Clones in Block-Based Languages: A Large Scale Analysis of Scratch Programs

Felienne Hermans
Delft University of Technology
Mekelweg 4
Delft, the Netherlands
f.f.j.hermans@tudelft.nl

Abstract

Recently, block-based programming languages like Alice, Scratch and Blockly have become popular tools for programming education. There is substantial research showing that block-based languages are suitable for early programming education. But do block-based programs suffer from code clones smelly *too? In a recent controlled experiment, we found that code clones in Scratch indeed hamper novice Scratch programmers. In this paper we explore how commonly code clones occur in Scratch. To that end we have scraped Scratch public repository and retrieved 250.000 programs. We have analyzed three types of clones within those programs': clones of entire scripts within sprites, clones of entire between sprites, and cloned conditions. We find that those clones occur in ... of programs.*

1 Introduction

Scratch is a programming language developed to teach children programming by enabling them to create games and interactive animations. The public repository of Scratch programs contains over 12 million projects. Scratch is a *block-based* language: users manipulate blocks to program. Block-based languages are visual languages, but also use some successful aspects of text-based languages such as limited text-entry and indentation, and as such are closer to 'real', textual programming than other forms of visual programming, like dataflow languages are.

Block-based languages have existed since the eighties, but have recently found adoption as tools for programming education. In addition to Scratch, also Alice [1], Blockly[1] and App Inventor [2] are block-languages aimed at novice programmers.

In this paper we explore code clones smells in the context of block-based languages. **hier nog wat background over clones, zie ICSE '13 uiteraard**

Clones so far have mostly been studied in the contact of object-oriented, textual source code, but some other directions have been taken, including spreadsheets and SimuLink programs.

We ourselves have recently performed an experiment where we compared the performance of novice Scratch programmers between three

Knowing that code clones can be harmful, we now turn to the question of whether they are common. As such, the goal of this paper is to **investigate whether code clones are common in public Scratch programs**. To address this goal we have obtained 250.166 public Scratch programs by scraping the list of recent programs **link**. We investigate three types of clones: clones of entire scripts within sprites, clones of entire between sprites, and cloned conditions.

The results show that **todo**

The contributions of this paper are as follows:

- A public data set of 233.514 Scratch programs (Section 4)
- A clone Scratch detection algorithm with an open source implementation (Section 5)
- An empirical evaluation of the occurrence of three types of clones in Scratch programs (Section 6)

2 Background and Motivation

Block-based languages go back to 1986, when Glinert introduced the BLOX language [8]. BLOX consists of puzzle-like programming statements that can be combined into programs by combining them both vertically and horizontally. After a decade of little activity into block-based languages, they became a research topic again, starting with Alice [1]. More recently, new block-based languages have gained widespread popularity, especially powered by Scratch [9] and Blockly[2]. Over 100 million students have

[1]`https://developers.google.com/blockly/`

[2]`https://developers.google.com/blockly/`

tried Blockly via Code.org, and the Scratch repository currently hosts over 12 million projects. Unlike in BLOX, in these new block-based languages the programming blocks can only be combined vertically, resembling textual code more.

Since their introduction, studies have demonstrated the applicability of block-based languages as a tool for education. Scratch, for example, was evaluated with a two-hour introductory programming curriculum for 46 subjects aged 14 [10]. This study indicated that Scratch could be used to teach computer science concepts: analysis of the pre- and post-tests showed a significant improvement after the Scratch course, although some concepts like variables and concurrency remained hard for students.

Moskal *et al.* [11] compared computer science students who studied Alice before or during their first programming course to students that only took the introductory computer science course. Their results show that exposure to Alice significantly improved students' grades in the course, and their retention in computer science in general over a two year period. A follow-up study by Cooper *et al.* [12] obtained similar results, showing that a curriculum in Alice resulted in improved grades and higher retention in computer science.

Most convincingly, Price and Barnes performed a controlled experiment in which students were randomly assigned to either a text-based or a block-based interface in which they had to perform small programming tasks [13]. Their experiment showed that students in the block-based interface were more focused and completed more of the activity's goals in less time.

Summarizing the above, we conclude that block-based languages have a clear potential to be a great tool for introductory programming education, in some cases even outperforming text-based languages.

