Abscratchion: Enhacning Scratch with higher-level custom blocks

Subtitle Text, if any

Name1
Affiliation1
Email1

Name2 Name3
Affiliation2/3
Email2/3

Abstract

Several recent studies have shown that Scratch is an appropriate language for introductory programs. But, is Scratch powerful enough to express everything users want to express? We have analyzed 250.000 programs from the public Scratch repository to find commonly occurring code patterns. We found ? that together occurred in ?. Some of the patterns were clearly game related, like 'change this characters costume when this happens' while others were generic, like 'perform this action until a key is released'. We see these commonly occurring blocks as opportunities to make Scratch powerful, and to that end we have implemented them as custom blocks and subsequently ran a study with ? demonstrating...TODO!!!.

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms terms

Keywords Scratch, block-based languages, programming practices, code smells, static analysis

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 14 million projects. Scratch is a *block-based* language: users manipulate blocks to program.

Block-based languages have existed since the eighties, but have recently gained traction as language for programming education. In addition to Scratch, also Alice (**?**), Blockly[1] and App Inventor (**?**) are block-languages aimed at novice programmers.

Several studies have shown that block-based languages are powerful as tools for teaching programming (**????**). Previous works involving static analysis of Scratch programs have evaluated the application of various programming concepts in Scratch projects (**??**). Recent works have focused on bad programming practices within Scratch programs (**?**), and automated quality assessment tools have been proposed for identifying code smells (**?**) and bad programming practices (**??**). A recent controlled experiment found that long scripts and code duplication decreases a novice programmer's ability to understand and modify Scratch programs (**?**).

The goal of this paper is twofold. Firstly, we aim to understand which combinations of Scratch blocks are commonly used together. To that end we run a clone detection algorithm over a previously released dataset of 250.000 Scratch programs, resulting in ?.

Subsequently, we investigate whether offering some of those combinations as built-in blocks will increase novice Scratch programmers productivity and programming enjoyment. To that end we run a controlled experiment with ?middle-school aged kids. The findings indicate TODO!!!. To the best of our knowledge, this is the first study into the effects of abstraction on the Scratch programming language.

The contributions of this paper are as follows:

- An analysis of commonly occurring block combinations in the previously released open dataset (Section **??**)
- A manual classification of those patterns into groups (Section 4)
- A controlled experiment researching the applicability of the patterns (Section 5)

2. Relevant Scratch Concepts

This paper is by no means an introduction into Scratch programming, we refer the reader to (**?**) 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 and as a web application. Figure 1 shows the Scratch user interface. The main concepts in the Scratch programming environment are:

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 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 contains two one sprites, the cat, which is Scratch's default sprite and a piano. The code in Sprite1 will cause the cat to move right when the right arrow is pressed, and when the green flag is clicked it will continuously sense touching the piano.

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

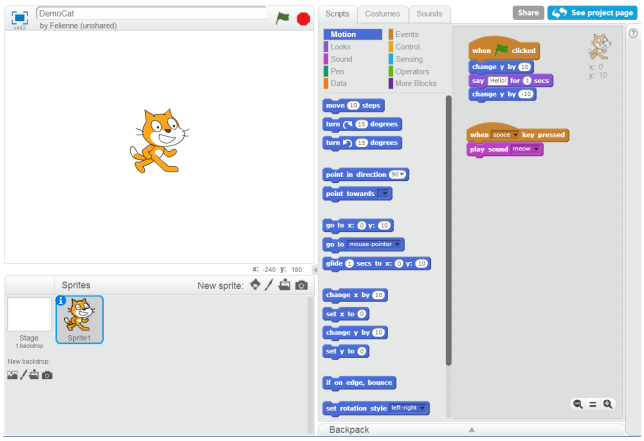

Figure 1. The Scratch user interface consisting of the 'cat' sprite on the left, the toolbox with available blocks in the category 'Events' in the middle and the code associated with the sprite on the right.

Scripts Sprites can have multiple code blocks, called scripts. 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 to have multiple scripts initiated by the same event. In that case, all scripts will be executed simultaneously.

Events Scratch is *event-driven*: all motions, sounds and changes in the looks of sprites are initiated by events called Hat blocks[2]). 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. The Scratch code in Sprite1 in Figure 1 contains two events: `when Green Flag clicked` and `when <right arrow> key pressed`, each with associated blocks.

Signals Events within Scratch can be user generated too: users can broadcast a message, for example when two sprites touch each other, like in Figure 1. All other sprites can then react by using the `when I receive` Hat block. In Figure 1, Sprite1 broadcasts 'bump' when the cat touches the Piano.

Custom blocks Scratch users can define their own blocks, which users can name themselves, called custom blocks. The creation of custom blocks in Scratch is the equivalent of defining procedures in other languages (**?**). Because the term 'procedures' is common in related work, we will refer to custom blocks as 'procedures' in the remainder of this paper. Procedures can have input parameters of type string, number, and boolean. When a user defines a procedure, a new Hat block called `define` appears, which users can fill with the implementation of their block.

3. Scratch source code patterns

The main focus of this study is to understand how people program in Scratch by analyzing the characteristics of Scratch projects. To answer our three research questions, we conducted an empirical quantitative evaluation of project data we collected from the Scratch project repository. In the following paragraphs we describe the dataset, the process and the tools we used for analyzing it, and the methods we followed for detecting code smells.

3.1 Dataset

3.2 Pattern Analysis

TODO!!!Fenia, can you explain how exactly obtained the blocks?

3.3 Results

The above described algorithm resulted in ? patterns, the most common ones are shown in table TODO!!!. The full set is available here: TODO!!!

4. Coding

While investigating the results of the pattern finding algorithm, we found many patterns differed only slightly from each other. We therefore decided to use TODO!!!describe coding in more detail.

4.1 Approach

The two authors of this paper coded the blocks individually and then compared notes TODO!!!etc.

4.2 Results

The above described coding resulted in ?, of which we found ... to be game related and ... generic programming blocks. Of those, we used ... in the subsequent experiment.

5. Experiment

5.1 Setup

5.1.1 Context

5.1.2 Participants

5.1.3 Selected Blocks

A. Appendix Title

This is the text of the appendix, if you need one.

Acknowledgments

We would like to that the schools.... TODO!!!

References

P. Q. Smith, and X. Y. Jones. ...reference text...

[2] `http://wiki.scratch.mit.edu/wiki/Hat_Block`