Learning Programming with Peer Support, Games, Challenges and Scratch

Roberto A. Bittencourt, David Moises B. dos Santos, Carlos A. Rodrigues, Washington P. Batista, Henderson S. Chalegre
UEFS – State University of Feira de Santana
Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana, Bahia, Brazil, 44036-900
Email: {roberto,davidmbs,carod}@uefs.br, {wstroks,hendersonchalegre}@gmail.com

Abstract—Helping college freshmen to learn basic computer programming is a longstanding research topic. Various environments, tools and languages have been developed to ease the initial steps of novice programmers. However, to be used to their full extent, such artifacts should be better combined with an appropriate learning approach. This work describes pre-term workshops offered to Computer Engineering undergraduate freshmen that combines the use of the Scratch learning environment with a learning approach based on peer support, game development and a strategy of challenge-response. Results are derived from quantitative data analysis: surveys and artifact analysis. Students evaluated the workshop well: they found it stimulant, lightweight, with good teaching, useful, well-organized, and conducive to learning. Results suggest the effectiveness of the approach to reduce initial difficulties for freshmen learning programming concepts.

Keywords—peer learning, programming, environments for novices, Scratch, game-based learning, challenges.

I. INTRODUCTION

For years, undergraduate computing programs have faced the challenge to teach programming to freshmen with no previous background in the field [1]. A large variety of concepts and skills is usually required, ranging from algorithmic thinking to expression in a formal language, from data structures to mastering an IDE [2]. To make matters worse, faculty typically adopt a formal approach that is usually detached from students’ worries and preferences.

Previous work is prolific with alternative approaches to teach college introductory programming, especially with environments geared to the novice [2], [3]. Among such environments, Scratch is one of the most popular, originally conceived to be used by both children and teenagers, and based on design decisions that make the environment more tinkerable, meaningful and social [4].

Nonetheless, even the most well designed environment may be misused if not accompanied by appropriate teaching and learning approaches. To us, students benefit more from learning environments when an active learning approach is used. For two years, we have been perfecting an approach that combines the use of Scratch, peer support, a strategy of challenge-response, and game development. We have also applied this approach by means of freshmen workshops, usually a week before the academic term starts.

In this paper, we describe one of our latest workshops, a one-week, 20-hour pre-class workshop with freshmen of a Computer Engineering program in a state university in Brazil, offered in August 2014. The workshop intended to overcome the initial difficulties with programming that students face. Three sophomore peer students guided the workshop presented to 18 freshmen. The challenges proposed led to the gradual development of an animation in Scratch, and three classic games: Pong, Bow and Arrow, and Space Invaders. One last challenge was the development of a simple calculator to allow a smooth transition from Scratch to the C language. To evaluate our approach, we used a quantitative approach based on artifact analysis, i.e., the projects students produced, and two surveys: one right after the workshop was over, and another one month after the term started, when students finished their first project in the CS1 course 1.

Students evaluated the workshop well in various criteria: they found it stimulant, lightweight, with good teaching, useful, well-organized, and conducive to learning. Students considered stimulant the active and playful learning approach. They also evaluated Scratch as having a friendly user interface and simple working logic, especially the Lego-style block fitting. The most used blocks were interface commands (Appearance), selection and repetition structures (Control), and movement commands (Motion), which were also the ones students asserted as less difficult.

Even though most students were not using Scratch after the workshop, 70% of them asserted that the tool helped a lot to learn programming. Especially regarding selection and loop structures, 80% agreed that, to some degree, the workshop helped them to develop their first project in the CS1 course. This seems remarkably relevant since these are concepts usually hard to grasp in regular courses. On the other hand, the same did not happen with the use of variables. Though used in the games, they were not the main student concern when answering the challenges.

Results point to the effectiveness of an approach based on game development, a challenge-response strategy, peer support and programming environments for novices. The gathering of empirical evidence in this work also contributes to the field, helping to better evaluate learning initiatives and tools.

This paper is organized as such: Section II discusses

1CS1 is the name usually given to the computer science introductory programming course.
the background for this paper and related work. Section III describes the research methods used. Then, Section IV presents results, followed by a discussion in Section V. Conclusions are derived in Section VI together with suggestions for future work.

