Martínez-Valdés, J. A., Velázquez-Iturbide, J. Á., & Hijón-Neira, R. (2017). A (Relatively) Unsatisfactory Experience of Use of Scratch in CS1. In J. M. Dodero, M. S. Ibarra Sáiz, & I. Ruiz Rube (Eds.), Fifth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'17) (Cádiz, Spain, October 18-20, 2017) (Article 8). New York, NY, USA: ACM. doi:10.1145/3144826.3145356

A (Relatively) Unsatisfactory Experience of Use of Scratch in CS1

José Alfredo Martínez-Valdés
ETS Ingeniería Informática
Universidad Rey Juan Carlos
28933 Móstoles, Madrid, España
ja.martinezv@alumnos.urjc.es

J. Ángel Velázquez-Iturbide
ETS Ingeniería Informática
Universidad Rey Juan Carlos
28933 Móstoles, Madrid, España
angel.velazquez@urjc.es

Raquel Hijón-Neira
ETS Ingeniería Informática
Universidad Rey Juan Carlos
28933 Móstoles, Madrid, España
raquel.hijon@urjc.es

ABSTRACT

Scratch is a "rich-media programming language" that has become very popular at high school because students may learn it very quickly and produce surprisingly animated programs. Consequently, some instructors have proposed using Scratch at the university in introductory programming courses. Their experiences report on students' high motivation and sometimes also on higher performance. We adopted Scratch as the introductory programming language for a CS1 course in a videogames major. It was used for two weeks and then the course switched to using Java. The results we obtained for both the Scratch language and the Dr. Scratch tool were less satisfactory than expected and, in some regards, disappointing. We describe our experience, analyze students' acceptance and discuss some consequences and lessons learnt to Scratch in university courses.

CCS CONCEPTS

• Social and professional topics~CS1 • Software and its engineering~Visual languages

KEYWORDS

CS1, Scratch, Java, students' acceptance

1 INTRODUCTION

Scratch is a "rich-media programming language" that has become very popular at high school, especially as a means to develop computation thinking [2][5]. Scratch has several features that smooths students' learning curve. Firstly, they do not have to learn the details of a programming language syntax but they just have to remember the names of statements (sometimes, it is even sufficient with knowing the intended effect) and selecting them. Secondly, the nature of media projects demands less abstraction effort from students than traditional numeric or information processing problems. Thirdly, the immediate visual and sonorous feedback easies the task of creating and debugging programs, and increases students' motivation. Fourthly, Scratch is more flexible and powerful than similar past efforts, such as Logo [10]. Elaborating on a metaphor by Seymour Papert, it is often claimed that Scratch has "low floor, high ceiling and wide walls" [11], that is, easy to be started, opportunities to create increasingly complex projects over time, and supporting many different types of projects. As a consequence, novice students may produce attractive and elaborate programs after only a few days of programming experience. Finally, Scratch provides the chance of collaborating and sharing by means of a web community.

These features make Scratch an attractive programming language also at the university level [14]. The literature on programming education has identified a number of difficulties and approaches to address them [13]. Block-based languages [4] such as Scratch, offer an opportunity to reduce some of these difficulties. Actually, some experiences exist on using Scratch in introductory courses. Malan and Leitner [6] report on the use of Scratch at a summer CS0 course. They devoted just two lectures and two assignments to Scratch. Students were engaged elaborating their projects and spent more hours than expected. Furthermore, most students with no prior programming experience considered that Scratch had assisted them in learning programming. Rizvi *et al.* [12] describe a semester-long CS0 course offered to improve the retention, performance and attitudes of at-risk majors. An evaluation indicated that the course was effective in preparing students with respect to perceived self-efficacy and performance.

Mishra and colleagues [7] report on a similar experience in a CS1 course. Lectures in the first two weeks were focused on Scratch and they later transited to C++. Consistently with the above experience, the authors conclude that students were

engaged and satisfied. The experience was satisfactory for both novice and expert programmers: the former were able to learn with little effort and the latter were motivated to address complex challenges.