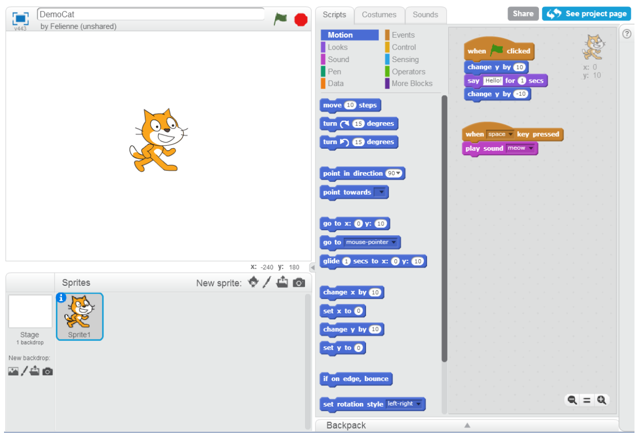
3 Relevant Scratch Concepts

This paper is by no means an introduction into Scratch programming, we refer the reader to [14] for an extensive overview. To make this paper self-contained, however, we explain a number of relevant concepts in this section.

Scratch is a block-based programming language aimed at children, developed by MIT. Scratch can be used to create games and interactive animations, and is available both as a stand-alone application and as a web application. Figure 1 shows the Scratch user interface in the Chrome browser.

3.1 Sprites

Scratch code is organized by 'sprites': two-dimensional pictures each having their own associated code. Scratch allows users to bring their sprites to life in various ways, for



```
when clicked
change y by 10
say Hello! for 1 secs
change y by -10

when space key pressed
play sound meow
```

Figure 1. The Scratch user interface consisting of the 'cat' sprite on the left, the toolbox with available blocks in the category 'motion' in the middle and the code associated with the sprite on the right. The upper right corner shows the actual location of the sprite.

example by moving them in the plane, having them say or think words or sentences via text balloons, but also by having them make sounds, grow, shrink and switch costumes. The Scratch program in Figure 1[3] consists of one sprite, the cat, which is Scratch's default sprite and logo. The code in the sprite will cause the cat to jump up, say "hello", and come back down, when the green flag is clicked, and to make the 'meow' sound when the space bar is pressed.

3.2 Events

Scratch is *event-driven*: all motions, sounds and changes in the looks of sprites are initiated by events. The canonical event is the 'when Green Flag clicked', activated by clicking the green flag at the top of the user interface. In addition to the green flag, there are a number of other events possible, including key presses, mouse clicks and input from a computer's microphone or webcam. In the Scratch code in Figure 1 there are two events: 'when Green Flag clicked' and 'when space key pressed'

3.3 Scripts

Source code within sprites is organized in scripts: a script always starts with an event, followed by a number of blocks. The Scratch code in Figure 1 has two distinct scripts, one started by clicking on the green flag and one by pressing the space bar. It is possible for a single sprite

[3] `https://scratch.mit.edu/projects/97086781/`

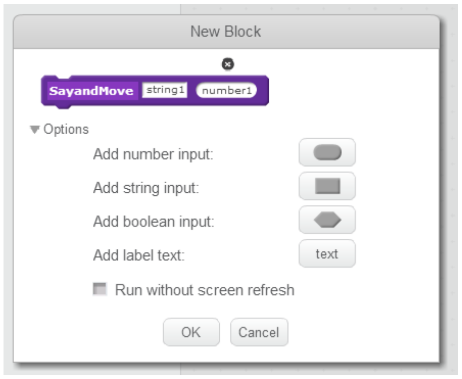


Figure 2.

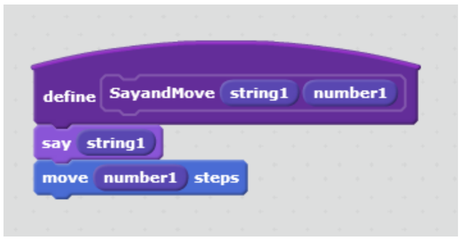


Figure 3.

to have multiple scripts initiated by the same event. In that case, all scripts will be executed simultaneously. For example, the code on the left of Figure **??** has five scripts associated with the 'when Green Flag clicked' event.

3.4 Remixing

Scratch programs can be shared by their creators in the global Scratch repository[4]. Shared Scratch programs can be 'remixed' by other Scratch users, which means that a copy of this program is placed in the user's own project collection, and can be then further changed. The 'remix tree' of projects is public, so users can track which users remix their programs, a bit similar forking in GitHub. Contrary to forking though, changes upstream cannot be integrated back into the original project.