II. BACKGROUND

In this section, we describe related work on causes for dropout and failure in CS1 courses, on programming environments for novices, and on guidelines and alternative initiatives in introductory programming courses.

A. Causes for dropout and failure in programming courses

Why is programming hard to learn? This question has been thoroughly pursued by the community of computing education for around five decades. Robins et al. surveyed the topic and describe differences between novice and expert programmers and between effective and ineffective novices. They also describe the complexity of the problem by devising a programming framework based on knowledge, strategies and mental models needed to design, generate and evaluate programs [5]. Program design involves knowledge of planning methods and algorithm design, strategies for planning and problem solving, and models of problem domain. Program generation requires knowledge of languages, libraries and tools, strategies for implementing algorithms and for coding, and models of a desired program. Finally, program evaluation requires knowledge of debugging and testing tools and methods, strategies for testing, tracing and repairing, and models of an actual program.

A survey among 67 higher education institutions all over the world reported a high variation in fail rates, reaching values as high as 60% [6]. Robins discusses the likely causes described by the academic literature to distinguish students between programmers and non-programmers [7]. Factors range between cognitive capacity, cognitive development, cognitive style, attitude and motivation. However, this author suggests that those factors do not actually predict student success, and proposes an alternative explanation, named learning edge momentum effect, where acquiring one concept eases the learning of other closely related concepts.

Another work investigates students’ decisions to quit the CS1 course in high school, with reasons ranging from lack of time to lack of motivation [8]. Those reasons are affected by factors such as perceived difficulty of the course, general difficulties with time management and planning studies, or the decision to prefer something else. Low comfort level and plagiarism also play a role in dropout levels. Finally, authors explain that dropout reasons accumulate, making efficient intervention require a combination of various different actions.

Lewis et al. use structural equation modelling to investigate the effect of technical and soft skills (i.e., emotional intelligence) on the affinity (i.e., satisfaction with the CS major) and the intention to quit the CS major [9]. Unexpectedly, they found that technical skills were less important than soft skills to predict affinity. They suggest incorporating soft skills in the curriculum of computing undergraduate programs.

Part of the difficulties with learning introductory programming is related to both programming languages and development environments used in the learning process. Researchers have investigated those issues [2], [3], which we shortly describe in the following.

B. Programming learning environments

Kelleher and Pausch have developed a taxonomy of programming environments and languages for novices [3]. Two large groups of environments and languages are proposed: teaching systems, which aim to help people learn to program, and empowering systems, whose goals are to empower users to build things tailored to their own needs. Inside each group, they classify environments and languages by their approach to either ease learning or to empower users.

Guzdial describes the evolution of learning environments for novice programmers [2]. He discusses three families of programming learning environments: the Logo family, the rule-based family; and the traditional programming family. He identifies three research trends: a clear trend towards tool development with traditional language syntax; students prefer to work on computational artifacts that have meaning to them; and environments and tasks should give students immediate feedback on their work.

The trends identified by Guzdial led to the development of some popular programming learning environments, where learning happens by developing and playing games and animations in microworlds, such as in Scratch [10], Greenfoot [11], Alice [12], Robocode [13] and AppInventor [14]. Uting et al. discuss the goals, mechanisms and effects of Scratch, Alice and Greenfoot, providing an interesting account of the affordances of each tool [15].

Scratch, the tool we use in this work, is based on three design concepts: being more tinkerable, more meaningful and more social [4]. It is more tinkerable because programming happens by fitting programming blocks together, just like blocks in Lego toys. It is more meaningful for its investment in diversity and personalization, allowing their users to create different project types such as games, simulations and animations. Finally, it is more social, because of its social network that allows sharing projects, creating new projects from previous ones through mixing, and cooperating with pairs to develop collaborative projects. Scratch uses a kindergarten learning cycle that fosters creative thinking [16]. The cycle uses the steps of imagining, creating, playing, sharing and reflecting to aid in the learning process. Immediate feedback from the tool and the use of the Scratch social network allows for the cycle to be performed easily.

C. Learning introductory programming in CS1 courses

Robins and colleagues argue that most CS1 courses in a conventional curriculum are based on a lecture and lab approach, and are largely focused on knowledge, especially knowledge of language features [5]. They recommend instruction focused not only on language features, but also on program design and the combination of those features. They also suggest addressing programming mental models, such as models of control, data representation, program design and
problem domain. Finally, they reinforce the pedagogical role of lab activities as case-based problem solving sessions.