Our motivation for using Scratch in a CS1 course for a major on videogames was similar to these experiences. We knew that students had different backgrounds, and we felt that using Scratch as an introduction could be motivating. We especially had this expectation for novice students without programming experience (who often fit an "artistic" profile). Scratch was used for two weeks and then the course switched to using Java. However, the results we obtained for both the Scratch language and the Dr. Scratch tool were not as satisfactory as we expected. We describe our experience, analyze students' acceptance and discuss some lessons learnt to use Scratch in university courses.

The structure of the paper is as follows. In the second section, we describe the research setting, including the course, our research questions and measurement instruments. The third section presents the results of analyzing data gathered from different sources. Finally, sections 4 and 5 contain a discussion of the results and our conclusions, respectively.

2 RESEARCH SETTING

In this section, we describe the course features, the research questions and the instruments used.

2.1 Course Outline

The CS1 course belongs to the degree on design and development of videogames. The course includes two weekly hours for lectures and two weekly hours for lab activities.

The videogames degree has distinctive features from conventional computing degrees. A number of courses deal with subjects which traditionally belonged to the expertise of arts or design sciences. Consequently, a number of students do not fit the stereotype of a technical student but a more "artistic" one.

We proposed addressing such a demographic diversity in the CS1 course by first introducing programming with a visual language and then transiting to a conventional programing language, such as Java. Scratch was instructed for two weeks, including an assignment that students had to deliver during the second week. Students were free to choose the topic of their project, but they had to conform to the use of several language elements, including loops, events, operators and variables.

The assignment was graded according to students' use of the elements of Scratch language and the project functionality and appearance. In addition, assignments were graded with the Dr. Scratch tool [8][9]. Dr. Scratch assesses Scratch projects with respect to 7 "dimensions", namely logical thinking, data representation, user interactivity, flow control, abstraction and problem decomposition, parallelism, and synchronization 0. We do not give a detailed definition of the dimensions, as their names are self-descriptive for a first oncoming (however, we get into a bit more detail in the discussion section). A project can be graded for each dimension in one of three levels, depending on the level of sophistication achieved by the project code. In addition, Dr. Scratch may give four figures on some bad habits detected in the project [9], namely attributes incorrectly initialized, inadequate sprite names, duplicated scripts, and dead code.

2.2 Course Demography

A total of 93 students were enrolled in the CS1 course of the videogames degree. Although instructors knew that the students enrolled had a different profile than students reenrolled in computing degrees, no rigorous study had been conducted. We considered that demographic information was relevant per se and it could also be relevant for the analysis of other data gathered. In addition, we expected that students with different features had different expectations and capabilities, and therefore they could have different performance and opinions. Finally, Scratch allows creating with little effort attractive animations, which seemed to be a good way to motivate students interested in videogames.

We asked students to fill in their first day of class a questionnaire where they had to give personal information. Some questions asked objective data whereas others demanded a subjective classification.

The questionnaire was composed of seven questions on:

- Personal data (name, gender and identity number).
- Students' profile. We asked students to self-classify in one of four categories, namely, technical, artistic, both, or undefined.
- Whether they had previous programming knowledge and, in the affirmative case, what language elements they knew (to select from a closed list) and what programming languages they knew.

The results of this questionnaire are given in the third section.

2.3 Research Questions and Measurement Instruments

We addressed two research questions:

RQ1. What is students' performance with Dr. Scratch and students' opinion about Dr. Scratch scores?

RQ2. What is students' acceptance of using Scratch as an introductory language to Java (and to programming in general)?

We describe the data and analysis conducted to answer each of the research questions. For RQ1, students were asked to fill in an online questionnaire. They were asked to copy all the results given by Dr. Scratch to their projects. In addition, students were asked to rate how well Dr. Scratch assessment had assisted them in enhancing their programming skills. The question was: "do you think that (Dr. Scratch) has assisted you in being aware of the issues where you must persevere and how?".

For RQ2, a questionnaire was elaborated and delivered at the end of the course. The questionnaire consisted of 4 open questions asking students to identify those parts of the course they would either incorporate or suppress, or to which they would devote more or less time.

As we explain in the following section, a number of students advocated for suppressing Scratch. As a consequence, we delivered two specific questions in June to students who did not pass the course in January. In the questionnaire, after explaining the rationale for the questions, we asked students the following questions: "Do you agree with suppressing Scratch in the course?, Why?".

