Using Pirate Plunder to Develop Children's Abstraction Skills in Scratch

Simon P. Rose
Sheffield Hallam University
Sheffield, UK
simon.rose@shu.ac.uk

M. P. Jacob Habgood
Sheffield Hallam University
Sheffield, UK
j.habgood@shu.ac.uk

Tim Jay
Sheffield Hallam University
Sheffield, UK
t.jay@shu.ac.uk

ABSTRACT

Scratch users often struggle to detect and correct 'code smells' (bad programming practices) such as duplicated blocks and large scripts, which can make programs difficult to understand and debug. These 'smells' can be caused by a lack of abstraction, a skill that plays a key role in computer science and computational thinking. We created Pirate Plunder, a novel educational block-based programming game, that aims to teach children to reduce smells by reusing code in Scratch. This work describes an experimental study designed to measure the efficacy of Pirate Plunder with children aged 10 and 11. The findings were that children who played the game were then able to use custom blocks (procedures) to reuse code in Scratch, compared to non-programming and programming control groups.

KEYWORDS
computational thinking; Scratch; Pirate Plunder; visual programming; computer science education; abstraction;

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CHI'19 Extended Abstracts, May 4–9, 2019, Glasgow, Scotland UK

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5971-9/19/05.
https://doi.org/10.1145/3290607.3312871

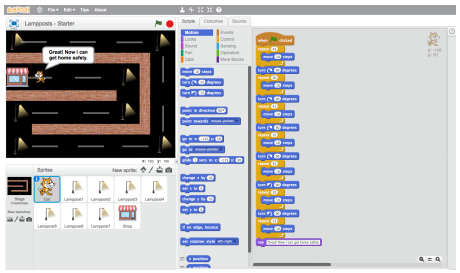

Figure 1: A Scratch project that does not use abstraction techniques.

INTRODUCTION

Computer science is becoming widespread in both primary and secondary education (ages 5 to 16) worldwide [6]. It is being pushed by both policymakers [e.g. 14] and the technology industry [3] in order to produce 'digital citizens' for an increasingly IT-based global economy. The lives of today's children will be greatly influenced by computing, both in the home and at work.

One of the main arguments behind teaching computer science to children is that the 'computational thinking' skills developed through programming are useful in a wider context [16]. Current definitions of computational thinking involve working at multiple levels of abstraction, writing algorithms, understanding flow control, recognising patterns and decomposing problems [e.g. 12].

A variety of block-based programming tools have been created to help teach programming to children. Scratch is one of the most widely-used of these tools. It is a block-based visual programming environment that can be used to create stories, animations and games. We described the design of a novel educational programming game, Pirate Plunder, in an earlier work [11]. The game aims to teach players how to reuse code in Scratch using repeat blocks (loops), custom blocks (parametrised procedures) and clones (instances of sprites). Using these blocks correctly involves abstracting duplicated functionality into reusable program components.

This work describes an experimental study designed to measure the efficacy of Pirate Plunder. We start with a summary of the background and rationale for the game, then explain the methodology before moving on to the results and discussion.

BACKGROUND

Scratch

Block-based programming tools (e.g. Scratch) are now widely-used to teach programming, particularly in primary education (5 to 11 years old). This is because they allow novices to program without learning syntax or memorising commands, unlike text-based tools.

Scratch (Figure 1) is designed for children aged 8 and above. It encourages a constructionist [9] bottom-up approach, where solutions are unplanned and created mostly through exploration. However, this approach can result in bad programming practices (known as 'code smells') because software engineering concepts like code reuse are not formally introduced. Examples of code smells in Scratch include large scripts, dead blocks (not attached to an event block) and duplicated blocks [7]. These problems are not helped by the limited computer science teaching expertise in primary education.

Code Smells & Abstraction

A code smell is a surface indication in a program that usually corresponds to a deeper problem, for example, duplicated code, long methods and long parameter lists [5]. These 'smells' can make

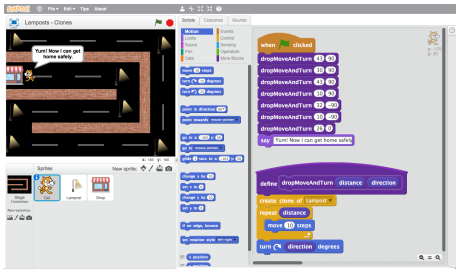

Figure 2: A Scratch project that uses abstraction techniques.

Figure 3: A Pirate Plunder level that requires a custom block with two inputs (parameters) and cannonball sprite cloning.