A systematic review of approaches to teaching introductory programming, based on 60 intervention reports, shows that alternative teaching interventions improve passing rates in CS1 courses by around one-third on average, when compared to a traditional lecture and lab approach [17].

Pears et al. collect and classify the literature of three decades of research on teaching introductory programming, identifying influential, synthesis and emerging work [1]. They classify 45 selected papers into four categories: curricula, pedagogy, language choice and tools for teaching. On the pedagogy category, they divide the studies of programming courses into three emphases: problem solving, learning a particular language, and code production.

Regarding assessment of alternative approaches in CS1 education, it is useful first to resort to Gross and Powers’ work, which evaluates various assessments of novice programming environments [18]. They classify assessment techniques as anecdotal, analytical or empirical, and then present an evaluation rubric for the quality of such assessments, evaluating the earlier assessments with the rubric.

There is a huge amount of literature of experience reports and assessments of alternative approaches for CS1. We focus here on some papers that uses the Scratch environment for higher education. One experience with Scratch uses it during the first three weeks of a CS1 course [19]. In this course, Scratch replaces the typical course sequence devoted to algorithms and pseudo-code. Students expressed higher motivation in contrast with the regular courses. However, no changes were found in either dropout rates or obtained scores when compared to the baseline course. Another experience uses Scratch during a two-week period to provide scaffolding for novices to learn basic programming concepts, and a tool for advanced learners to remain engaged through challenging work [20]. Both experiences are similar to ours in using Scratch to introduce programming concepts during a short period. Our approach, nonetheless, is different since our workshops happen before classes start, are led by peers, and are based on challenges and free interaction with the environment, instead of regular classes.

III. METHODOLOGY

The research approach used in this work is the case study. A case study is a contemporary phenomenon delimited in space and time [21], [22]. In this approach, the researcher is interested in investigating the phenomenon in a real environment, and in all its complexity.

A. Research Design

Our case was a five-day, 20-hour workshop of introductory computer programming for Computer Engineering freshmen undergraduates, which happened in August, 2014. The workshop used a particular learning approach, based on peer support, game development, a challenge-response strategy, and a playful programming environment for novices (i.e., Scratch). The phenomenon studied was the learning process developed during this workshop.

Our data analysis approach is both exploratory and quantitative. In an exploratory case study, one tries to answer research questions related to the phenomenon with no predefined hypotheses. The quantitative analysis intends to measure variables inside the case that are related to the phenomenon. To do such, we surveyed participants and we also analyzed the artifacts (i.e., software) they produced. We also performed qualitative data analysis, but the data are still under analysis and will be reported in a later work.

Our main goal was to understand the limitations and affordability of the aforementioned learning approach, and the learning issues that develop during the workshop.

Our research questions were four:

1) How do participants assess the workshop structure?
2) How do participants assess the workshop learning approach?
3) How do participants assess the Scratch learning environment?
4) Which skills and knowledge do participants acquire in the workshop?

Finally, approval for this research was obtained from the Institutional Review Board at the State University of Feira de Santana (UEFS).

B. Participants

Eighteen students participated in the research: 15 male and 3 female. All of them were freshmen in the Computer Engineering Undergraduate Program at the State University of Feira de Santana (UEFS). All participants signed an informed consent form, allowing their participation in this research.

C. Workshop Planning

Our workshop was based on previous workshops we have been offering since February, 2013. The present workshop happened one week in advance of the academic term, when students are free from activities of regular courses. The workshop was conducted during five consecutive days, four hours each day, in a computer lab with one desktop computer per student. Students were invited during enrollment and though notices at the UEF Computer Engineering web site.

The main learning goal was that, after the workshop, students could understand and apply basic concepts of algorithms and programming to develop small applications in a graphic programming language, thus, reducing potential difficulties in the first regular course on programming (CS1).

The learning approach used in the workshop was based on a combination of:

- peer support: three sophomore students led the workshop, helping participants to clear their doubts;
- game development: participants developed games in order to learn programming skills;
- challenge-response strategy: tutors challenged participants with small challenges during game development, instead of providing detailed explanations;
learning environments for novices: the Scratch environment was used as the main tool to learn programming, avoiding issues with language syntax and compilation.