Multiple choice questions have been analyzed using descriptive or inferential statistics, depending on the case. Open questions were analyzed using qualitative methods. In particular, they were analyzed according to the principles of grounded theory [3]. Given that the focus of open questions was very specific, the diversity of answers was not too high and most categories could be foreseen.

3 RESULTS

We present in this section the results of our research. We first present the demographic characteristics of the enrolment. We then present the findings related to the two research questions.

3.1 Course Demography

The results of the initial questionnaire are shown in Table 1.

Table 1. Different features of enrolled students

Factor	Value	# students	%
Gender	Man	78	90%
	Woman	9	10%
Profile	Technical	39	45%
	Artistic	42	48%
	Both	5	6%
	Undefined	1	1%
Repeat student	Yes	15	17%
	No	72	83%
Previous programming knowledge	Yes	42	48%
	No	45	52%

Note that most students are male and are non-repeat students. However, there exists a balance between technical and artistic students, and also between students with and without previous programming knowledge.

Obviously, there were differences in programming knowledge between repeat and non-repeat students. Only one repeat student did not know any language vs. 63% of non-repeat students. The most widely known language was Java for any kind of students, tied with C in the case of non-repeat students. Surprisingly, only one student knew Scratch, and he was a non-repeat student.

3.2 Performance with and Acceptance of Dr. Scratch

3.2.1 Performance According to Dr. Scratch

We use Dr. Scratch scores as the measure of students' performance on Scratch mastery. Dr. Scratch may score each dimension with a value ranging between 0 and 3. As 7 dimensions are established, the total score may range between 0 and 21.

The projects gathered were scored with values ranging between 10 and 20 (see Table 2).

If we break down total scores according to the seven dimensions, we obtain the more elaborate data shown in Table 3.

Table 2. Scores given by Dr. Scratch (N=89)

Score	# students	%
20	2	2%
19	1	1%
17	8	9%
16	8	9%
15	10	11%
14	18	20%
13	15	17%
12	14	16%
11	7	8%
10	6	7%

Table 3. Scores given by Dr. Scratch broken down into dimensions (N=89)

Statistical measure	Dimensions of computational thinking*							Total
	PA	LT	FC	IN	IR	AB	SN	
Mean	2.18	1.90	2.28	2.01	2.01	1.17	2.21	13.76
Std. dev.	0.90	0.98	0.45	0.11	0.18	0.55	0.73	2.23
Median	3	2	2	2	2	1	2	14
Mode	3	1	2	2	2	1	2	14
Minimum	1	0	2	2	1	1	1	10
Maximum	3	3	3	3	3	3	3	20

*PA: parallelism, LT: logical thinking, FC: flow control, IN: interactivity, IR: information representation, AB: abstraction, SN: synchronization

Students also submitted the bad habits reported by Dr. Scratch. Table 4 gives the figures of these messages.

Table 4. Bad habits reported by Dr. Scratch (N=89)

Bad habits	# students	%
Attributes incorrectly initialized	87	98%
Inadequate sprite names	47	53%
Duplicated scripts	25	28%
Dead code	7	8%

We also inquired whether there was any correlation between scores and the different factors in the population (gender, profiles, etc.) We did not find any correlation.

3.2.2 Acceptance of Dr. Scratch

We also analyzed students' opinion on how useful Dr. Scratch was to improve their coding skills. Table 5 shows the kinds of answers given by students to the open question on whether Dr. Scratch helped them to better code and how. The "yes, but…" category clusters answers which are positive but also report on some negative aspects of Dr. Scratch. An example of this kind of opinion follows: "Yes, although some errors (…) are, I think, irrelevant" (Student 71, represented S71).

Table 5. Kind of answers on Dr. Scratch usefulness (N=89)

Kind of answer	# students	%
In blank	39	44%
Yes	27	30%
Yes, but…	9	10%
No	14	16%

Students gave different explanations for giving these answers. Table 6 shows the different ways in which Dr. Scratch helped some students in their learning.

Table 6. How Dr. Scratch assisted students

Reason	# students
To program better	11
To improve the use of some element of Scratch	8
To better know Scratch	3

Table 7 shows the explanations given on negative opinions.

