Automatic Detection of Bad Programming Habits in Scratch: A Preliminary Study

Jesús Moreno
Programamos.es & Instituto Nacional de
Tecnologías Educativas y de Formación del Profesorado
Madrid, Spain
jesus.moreno@programamos.es

Gregorio Robles
GSyC/LibreSoft
Universidad Rey Juan Carlos
Madrid, Spain
grex@gsyc.urjc.es

***Abstract*—Using the Scratch environment as a tool to teach programming skills or develop computational thinking is increasingly common in all levels of education, well-documented case studies from primary school to university can be found.**

However, there are reports that indicate that students learning to program in this environment show certain habits that are contrary to the basic programming recommendations. In our work as instructors with high school students, we have detected these and other bad practices, such as the repetition of code and object naming, on a regular basis. This paper focuses on verifying whether these issues can generally be found in the Scratch community, by analyzing a large number of projects available on the Scratch community website.

To test this hypothesis, we downloaded 100 projects and analyzed them with two plug-ins we developed to automatically detect these bad practices. The plug-ins extend the functionality of the Hairball tool, a static code analyzer for Scratch projects.

The results obtained show that, in general, projects in the repository also incur in the investigated malpractices. Along with suggestions for future work, some ideas that might assist to address such situations are proposed in the conclusions of the paper.

Keywords— Scratch; programming; bad habits; learning; automatic assessment; Scratch repository; empirical analysis

I. Introduction

Scratch [10] is a programming environment including a visual programming language designed for children over 6 years old, a development environment and a website where the community can host their projects, run, study and reuse other programs, and share ideas or suggestions with other programmers. Although there are several similar visual programming languages, Scratch is undoubtedly the most successful one with more than three million registered users in its website and more than five million shared projects in its repository[1].

Scratch is being used both in extracurricular activities [4] or summer camps [1], as in elementary schools [13], high schools [9] and even universities worldwide [5]. Its capacity to bring computer programming to children and teenagers with the aim of fostering computational thinking abilities and to promote ICT studies, and its success to teach both basic and advanced programming concepts (concepts such as user interaction, conditional statements, communication and synchronization, logical expressions and data representation are taught) has been well documented [7], [3], [15].

However, it is also possible to find studies that have detected some bad programming habits in students learning to program in this environment [8]. In this line, in our work as instructors we have also found some bad habits despite having insisted our students to avoid them. As these habits are contrary to the basic programming recommendations, our study tries to find out if they are also common in the projects shared in the Scratch repository.

The goal of this paper, therefore, is to determine whether these bad habits are common practices in the Scratch community in an automatic manner and propose some ideas to try to avoid such situations. In order to assist evaluators to detect these malpractices we have developed two plug-ins that automate the detection of the bad programming practices by extending the functionality of Hairball, a static code analyzer of Scratch projects.

The structure of this paper is as follows: In section II a brief overview of Scratch is presented, including some research studies that have investigated learning in this environment and other studies that automate the evaluation of learning. In section III our work methodology is briefly described. Section IV presents the results we have obtained from applying our methodology on a small sample of Scratch projects. Finally, Section V contains the conclusions of our study and some ideas and suggestions for future work are discussed.

II. Background

Scratch is a visual programming environment, developed by the Lifelong Kindergarten group at the MIT Media Laboratory, that allows to program interactive projects using a drag-and-drop approach with different pieces or *Lego* style blocks [10]. Scratch was designed with the main goal of encouraging young people to program, and most of its users are novice learners with no prior programming experience [6]. The environment is designed to encourage users to manipulate and modify various media types, so that children and teenagers can create projects that they find interesting and attractive, such as animated stories, games or interactive presentations [7].

Scratch also offers a website where the projects programmed by the community are published. The code of programs hosted there can be run, and following the community

[1] See http://scratch.mit.edu/statistics/

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

idea of the free software movement it can be studied and analyzed. So, it is possible to make suggestions and to reuse the code, and there are lots of valuable resources available for anyone willing to start to program. This approach offers as well the opportunity to learn social skills related to programming and beyond, such as sharing and contributing to the community [11].