The challenges proposed led to the gradual development of an animation and three classic games: Pong, Bow and Arrow, and Space Invaders (see Table I). One last challenge was the development of a simple calculator to allow for a smooth transition from Scratch to the C language.

<table>
<thead>
<tr>
<th>Day</th>
<th>Challenge</th>
<th>Main Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Animation</td>
<td>Motion, Looks</td>
</tr>
<tr>
<td>2</td>
<td>Pong / Space Invaders</td>
<td>Motion, Controls, Sensing, Operators, Variables</td>
</tr>
<tr>
<td>3</td>
<td>Space Invaders / Bow &amp; Arrow</td>
<td>Motion, Looks, Controls, Sensing, Operators, Variables</td>
</tr>
<tr>
<td>5</td>
<td>Bow &amp; Arrow / Calculator</td>
<td>I/O, Variables, Operators, Conditionals, Loops</td>
</tr>
</tbody>
</table>

One undergraduate tutor conducted the workshop, and was aided by two other undergraduate assistants. Initially, the tutor presented a short overview of the Scratch environment. Then, she proposed building an animation by interactively exploring the environment. Later, the tutor challenged the participants by showing the Pong game, and then asked them to produce that game. Then, she started to propose small challenges to develop parts of the game. Students actively explored the environment to answer to the challenges. Whenever they needed, tutor and assistants cleared their doubts. The same process happened with the other games.

D. Data Collection and Analysis

We used both quantitative and qualitative data collection procedures in order to deepen and enrich the research. This strategy is known as methodological triangulation. If the results of different procedures lead to the same conclusion, the research provides a higher degree of confidence [23], [24].

We used surveys and analysis of source code as quantitative data, and interviews and observations as qualitative data. However, we only describe quantitative results in this paper, since qualitative data are still under analysis. Two surveys were conducted with participants: one immediately after concluding the workshop (response rate of 72.2%, with 23% female and 77% male), and a second survey one month after term begin, right after the students had turned in their first lab assignment (response rate of 55.5%, with 20% female and 80% male). Even though the second survey was conducted in loco, the response rate was low because research participation was optional. Average age in both surveys was 18.6, with a standard deviation of 1.25 and 1.34, respectively, for the first and second surveys. Regarding the source code analysis, we examined only those kept in the computer desktops after the workshop, totalling nine student projects. From these, only four students fully participated in the five days of workshop.

IV. RESULTS

All the respondents of the first survey asserted that used the Internet every day. Regarding previous experience with programming, only one had thorough knowledge of the subject, while the rest had either no or few knowledge of it. None of them knew or had heard of Scratch before the workshop.

A. Workshop Assessment

We used a five-point scale between two extremes to let participants assess the workshop. For instance, one assessed criterion was the boring/stimulant dimension. The more to the left (right) the answer, the more boring (stimulant) the workshop was. The other criteria are described in Figure 1. All criteria but one had average value above 4.5. The workload was considered a little tiresome by some students.

B. Tool Assessment

Participants also assessed the tool, i.e., the Scratch programming environment, as shown in Figure 2. The friendly user interface was one of aspects best assessed, both the interface itself and the ability to immediately see code change results. Another relevant aspect was that the tool motivated participants’ creativity. The playful way how coding blocks are fitted was also well valued. Few students had difficulties to use the tool (23% of partial agreement), while very few had issues with block fitting (8% of partial agreement).

C. Learning Approach Assessment

Participants assessed the learning approach used in the workshop as well. Figure 3 shows participants’ opinions on
the learning approach we used. It is worth reminding that we provided only brief explanations of either the tool or of programming concepts, and we fostered participants to solve challenges we posed. The use of challenges had 76% of agreement to some degree, while 23% disagreed. The importance of free time to explore the tool was unanimous among participants. The adequate quantity of tutors available during the workshop also had unanimous agreement (38% partially agreeing, and 62% agreeing). Everyone fully agreed with the use of games, although there was some variation according to the particular game. The animation with the Scratch cat character was the less well assessed item. When asked to grade, between 1 and 5, how much the workshop motivated their learning, 85% of the participants graded it with 5 points.