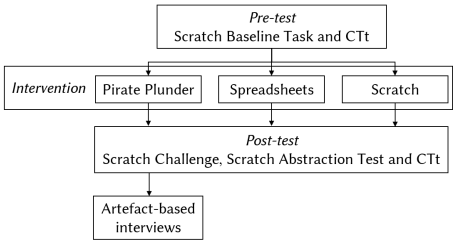

Figure 4: The step-by-step process of the study procedure.

programs difficult to understand, debug and maintain, even in Scratch [7]. Furthermore, there are suggestions that novice programmers "prone to introducing some smells do so even as they gain experience" [13, p. 10], meaning that children should be taught to detect and correct code smells early.

The concept of abstraction is the main tenet of computational thinking [16] and is key in computer science, yet it is difficult to teach to novices [2]. Abstractions are used by computer scientists to arrange programs using reusable components, such as procedures and classes. These are often used when refactoring (or restructuring) existing code, for example creating or editing procedures to generalise duplicated functionality.

In Scratch, block and sprite duplication and script size can be reduced using repeat blocks (loops), custom blocks (parameterised procedures) and clones (instances of sprites), yet the latter two are rarely used [1]. An example of this is shown in Figure 2, which produces the same result as Figure 1 but uses a custom block and cloning of the lamppost sprite. Dr. Scratch [8] is a tool for measuring computational thinking in Scratch projects. It measures abstraction and decomposition skills through the use of multiple scripts in multiple sprites (decomposition), custom blocks and clones (both abstraction).

Pirate Plunder

Pirate Plunder (Figure 3) is a novel educational block-based programming game designed for children aged 9 and above. It aims to teach code reuse using loops (repeat blocks), parameterised procedures (custom blocks) and instances (clones) in a game-based Scratch-like setting. Players use Scratch blocks to program a pirate ship to navigate around a grid, collect items and interact with obstacles. Players progress through a difficulty progression that forces them to duplicate code before introducing a strategy or block (loops, procedures or instances) to remove this duplication through code reuse in future levels. This concept is introduced solely through the game, with researchers and teachers only giving support to individual players when necessary. We discuss the design of Pirate Plunder and how it supports teaching abstraction in more detail in earlier work [11].

METHODOLOGY

Participants

The participants were 91 children (45 male and 46 female) aged between 10 and 11 (M = 10.58, SD = .32) from a large primary school in the north of England.

Procedure

In this work, we report the pre-to-mid-test results obtained as part of a larger study. The participants were assigned to three conditions after the pre-test and each given a 6-hour teaching intervention over 4 weeks of either Pirate Plunder, spreadsheets (non-programming control) or Scratch (programming

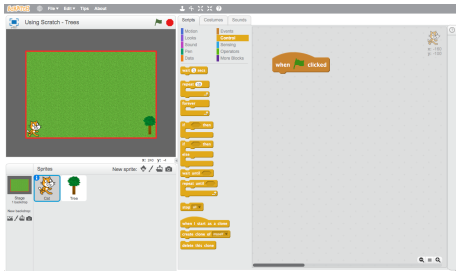

Figure 5: A starter project for the Scratch baseline task.

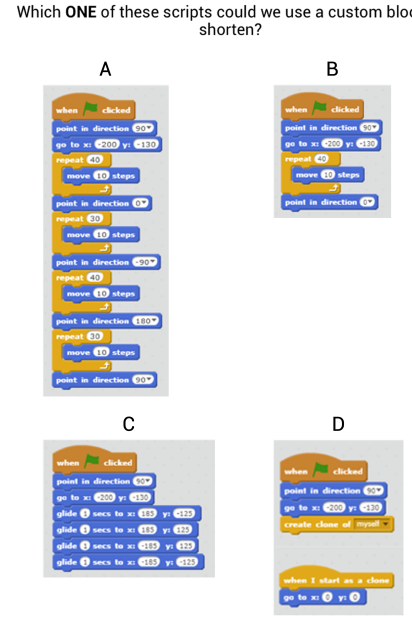

Figure 6: A sample question from the Scratch abstraction test.

control) (Figure 4). The control groups were active to make them more comparable in terms of researcher contact during the training period and to reduce motivational differences between groups.

Materials

Other than the Pirate Plunder game, the study used three assessments to measure participants' use of custom blocks and clones in Scratch, a computational thinking test and two 6-lesson ICT curricula:

Scratch Baseline Task. At pre-test, participants were asked to complete a programming task in Scratch that involved animating an object around the sides of a rectangle and leaving another object on each corner (Figure 5 shows one of the starter projects). This was designed to allow participants to demonstrate their Scratch proficiency, but also to let them use custom blocks and clones if they were able because the solutions often involved block and sprite duplication. Participants were given 40 minutes for both this and the Scratch challenge.