Table 7. Negative opinions on Dr. Scratch

Reason	# students
The student does not agree with Dr. Scratch criteria	8
Dr. Scratch does not assess the global quality of the program	5
Dr. Scratch functionality	4
Dr. Scratch does not guide on how to program better	1

Let us elaborate the first three categories:

- The student does not agree with Dr. Scratch criteria. One student does not agree with having to include additional language features in order to increase Dr. Scratch score in those cases when the intended project functionality is satisfied: "(…) and it demands code, which may be unnecessary in your project, to give you a higher score" (S72). Other students disagree with being forced to change the default names assigned by Scratch to sprites or variables. Finally, some students disagree with other specific assessment criteria.
- Dr. Scratch does not assess the global quality of the program. Most students assigned to this category consider that Dr. Scratch should also assess the whole project, not only partial aspects. For instance: "Not exactly. I played my game and I offered it to my mates to test it and they found it fun, as a game should be. I used all the functions practiced in the classroom, demonstrating the knowledge acquired." (S89).
- Dr. Scratch functionality. Several students complained about the understandability of Dr. Scratch messages. Another student reported that Dr. Scratch does not evaluate well according to its own criteria.

3.3 Acceptance of Scratch as an Introductory Programming Language

We present the results of the two questionnaires delivered to gather students' opinion about the use of Scratch as a first language, before being instructed on Java.

3.3.1 First Final Questionnaire

We gathered 76 answers to the questionnaire. The four open questions in the questionnaire delivered at the end of the course contained a number of heterogeneous opinions and suggestions on the course. However, we do not conform to the questions for the answers analysis due to several idiosyncrasies. Thus, some answers should be given in a different question. In addition, some students give the same opinion several times, to different questions. Furthermore, some answers to a question are compound, i.e. they contain several simpler opinions.

In summary, we did two operations on the answers for their analysis: we grouped answers into two broad categories

(positive/negative) and we decomposed the answers of each student into simple comments (and removing duplicates).

Seven comments about the use of Scratch in the course were (relatively) positive. They can be grouped into two categories:

- Scratch is potentially useful as an introduction to Java, but their relation should be explicitly addressed (5 answers).
- Scratch is useful as an introduction to programming (2 answers).

However, the number of negative opinions was larger by far. Forty one students (55% of the number of respondents) thought that Scratch should be suppressed or it should deserve shorter time in the course schedule. These negative answers can be grouped into two categories:

- Scratch should be removed (17 students, 23%).
- Time devoted to Scratch should be shortened (24 students, 32%). Some answers suggest the number of sessions that should be devoted to Scratch, ranging from one session to two weeks. Note that the latter suggestion coincides with the time actually devoted in the current academic year; thus, we must take this opinions as a phrasing of a feeling rather than as a meditated suggestion. Some more specific suggestions follow:
    - Change part of the contents of Scratch lectures, by changing the nature of examples or explaining better its relation to Java (2 students).
    - Suppress or change the assignment status from mandatory to voluntary (2 students).

3.3.2 Second Final Questionnaire

In the second final exam (for repeat students in the current academic year), we wanted to obtain a confirmation of students' negative opinion on the use of Scratch as an introduction to programming and Java. We explicitly asked whether they agreed with suppressing Scratch in the course.

We obtained 26 answers, as Table 8 shows.

Table 8. Opinion on suppressing Scratch (N=26)

Answer	# students	%	# students with an explanation
Yes	15	58 %	9
No	11	42 %	11

We analyzed in detail the reasons given by the students for their opinion. We divided the explanations into simple explanations, as some students gave several reasons. Thus, the 9 students who were in favor of suppressing Scratch contributed with 14 simple reasons. They can be classified into three categories, as Table 9 shows:

Table 9. Opinions on suppressing Scratch (N=14)

Reason	# answers
It does not assist in learning Java	7
Time devoted to Scratch could be devoted to a different topic	5
It is studied in high school	2

Analogously, we analyzed the reasons given by students who disagree with suppressing Scratch. We obtained 19 simple arguments. They can be grouped into three categories. Note, however, that the most frequent category corresponds to a negative opinion. They were given by students who disagreed with completely removing Scratch, but they did consider that the time devoted should be shortened. Actually, all the 8 students who gave the second reason also gave the first one.