We analyzed students’ multitasking behavior, rather common in this generation [25]. Parallel access to the Internet for goals unrelated to the workshop issues was almost zero: 92% asserted that had not accessed the Net (only one student recognized using it a little). Listening to music, although not significant, was more common than Internet use. One student said he had listened a lot of music, while 62% asserted not listening to any music at all.

D. Assessment of the Knowledge Construction Process

The Scratch tool groups programming blocks by theme, one tab for each theme. The analysis of student answers about the degree of difficulty to learn each theme showed that: Variables, Operators and Sensing had the highest degree of difficulty, while Control, Motion and Looks had the lowest, as shown in Figure 4. Nonetheless, in a scale between 1 and 5 (1=none; 5=substantial), no theme had average degree of difficulty higher than 3. The themes of Pen and Sound were not assessed, since they were not used during the workshop.

Table II shows the number of coding blocks used by each student, organized by theme, for the Space Invaders game, the most complex game in the workshop. This game was developed in two days, but students 4, 5 and 7 were present in only one day. On average, each student used 450 blocks. Looks, Control and Motion were the most used blocks.

Table III shows the features implemented by each student in the Space Invaders game. Each row refers to one particular feature, while each column refers to one participant. For fully implemented features, we assign value 1. Partially implemented features receive value 0.5. And unimplemented features get zero. Averaging over the features, we noticed that seven students had a game completion average of at least 0.7. Student 7, who missed one workshop day, was below average. Students 4 and 5 had average greater or equal to 0.7, even though they also missed one day.
Computing the Pearson correlation for the average blocks per student and the average features per student, we get a value of 0.69, showing strong correlation between both variables. In the Bow & Arrow game, correlation grows to 0.83. Obviously, quantity does not imply quality. But quantity of blocks is important, in this case, to successfully implement a feature.

Table IV shows the sum of blocks used for all games but the animation. Results show 5,550 blocks for all the nine students whom we retrieved the source code. Sensing, Variables and Operators were the least used blocks, while Control, Looks and Motion were the most used. We also show the number of days students have attended the workshop. The ones who came every day used more than 700 blocks, while the others used less than 600.

<table>
<thead>
<tr>
<th>TABLE IV. BLOCKS PER STUDENT FOR ALL THE GAMES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Days</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Student 9</td>
</tr>
<tr>
<td>Student 8</td>
</tr>
<tr>
<td>Student 7</td>
</tr>
<tr>
<td>Student 6</td>
</tr>
<tr>
<td>Student 5</td>
</tr>
<tr>
<td>Student 4</td>
</tr>
<tr>
<td>Student 3</td>
</tr>
<tr>
<td>Student 2</td>
</tr>
<tr>
<td>Student 1</td>
</tr>
<tr>
<td>Average</td>
</tr>
<tr>
<td>Standard Deviation</td>
</tr>
</tbody>
</table>

Finally, to get an idea of the effects of the workshops in our CS1 course, which uses the C language, we surveyed participants right after they finished their first coding lab assignment, one month after term begin. From 10 respondents, 90% asserted that either have not used or have briefly used Scratch after the workshop. Regarding the C language, 70% fully agreed that enjoyed programming in that language, while 40% agreed in some degree that had difficulties to code in that language (with 75% of these with partial agreement).

When asked whether the use of Scratch before the term helped them to learn how to solve the first assignment, 70% asserted it did so, either reasonably or a lot. The more specific we were, the more agreement there was: 80% agreed to some degree about the learning of control structures, while 90% said so for logic operators.

V. DISCUSSION

Here we discuss our findings, in the light of our goals and the literature. Discussion is ordered by our research questions (RQ), established in III-A.

A. RQ1: How do participants assess the workshop structure?

In general, the workshop was well evaluated by the students, who considered it: (1) stimulating: in general, they kept quite entertained throughout the days, working on the challenges proposed. Being stimulating is critical in that it is their first contact with the knowledge area that they chose to pursue a career in. Moreover, the current generation has a low threshold for boredom. In this sense, another evidence was the lack of multitasking behavior, reported by students, and observed by researchers. This type of behavior, which manifests as holding two or more usually unrelated complex tasks either simultaneously or alternately [26]. Recurring work in the literature point out to disruptions in learning when such behavior arises [27]–[29]; (2) useful: students learned skills that would be useful in the short term, in the CS1 course they began to attend a few days after the workshop; (3) organized: workshop schedule followed as planned, as described in Section III, along with well-defined goals, helped the fluency of activities with no setbacks, as perceived by participants.