Despite the positive impact of Scratch, Meerbaum-Salant, Armoni and Ben-Ari [8] prove that students learning to program in this environment show some bad habits that are contrary to recommended programming practices. In particular, they show a tendency to excessive decomposition and a bottom-up development process, starting with individual blocks of Scratch.

Aiming to simplify the process of evaluating computational thinking and help evaluators with an automatic tool, Hairball was created [2]. Hairball is a static analyzer of Scratch projects inspired by Lint[2] that tries to detect programming errors in Scratch projects, such as not initializing the value of a variable or the state of a character, sending messages that are not received by any program, including code that is never run, among others.

III. Methodology

In our work as instructors with high school students, we found that, despite our constant insistence and warnings, many students show on a regular basis two bad habits that have to do with object naming and code repetition:

1) Students do not change the names of the characters, which are automatically named by the environment as *SpriteX* where X is an incremental number. When the amount of characters in a project is large, it becomes very hard to know what object relates to a given statement, contributing to poor code readability, slowing the project programming, and complicating debugging to fix bugs.
2) Students repeat code in the same project, sometimes even in the same character programs. Thus, abstraction and modularization, two key components of the development of computational thinking [14], are not trained. And the *code base* becomes more complicated to maintain and update in the future.

In order to detect these two behaviors in Scratch projects automatically, we have developed following plug-ins for the Hairball environment [2], which in turn make use of *kurt*[3], a library implemented in Python. The developed plug-ins are:

1) *convention.SpriteNaming*[4] analyzes a Scratch project to check if the names of its characters begin with the string *Sprite*, indicative for the user not having changed the default name.
   As an example to illustrate the drawbacks associated with this bad programming behavior, Fig. 1 shows a script that is difficult to read because the complete character area has to be explored to find what character refers to a particular block. This makes the program difficult to debug should errors arise, making it hard to improve the code and slowing down the programming tasks.