Scratch Challenge. At post-test, participants were asked to reduce the block count in an existing Scratch project that contained both duplicated blocks and sprites (the project in Figure 1). The project involved animating an object around a map and having it leave an item on each corner (the ideal solution would use custom blocks and clones as seen in Figure 2). We then conducted artifact-based interviews [4] for the Pirate Plunder group to see if participants had understood the rationale behind their solutions and if they could explain this in contexts outside Pirate Plunder.

Scratch Abstraction Test. At post-test, participants were given a 10-question multiple-choice assessment designed by the first author on correctly using custom blocks and clones in Scratch (Figure 6 shows a sample question).

Computational Thinking Test. The CTt [10] was used at pre-and post-test to measure participant computational thinking ability. It contains 28 multiple-choice questions that use visual arrows or blocks common in educational programming tools.

Control Group Activities. The control activities were age-appropriate 6-week curricula produced by an educational resources company [15]. These involved creating animations in Scratch (without custom blocks or clones) and using spreadsheets for data entry, basic analysis, sorting and graphs.

Hypotheses

We hypothesised that the Pirate Plunder group would achieve better results on both the Scratch challenge and the Scratch abstraction test in comparison to the control groups. Secondly, that the Pirate Plunder group would improve on the CTt in comparison to the non-programming control group, who were not doing explicit computational thinking (in line with the literature) during their activity.

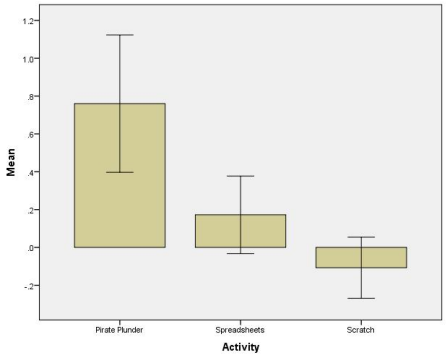

Figure 7: Dr. Scratch abstraction and decomposition learning gains between pre- and post-test.

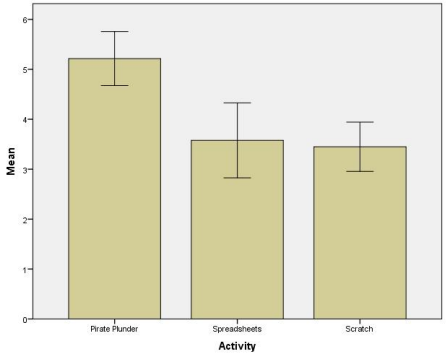

Figure 8: Scratch Abstraction Test scores at post-test.

RESULTS

Scratch Challenge

The finished Scratch projects were analysed using Dr. Scratch for use of abstraction and decomposition; 1 point for using multiple scripts in multiple sprites (the starter project achieved this), 2 points for using custom blocks, and 3 points for using clones. Figure 7 shows the mean learning gains between the Scratch baseline task (pre-test) and Scratch challenge projects (post-test) for each group. A one-way ANOVA showed a significant difference in learning gains between the three groups ($F(2, 79) = 12.9$, $p < .001$, $\eta_p^2 = .25$), with the Pirate Plunder group ($M = 0.76$, $SD = 0.88$) improving significantly compared to both controls (non-programming: $M = 0.17$, $SD = 0.54$ and programming: $M = -0.11$, $SD = 0.42$).

Scratch Abstraction Test

There was a significant difference in the Scratch Abstraction Test scores between the three groups ($F(2, 80) = 11.64$, $p < .001$, $\eta_p^2 = .23$). The Pirate Plunder group scored significantly higher ($M = 5.21$, $SD = 1.4$) than both the non-programming control ($M = 3.58$, $SD = 1.86$) and the programming control ($M = 3.45$, $SD = 1.3$) (Figure 8).

Computational Thinking Test

There was a significant difference in the CTt learning gains between the three groups ($F(2, 84) = 3.72$, $p = .028$, $\eta_p^2 = .081$) (Figure 9), with the only significant pairwise comparison between the Pirate Plunder group ($M = 3.07$, $SD = 3.22$) and the non-programming control ($M = .20$, $SD = 5.1$); $t(55) = -2.87$, $p = .015$, $d = 0.67$.