Furthermore, the average workshop reviews of being conducive to learning and with good teaching by the peers were significant. Only the tiresome-light binomial scored a little low when compared to the others, maybe because the workshop took all morning shifts for five consecutive days. However, one should also consider that producing almost two games per day may have triggered some exhaustion, i.e., the workshop workload may have been excessive. Various students failed to implement all the proposed features of all the games. Therefore, either removing one of the games or rethinking the schedule in a next workshop offering should be considered.

Finally, to get an idea of the effects of the workshops in our CS1 course, which uses the C language, we surveyed participants right after they finished their first coding lab assignment, one month after term begin. From 10 respondents, 90% asserted that either have not used or have briefly used Scratch after the workshop. Regarding the C language, 70% fully agreed that enjoyed programming in that language, while 40% agreed in some degree that had difficulties to code in that language (with 75% of these with partial agreement).

When asked whether the use of Scratch before the term helped them to learn how to solve the first assignment, 70% asserted it did so, either reasonably or a lot. The more specific we were, the more agreement there was: 80% agreed to some degree about the learning of control structures, while 90% said so for logic operators.

B. RQ2: How do participants assess the workshop learning approach?

As previously stated, the workshop learning approach was based on a combination of peer support, a challenge-response strategy, game development, and a learning environment for novices. All participants agreed, to some degree, that the approach was adequate. With 18 participants for three tutors, we had a ratio of six students per tutor. Peer support is a widely used strategy in introductory programming learning, as reported in the literature: both formal and informal, in classroom or in the lab, or even outside the classroom [30]. Our approach is distinct from other experiences because the workshop is made by students and for students. Two positive consequences follow: (1) integration: since the workshop happens a week before regular classes begin, it ends up being an event where freshmen get to know each other, and integrate with senior students; (2) sense of belonging: being next to fellow senior students allows freshmen to feel they belong in that community.

The use of challenges was well received by students. The goal was that they explored both solutions and the environment in a self-directed way. Tutors gave just short explanations, and intervened only when needed or requested. In any case, that strategy led more to the use of leading questions instead of giving straight answers.

Challenges are one of the foundations of active learning, a widely discussed topic in pedagogy [31]. An example of active learning is problem-based learning (PBL), which has been widely used in computing courses [32], [33]. In PBL, challenges take the form of a problem that resembles professional practice, arousing students’ interest on content knowledge, and stimulating self-directed learning, among other benefits [34]. A difference from our approach to PBL is that although we encourage discussion among students, it happens only in a few moments, while in PBL, discussion takes most of the group
meetings devoted to solving problems. By the way, similarities between challenges and PBL are welcome in our context, since our Computer Engineering undergraduate program is strongly based in PBL.

One benefit of using challenges is letting students free to advance in their own pace. Sometimes they did more than requested, making them more proactive. We had, for instance, some students who took their project home and improved them with more features. Such attitude was encouraged by the tutors.

Using games as challenges has proved a right decision. It may seem obvious, but the fact that participants rated games like Pong or Bow and Arrow more motivating than the initial animation suggests that they prefer more complex tasks, provided that they are motivating. Challenges that assume a playful facet, such as games, are relevant because such activities are part of human beings’ history, and involve various elements such as rules, goals, tension and competition [35] — also present in the workplace. Indeed, we may say that game development is a challenging task, since it implies a substantial complexity in tasks of communication between objects, logical and mathematical operators, scenery, among others. In addition, the use of games adds at least two important features to the teaching-learning process: (1) everyday life: games are part of youth daily life. Thus, working with something that fascinates them, that is part of their reality, is likely to increase motivation; (2) amusement: learning by playing minimizes the view of studying as obligation, reinforcing entertainment and, hence, enhancing learning [36].

C. RQ3: How do participants assess the Scratch learning environment?

The choice of Scratch as the learning environment for beginners also facilitated the decision of merging games with challenges. First, participants strongly agreed that Scratch was favorable for game development and learning: a playful environment for creating other playful environments. Second, immediate feedback on the screen was also much appreciated. Third, avoiding issues of compilation and syntax was also welcome. Those features led to a good assessment by participants. They also had no bigger issues with using the tool, besides considering it user-friendly and leading to creativity development.