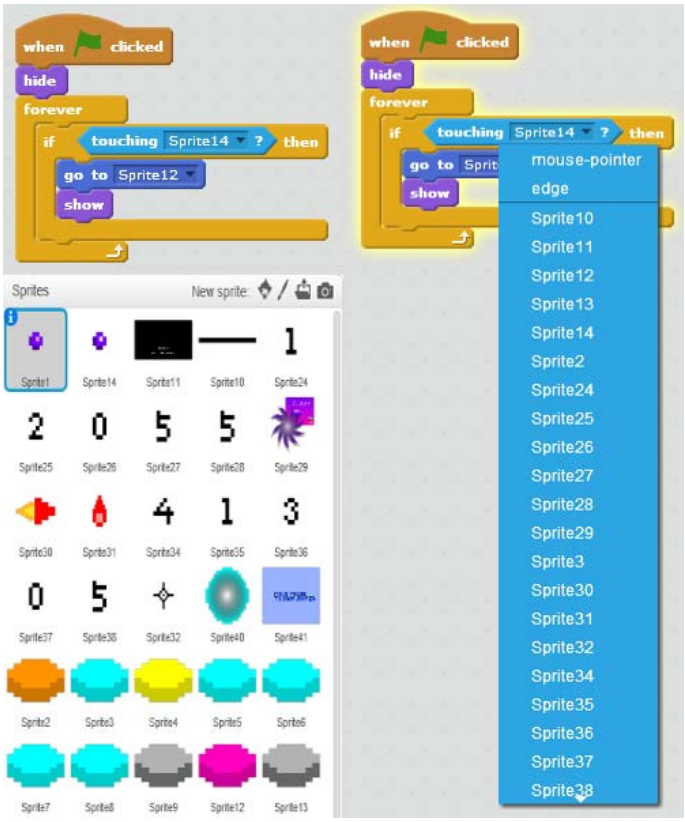

Fig. 1. Sprites with default names

2) *duplicate.DuplicateScripts*[5] analyzes a Scratch project to try to locate repeated complete programs within a project. In order to do so, the plug-in runs each script of the project and gets the *tokens* of the different blocks of the programs, so that two blocks that only differ in the values they receive are considered to be equal. A minimum length of five blocks is applied for programs to be considered duplicate rather than coincidentally similar.
   To illustrate the operation of this plug-in, Fig. 2 shows two scripts containing the same blocks with the only difference of the values received by parameter.
   The Scratch scripts are analyzed by the plug-in and translated into *tokens*. The analysis would result in considering them equal, as their code would be translated in both cases into the following sequence of *tokens*:

```
'set rotation style %s'
'repeat %s%s'
'move %s steps'
'next costume'
'if on edge, bounce'
'wait %s secs'
```

[2]http://www.unix.com/man-page/FreeBSD/1/lint
[3]https://github.com/blob8108/kurt
[4]https://github.com/jemole/hairball/blob/master/hairball/plugins/convention.py
[5]https://github.com/jemole/hairball/blob/master/hairball/plugins/duplicate.py

Fig. 2. Two scripts repeating code

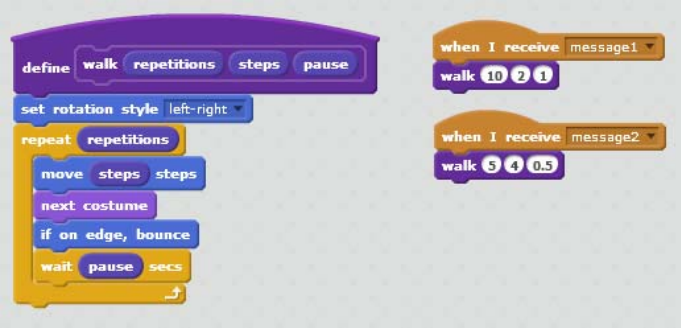

Fig. 3. Definition of blocks to avoid code repetition.

> Hence, the correct way to implement this functionality would have been as shown in Fig. 3, defining a new set of blocks that receives several values per parameter, and then reusing the block created in both programs. This solution would facilitate future maintenance, updating and debugging.

In order to test the functionality of the developed plug-ins and to verify whether these behaviors are common in projects found in the Scratch repository, we proceeded to randomly download 100 projects from the repository. As the possibility for users to define their own blocks, using *def_block* feature, is only available since the release of Scratch 2.0, we selected and downloaded projects created for this version. A replication package of the projects under study, and the specific version of the plug-ins used in it can be obtained publicly[6].

IV. Findings

The results obtained after analyzing the 100 projects confirm that observations made with our students can also be generally found in the community projects, as shown in Table I.

Regarding the lack of habit of renaming the characters and not using the default name assigned by the environment, 79% of the analyzed projects contained at least one non-renamed *SpriteX* character. The maximum number of non-renamed *SpriteX*s found has been 67, which makes the code very hard to maintain (i.e., one could consider this almost as *code obfuscation*). In mean, almost 6 *SpriteX*s can be found in a project, while the median is 3. Our results show that this is a severe problem in the analyzed projects in Scratch, as in addition to the fact that re-naming occurs in very few projects, the number of elements used in the others are significant in number; being those programs are very hard to understand, its maintenance and reuse are hindered.

As for the repetition code, 62% of the analyzed projects contained at least one repeated program. In fact, only 17% of the analyzed projects made use of block *def_block* to define its own blocks that can be reused in other parts of the project. The mean of duplicated scripts and blocks defined provides us insight about how difficult is for the authors of the analyzed projects to use abstraction and modularization: duplicated code appears over six times more than blocks.

[6]http://gsyc.urjc.es/~grex/repro/2014-fie-scratch

	Default names	Duplicated scripts	Blocks defined
Projects	79	62	17
Mean	5.94	7.23	1.11
Median	3	2	0
Maximum	67	71	25

TABLE I. Table of results

V. Conclusions and Further Research

This paper shows the preliminary results of a study regarding two bad programming habits we have detected in our work as instructors with high school students learning to program with Scratch. In order to check if these bad habits, which have to do with object naming and code repetition, are also commonly found in the projects shared in the community repository, we downloaded 100 projects and analyzed them with two plug-ins we developed for Hairball, detecting that most of the inspected projects, 79% and 62% respectively, fall into these issues.