Table 10. Opinions on non-suppressing Scratch (N=19)

Reason	# answers
Time devoted to Scratch could be devoted to a different topic	9
It assists in learning to program	8
It is fun	2

4 DISCUSSION

We discuss the results presented in the previous section. Results about demography are useful because they confirm our suspicions about the dual profile of technical and artistic students. In addition, and to our surprise, show that Scratch is not as widely known by high school students as we expected. However, demography is not a central issue of our study. We expected to find any correlation between profiles according to any dimension and some results, but we did not succeed.

Therefore, we only discuss results about Scratch and about Dr. Scratch.

4.1 Students' Performance According to Dr. Scratch

Scores assigned by Dr. Scratch are in the middle strip of potential scores, with a mean equal to 13.76, and median and mode equal to 14. These results are consistent with other results reported by Moreno-León et al. [9]. In their study, high school students obtained the above three measures equal to 11.5, 11 and 10, respectively. As their population are between

10 and 14 years old and our students are freshmen, it was expected higher performance.

The results about the seven dimensions distinguished by Dr. Scratch are more puzzling. Students obtained higher values for some dimensions than for others. In particular the score obtained in the abstraction and problem decomposition dimension (AB, 1.17) are much lower than for the rest of dimensions, which scored a median equal to 1.9 or higher. Logical thinking (LT) also ranks lower than other dimensions, but the results are not as sharp (mean=1.9, mode=1, minimum=0).

At the other extreme, we place parallelism (PA, mean=2.18, median=3), flow control (FC, mean=2.28, minimum=2), interactivity (IN, minimum=2), and synchronization (SN, mean=2.21).

Several explanations can be hypothesized for these different results. Some features of Scratch can simply be more intuitive than others. For instance, parallelism is natural in Scratch as opposed to its support in other languages, such as Java. Alternatively, it could be that some features could be more adequate than others to implement typical students' projects (games, simulations, etc.) For instance, clones (AB, level 3) are an advanced feature and logic operations (LT, level 3) are not used too frequently. On the contrary, the repeat-until block (FC, level 3) or the wait-until block (SN, level 3) are frequently used to implement projects of not-trivial complexity. A last explanation for different results could be that the design of dimensions and levels should be further elaborated. Further work is necessary to further explore these issues.

4.2 Students' Acceptance of Dr. Scratch Scores

The percentage of students who considered that Dr. Scratch had been useful (40%, counting the "yes" and "yes, but…" categories) is higher than those who had any negative opinion (26%, counting the "yes, but…" and "no" categories). However, the percentage of negative opinions is higher than we would like for a learning tool.

Students with a positive opinion claimed that Dr. Scratch assisted them in coding better. Those who are more specific in their opinion claim that, thanks to Dr. Scratch, they were more aware of some language features or they even got to know about their existence.

However, concerns about the use of Dr. Scratch are of several types. Some students complain about the cryptic nature of messages. This criticism is similar to the problem found in novice programmers to understand compiler messages [1].

However, other criticisms are more related to the (current) design of Dr. Scratch itself. The tool assesses the user's usage of the different elements of the language. In some contexts, an assignment may be stated in terms of the language elements used [6] (actually, we stated it in this way). However, it is more common to write the assignment statement in terms of the intended functionality of the program. Thus, students become disappointed when they think that their program is well developed (sometimes in consensus with their classmates) and Dr. Scratch only gives an intermediate score.

An alternative approach would consist in allowing the instructor to identify the subset of the dimensions to assess and the level of mastery expected in each dimension. As a consequence, Dr. Scratch would assess projects aligned with the learning goals of the assignment.

4.3 Students' Perception of Scratch as an Introductory Programming Language

The most unexpected result was students' consensus on considering that shorter schedule time should be devoted to Scratch, and it should even be suppressed. Obviously, our expectations were not fulfilled, as students were not motivated. There are several reasons for advocating students for a reduction of time. Firstly, some students associate Scratch with pre-university education, thus they consider its use childish and "non-serious". One student even used this term in his/her answer: "It is not useful to learn the more «serious» programming" (S03). Other students claim that, given that the intended goal of learning Scratch is easing the path to Java, their relation should be addressed explicitly. Also, given the narrow slot of time devoted to Scratch, its assignment completion should be voluntary.

