Strout, "Programming board game strategies in CS2," in *Proc. IEEE Frontiers Edu. Conf. (FIE)*, Oct. 2013, pp. 4–5.

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

[12] 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 *Proc. 5th Malaysian Conf. Softw. Eng.*, Dec. 2011, pp. 307–312.

[13] P. Doerschuk, V. Juarez, J. Liu, D. Vincent, K. Doss, and J. Mann, "Introducing programming concepts through video game creation," in *Proc. IEEE Frontiers Edu. Conf. (FIE)*, Oct. 2013, pp. 523–529.

[14] 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.

[15] M. Resnick *et al.*, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[16] S. Arakawa and S. Yukita, "An effective agile teaching environment for Java programming courses," in *Proc. 36th Annu. Frontiers Edu. Conf.*, Oct. 2006, pp. 13–18.

[17] I. F. de Kereki, "Scratch: Applications in computer science 1," in *Proc. 38th Annu. Frontiers Edu. Conf.*, Oct. 2008, pp. T3B-7–T3B-11.

[18] W.-K. Chen and Y. C. Cheng, "Teaching object-oriented programming laboratory with computer game programming," *IEEE Trans. Educ.*, vol. 50, no. 3, pp. 197–203, Aug. 2007.

[19] M. Chang and Kinshuk, "Web-based multiplayer online role playing game (MORPG) for assessing students' Java programming knowledge and skills," in *Proc. 3rd IEEE Int. Conf. Digit. Game Intell. Toy Enhanced Learn.*, Apr. 2010, pp. 103–107.

[20] P. Sancho, J. Torrente, and B. Fernández-Manjón, "Do multi-user virtual environments really enhance student's motivation in engineering education?" in *Proc. 39th IEEE Frontiers Edu. Conf.*, San Antonio, TX, USA, Oct. 2009, pp. 1–6.

[21] T. Mitamura, Y. Suzuki, and T. Oohori, "Serious games for learning programming languages," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2012, pp. 1812–1817.

[22] R. Ibrahim and A. Jaafar, "Using educational games in learning introductory programming: A pilot study on students' perceptions," in *Proc. Int. Symp. Inf. Technol.*, Jun. 2010, pp. 1–5.

[23] S. H. Ab Hamid and L. Y. Fung, "Learn programming by using mobile edutainment game approach," in *Proc. 1st IEEE Int. Workshop Digit. Game Intell. Toy Enhanced Learn. (DIGITEL)*, Mar. 2007, pp. 170–172.

[24] N. Masso and L. Grace, "Shapemaker: A game-based introduction to programming," in *Proc. 16th Int. Conf. Comput. Games (CGAMES)*, Jul. 2011, pp. 168–171.

[25] I. Paliokas, C. Arapidis, and M. Mpimpitsos, "PlayLOGO 3D: A 3D interactive video game for early programming Education: Let LOGO be a game," in *Proc. 3rd Int. Conf. Games Virt. Worlds Serious Appl.*, May 2011, pp. 24–31.

[26] H. C. Jiau, J. C. Chen, and K.-F. Ssu, "Enhancing self-motivation in learning programming using game-based simulation and metrics," *IEEE Trans. Educ.*, vol. 52, no. 4, pp. 555–562, Nov. 2009.

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

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

[29] Á. Serrano-Laguna and B. Fernández-Manjón, "Applying learning analytics to simplify serious games deployment in the classroom," in *Proc. IEEE Global Eng. Edu. Conf. (EDUCON)*, Apr. 2014, pp. 872–877.

Ángel Serrano-Laguna received the master's degree in computer science from the Universidad Complutense de Madrid (UCM), in 2012, where he is currently pursuing the Ph.D. degree. He is currently a full-time Researcher with the e-UCM Research Group, UCM. His research focuses on the design and implementation of serious games and the creation of tools to ease their introduction in the classroom, and the study of learning analytics techniques to assess those games.

Javier Torrente received the M.Sc. and Ph.D. degrees in computer science from the Universidad Complutense de Madrid (UCM), Madrid, Spain, in 2014. His Ph.D. research focused on how to reduce the cost of making digital games more accessible for people with a disability. He was a member of the e-UCM Research Group at UCM. He is currently a full-time Researcher with University College London, U.K. He has authored over 70 research papers in academic journals and conferences in the fields of serious games, HCI, and accessibility.

Borja Manero Iglesias received the bachelor's degrees in physics and computer science from the Universidad Complutense de Madrid (UCM), Madrid, Spain, in 1999, and the bachelor's degree in dramatic arts from the Réplika School of Theater, in 2007. He is currently a Lecturer with the Department of Software Engineering and Artificial Intelligence, UCM. His research interests focus on educational games and their use in drama plays and cinema.

Baltasar Fernández-Manjón (SM'–) is currently a Full Professor of Computer Science with the Universidad Complutense de Madrid (UCM), where he is the Director of the e-UCM e-Learning Research Group. He is also involved in the application of e-learning standards to the integration of those technologies in e-learning systems. His research interest is focused on the applications of ICT in education and in serious games and educational simulations applied to different domains (e.g., medicine, culture, and languages).