We must also highlight that, for the aforementioned factors, it is possible to build complex programs in Scratch in a relatively short time. An undergraduate student, new to programming, would likely take a long time to develop games such as Space Invaders using traditional programming languages, even at the end of the first term. But they did it after only a few days. It is worth noting that the workshop led to an average of 450 blocks used per student, for developing three games. Participants who came to every workshop day used more than 700 blocks. If we equate blocks to lines of code, we realize how much code our beginning programmers had developed in such a short period, especially when compared to the code students develop in their first CS1 classes.

Nevertheless, after the workshop, students made it clear that they did not keep using Scratch. Natural reasons for this were: they were then focused on learning a programming language, had a high workload for the courses they were registered in, and needed to adapt to a new student routine. Another reason was that their perception of Scratch changed: Scratch was now perceived as an amateur language, while C felt as something more professional and, therefore, more relevant.

D. RQ4: Which skills and knowledge do participants acquire in the workshop?

The most used blocks were from the themes of Looks (user interface), Control (selection and repetition structures) and Motion (for moving objects in the game). These are precisely the themes that showed the least degree of difficulty for participants. Control blocks, which are usually hard for beginning programmers, were the most used blocks, while blocks of Operators and Variables were the least used. It is worth mentioning that the less implemented features in the Space Invaders game were counters (e.g., for life and score), even though the final results depended on these counters. This may likely have happened because counters require variables, one of the difficult themes according the students.

The theme Operators, which handles logical and mathematical operators, had a greater degree of difficulty than the theme Control, responsible for selection and repetition structures. Interestingly, those structures are not usually easy to learn — hence the effort to offer this workshop — and require, a priori, a more elaborate logic than such operators. Working in a playful manner with such structures may have facilitated learning.

In summary, we finished the workshop with 5,550 blocks used (given that only nine participants provided their projects for analysis), which allows us to infer that students have really evolved in terms of introductory programming. Sensors, Variables and Operators were the topics less practiced, while Control, Looks and Motion, the most commonly used.

Finally, despite students have not kept on using Scratch, we may say that the experience contributed substantially to their learning of programming during the first month of the term. 70% asserted that the workshop helped a lot in the process of knowledge construction of the first CS1 lab project. 80% agreed, to some degree, that it helped on selection and repetition structures, and 90%, on logical operators. This result is significant, and it was precisely one of the workshop goals, suggesting that the learning approach together with the adopted tool provided a more fluid and meaningful learning of programming.

VI. Conclusions and Future Work

This paper described a case study of a teaching and learning approach aimed at facilitating learning of programming concepts by Computer Engineering undergraduate freshmen. Our approach combined the use of the Scratch learning environment, peer support, a strategy of challenge-response, and game development. The case developed as a workshop for freshmen that lasted one week, with a workload of 20 hours. Three sophomore peer students guided the workshop, presented to 18 freshmen. The challenges led to the development of animations, games and a calculator. To evaluate our approach, we used a quantitative approach based on artifact analysis and
two surveys, one post-workshop, and another after students finished their first CS1 lab project.

Students assessed the approach well in some dimensions: stimulant, lightweight, good teaching, useful, well-organized, and conducive to learning. They also evaluated the Scratch environment as user-friendly, and with good working mechanics, especially the Lego-style blocks. Commands of interface (Looks), control structures (Control), and movement (Motion) were the most used, considered less difficult by the students. Although most students have not used Scratch after the workshop, 70% of them agreed that the environment helped them to learn programming. Regarding control structures, 80% agreed that the workshop aided them in developing their first CS1 project to some degree. This is relevant, for these are concepts usually hard to learn in regular courses. On the other hand, variables were not grasped so well, even though they had been used in the games. Results suggest the effectiveness of the approach. Moreover, the work also contributes to the field of computing education with empirical evidence, helping to better evaluate learning approaches and tools.

Further work should deal with qualitative analysis of these workshops, based on interviews and observation. We also intend to add a longitudinal dimension to the work, analyzing workshops from different terms and their evolution.

ACKNOWLEDGMENT

We would like to thank the freshmen students who voluntarily took part in this research. We also thank FAPESP – Bahia State Foundation for Research Aid, and UEFS – State University of Feira de Santana, for their funding of undergraduate research fellowships.

REFERENCES