4 Dataset

We scraped all Scratch programs uploaded **add date range?** from the Scratch website with a scraping program. This resulted in he JSON code for 250.166 Scratch programs. In addition to the programs themselves, we also gathered metadata including the numbers of views, loves, favorites and remixes. Out of the 250.166 , we failed to analyze 2.367 programs due to various technical difficulties **add details?**

[4]`https://scratch.mit.edu/explore/projects/all/`

4.1 Empty programs

Also, interestingly enough, 14.285 were empty (5.7%), at least in terms of scripts. So Scratch users were sharing program already that did not contain code yet. In some cases **analyze too?** they contained scripts and costumes but no code and in others they were entirety empty apart from the Scratch cat added by default.

4.2 Block Usage

4.3 Block Size

4.4 User-defined Blocks

4.5 Remixing

5 Clones Detection for Scratch programs

6 Results

The overarching research question of this work is how common Scratch users clone in their programs. As explained in Section 5 we distinguish three different forms of clones: Exact clones within one sprite, exact clones between sprites of one program, clones of 'wait' blocks.

6.1 Exact clones within one sprite

6.2 Exact clones between sprites of one program

6.3 Clones of 'wait' blocks

This section presents the results of our clone detection algorithm on the 233.514 with scripts.

7 Related Work

References

[1] M. Conway, R. Pausch, R. Gossweiler, and T. Burnette, "Alice: A Rapid Prototyping System for Building Virtual Environments," in *Conference Companion on Human Factors in Computing Systems*, ser. CHI '94. New York, NY, USA: ACM, 1994, pp. 295–296. [Online]. Available: http://doi.acm.org/10.1145/259963.260503

[2] D. Wolber, H. Abelson, E. Spertus, and L. Looney, *App Inventor: Create Your Own Android Apps*, 1st ed. Sebastopol, Calif: O'Reilly Media, May 2011.

[3] M. Fowler, *Refactoring: improving the design of existing code*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[4] F. Hermans, M. Pinzger, and A. v. Deursen, "Detecting and Visualizing Inter-Worksheet Smells," in *Proceeding of the 34rd international conference on Software engineering (ICSE 2012)*. ACM Press, 2012, pp. 451–460, to appear.

[5] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and refactoring code smells in spreadsheet formulas," *Empirical Software Engineering*, vol. 20, no. 2, pp. 549–575, 2014. [Online]. Available: http://link.springer.com/article/10.1007/s10664-013-9296-2

[6] K. T. Stolee and S. Elbaum, "Refactoring Pipe-like Mashups for End-user Programmers," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 81–90. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985805

[7] C. Chambers and C. Scaffidi, "Smell-driven performance analysis for end-user programmers," in *2013 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sep. 2013, pp. 159–166.

[8] E. Glinert, "Towards "Second Generation" Interactive, Graphical Programming Environments,," in *Proceedings of the IEEE Workshop on Visual Languages*, 1986.

[9] M. Resnick, J. Maloney, A. Monroy-Hernndez, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for All," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592761.1592779

[10] O. Meerbaum-Salant, M. Armoni, and M. M. Ben-Ari, "Learning Computer Science Concepts with Scratch," in *Proceedings of the Sixth International Workshop on Computing Education Research*, ser. ICER '10. New York, NY, USA: ACM, 2010, pp. 69–76. [Online]. Available: http://doi.acm.org/10.1145/1839594.1839607

[11] B. Moskal, S. Cooper, and D. Lurie, "Evaluating the Effectiveness of a New Instructional Approach," in *Proceedings of the SIGCSE technical symposium on Computer science education*, 2005.

[12] S. Cooper, W. Dann, and R. Pausch, "Teaching Objects-first in Introductory Computer Science," in *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '03. New York, NY, USA: ACM, 2003, pp. 191–195. [Online]. Available: http://doi.acm.org/10.1145/611892.611966

[13] T. W. Price and T. Barnes, "Comparing Textual and Block Interfaces in a Novice Programming Environment," in *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ser. ICER '15. New York, NY, USA: ACM, 2015, pp. 91–99. [Online]. Available: http://doi.acm.org/10.1145/2787622.2787712

[14] K. Brennan, C. Balch, and M. Chung, *CREATIVE COMPUTING*. Harvard Graduate School of Education, 2014.