Some negative opinions can also be attributed to the instructor. We have identified that some students demand explicit explanations of the relation between Scratch and Java. In addition, some students demand more interesting, less routine examples. Finally, students suffered the lack of time to address the last topic of the course. Therefore they consider that if Scratch had not been included in the syllabus, more time would have been available for the last topic.

5 CONCLUSIONS

Similarly to other experiences, we introduced Scratch in the first weeks of a CS1 course (in our case, offered to videogames students). However, the results were less satisfactory than in other proposals found in the literature with respect to students' motivation. The dissatisfaction not only affects the Scratch language but also (partially) to the assessment tool Dr. Scratch. We have identified a number of reasons for this dissatisfaction. With respect to Dr. Scratch, two recommendations are to review the scoring criteria, and to include support to a more flexible assessment scheme (where the expected score is not always 3 in all the dimensions). In addition, instructors should not use Dr. Scratch scores as the only factor for grading. At least, they should assess the project functionality.

With respect to the Scratch language, the main conclusion is that an informal introduction of Scratch, as reported in other experiences, may be unsatisfactory for some university students. The instructor should argue that Scratch is neither a toy language nor a language for children. The relation of

Scratch features to programming and to the next programming language should be explicitly addressed. Finally, examples and assignments should also be revisited.

ACKNOWLEDGMENTS

This work was supported by research grants iProgr of the Ministry of Economy and Competitiveness (ref. TIN2015-66731-C2-1-R), eMadrid of the Regional Government of Madrid (ref. S2013/ICE-2715), and ref. 30VCPIGI15 of the Universidad Rey Juan Carlos.

REFERENCES

[1] Becker, B. A., Glanville, G., Iwashima, R., McDonnell, C., Goslin, K., and Mooney, C. 2016. Effective compiler error message enhancement for novice programming students. *Computer Science Education* 26, 2-3 (2016) 148-175, DOI= 10.1080/08993408.2016.1225464.

[2] García-Peñalvo, F. J., Reimann, D., Tuul, M., Rees, A., and Jormanainen, I. 2016. *An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers.* Belgium: TACCLE3 Consortium. DOI= 10.5281/zenodo.165123.

[3] Glaser, B., and Strauss, A. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago.

[4] Kelleher, C., and Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* 37, 2 (2005) 83-137. DOI= 10.1145/1089733.1089734.

[5] Lye, S. Y., and Koh, J. H. L. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41 (2014) 51-61. DOI= 10.1016/j.chb.2014.09.012.

[6] Malan, D. J., and Leitner, H. H. 2007. Scratch for budding computer scientists. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education.* SIGCSE'07. ACM, New York, NY, 223-227. DOI= 10.1145/1227310.1227388.

[7] Mishra, S., Balan, S., Iyer, S., and Murthy, S. 2014. Effect of a 2-week Scratch intervention in CS1 on learners with varying prior knowledge. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education.* ITiCSE'14. ACM, New York, NY, 45-50. DOI= 10.1145/2591708.2591733.

[8] Moreno-León, J., and Robles, G. 2014. Automatic detection of bad programming habits in Scratch: A preliminary study. In *Proceedings of the 2014 Frontiers in Education Conference*. FIE 2014, IEEE, 1-4. DOI= 10.1109/FIE.2014.7044055.

[9] Moreno-León, J., Román-González, M., and Robles, G. 2015. Dr. Scratch: Automatic analysis of Scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia* 46, 10 (2015).

[10] Papert, S. 1980. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, New York.

[11] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. 2009. Scratch: Programming for all. *Commun. ACM* 52, 11 (2009) 60-67, DOI= 10.1145/1592761.1592779.

[12] Rizvi, M., Humphries, T., Major, D., Jones, M., and Lauzun, H. 2011. A CS0 course using Scratch. *Journal of Computing Sciences in Colleges* 26, 3 (January 2011) 19-27.

[13] Robin, A., Roundtree, J., and Roundtree, N. 2003. Learning and teaching programming: A review and discussion. *Computer Science Education* 13, 2 (2003) 137-172. DOI= 10.1076/csed.13.2.137.14200.

[14] Wolz, U., Leitner, H. H., Malan, D. J., and Maloney, J. 2009. Starting with Scratch in CS 1. In *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE'09. ACM, New York, NY, 2-3. DOI= 10.1145/1508865.1508869.