DISCUSSION

The results of both the Scratch Challenge and Scratch Abstraction Test indicate that Pirate Plunder is effective in teaching children to reuse code using custom blocks and clones in Scratch, compared to a programming (Scratch) and a non-programming (spreadsheet) activity. The mean Dr. Scratch abstraction and decomposition score of 1.82 ($SD = .61$) for the Pirate Plunder group shows that most participants used custom blocks in the Scratch challenge at post-test (20/28).

The artifact-based interviews suggested that higher-scoring participants could apply the concept of code reuse to other situations in Scratch, but lower-scoring participants struggled to separate the use of custom blocks and clones from the context of Pirate Plunder, even if they had used them successfully in the Scratch challenge.

The CTt results suggest that Pirate Plunder improves computational thinking compared to a non-programming curriculum, but not compared to the programming curriculum that involves some computational thinking, as hypothesised.

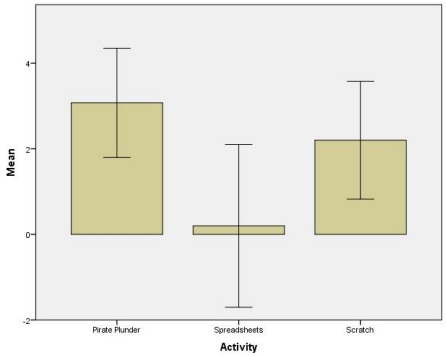

Figure 9: CTt learning gains between pre- and post-test.

CONCLUSION

In conclusion, the results of this study suggest that Pirate Plunder is effective in teaching children to reduce code smells (duplication and long scripts) in Scratch using repeat blocks, custom blocks and clones (to some extent). The next step is to analyse the data from the full study, both the assessment tasks and Pirate Plunder itself, to provide a more detailed explanation of why and how it is effective.

REFERENCES

[1] Efthimia Aivaloglou, Felienne Hermans, Jesús Moreno-León, and Gregorio Robles. 2017. A Dataset of Scratch Programs: Scraped, Shaped and Scored. In *Proceedings of the 14th International Conference on Mining Software Repositories*. 511–514. https://doi.org/10.1109/MSR.2017.45

[2] Michal Armoni. 2013. On Teaching Abstraction in Computer Science to Novices. *Journal of Computers in Mathematics and Science Teaching* 32, 3 (2013), 265–284.

[3] Paulo Blikstein. 2018. *Pre-College Computer Science Education: A Survey of the Field.* Technical Report. Google. 45 pages.

[4] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*. 1–25.

[5] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional. 1–337 pages. https://doi.org/10.1007/s10071-009-0219-y

[6] Fredrik Heintz, Linda Mannila, and Tommy Farnqvist. 2016. A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education. In *Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE)*. 1–9. https://doi.org/10.1109/FIE.2016.7757410

[7] Felienne Hermans and Efthimia Aivaloglou. 2016. Do Code Smells Hamper Novice Programming? A Controlled Experiment on Scratch Programs. In *Proceedings of the IEEE 24th International Conference on Program Comprehension (ICPC)*. 1–10. https://doi.org/10.1109/icpc.2016.7503706

[8] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2015. Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *Revista de Educación a Distancia (RED)* 46 (2015). https://doi.org/10.6018/red/46/10

[9] Seymour Papert. 1980. *Mindstorms: Children, computers, and powerful ideas.* Basic Books, Inc.

[10] Marcos Román-González, Juan-Carlos Pérez-González, and Carmen Jiménez-Fernández. 2016. Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior* 72 (2016), 678–691. https://doi.org/10.1016/j.chb.2016.08.047

[11] Simon P. Rose, M. P. Jacob Habgood, and Tim Jay. 2018. Pirate Plunder: Game-Based Computational Thinking Using Scratch Blocks. In *Proceedings of the 12th European Conference for Game Based Learning*. 556–564.

[12] Valerie J. Shute, Chen Sun, and Jodi Asbell-Clarke. 2017. Demystifying computational thinking. *Educational Research Review* 22 (2017), 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

[13] Peeratham Techapalokul and Eli Tilevich. 2017. Understanding Recurring Software Quality Problems of Novice Programmers. In *Proceedings of the 2017 IEEE Symposium on Visual Languages and Human-Centric Computing*. 43–51.

[14] The Royal Society. 2012. *Shut down or restart? The way forward for computing in UK schools.* Technical Report. London. 122 pages.

[15] Twinkl Educational Publishing. 2018. Computing Year 6. https://twinkl.co.uk.

[16] Jeannette M. Wing. 2006. Computational Thinking. *Commun. ACM* 49, 3 (2006), 33. https://doi.org/10.1145/1118178.1118215