Regarding object naming, it is worth to note that Scratch users name the variables used in their projects correctly, i.e., semantically meaningful. In our opinion the reason to explain this paradox is because it is mandatory to assign a name to a variable the moment it is created, as the environment does not name it by default. However, when creating a new character a *SpriteX* name is assigned automatically. In our views, a change of this feature of the environment, forcing users to name new objects, might eliminate this bad practice.

Regarding code repetition and the low usage Scratch users make of the possibility of creating their own blocks (the equivalent of defining methods or procedures in other languages), works as the one performed by Seiter and Foreman [12], in which they show that abstraction and modularization capacities are concepts that seem to be developed from a certain age, may explain the low rates of use of this functionality, since a significant part of the Scratch community consists of children under 10 years[7]. Future studies should further investigate how to continue advancing in the definition of an appropriate framework for different ages, as well as to check whether users learning online and those who do it in a regulated environment show different use rates of this functionality.

In the near future we plan to extend the scope of our study developing new plug-ins to automate the detection of other issues, such as the degree of development of computational thinking demonstrated in a project, and by analyzing a much

[7]http://scratch.mit.edu/statistics/#age

larger number of projects to achieve more accurate conclusions. In this sense, the release of a research dataset with the first five years of data from the Scratch repository[8] will be certainly a great help to our work. Finally, we are in the process of developing a website that will allow students to receive immediate feedback for their programs with warnings of potential errors and tips to improve their code.

ACKNOWLEDGMENTS

The work of Gregorio Robles has been funded in part by the Spanish Government under project SobreSale (TIN2011-28110) and under project "eMadrid - Investigación y Desarrollo de tecnologías para el e-learning en la Comunidad de Madrid" (S2009/TIC-1650) funded by the Region of Madrid.

REFERENCES

[1] J. C. Adams. Scratching middle schoolers' creative itch. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 356–360. ACM, 2010.

[2] B. Boe, C. Hill, M. Len, G. Dreschler, P. Conrad, and D. Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 215–220. ACM, 2013.

[3] D. Franklin, P. Conrad, B. Boe, K. Nilsen, C. Hill, M. Len, G. Dreschler, G. Aldana, P. Almeida-Tanaka, B. Kiefer, et al. Assessment of computer science learning in a scratch-based outreach program. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 371–376. ACM, 2013.

[4] Y. B. Kafai, D. A. Fields, and W. Q. Burke. Entering the clubhouse: Case studies of young programmers joining the online scratch communities. *Journal of Organizational and End User Computing (JOEUC)*, 22(2):21–35, 2010.

[5] D. J. Malan and H. H. Leitner. Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, 39(1):223–227, 2007.

[6] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):16, 2010.

[7] J. H. Maloney, K. Peppler, Y. Kafai, M. Resnick, and N. Rusk. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin*, 40(1):367–371, 2008.

[8] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari. Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pages 168–172. ACM, 2011.

[9] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari. Learning computer science concepts with scratch. *Computer Science Education*, 23(3):239–264, 2013.

[10] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[11] C. Scaffidi and C. Chambers. Skill progression demonstrated by users in the scratch animation environment. *International Journal of Human-Computer Interaction*, 28(6):383–398, 2012.

[12] L. Seiter and B. Foreman. Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 59–66. ACM, 2013.

[13] A. Wilson, T. Hainey, and T. Connolly. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *6th European Conference on Games-based Learning (ECGBL)*, pages 4–5, 2012.

[14] J. M. Wing. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3717–3725, 2008.

[8] https://sites.google.com/site/scratchdatameeting/

[15] U. Wolz, H. H. Leitner, D. J. Malan, and J. Maloney. Starting with scratch in cs 1. *ACM SIGCSE Bulletin*, 41(1):2–3, 2009.