PROGRAMMING FOR YOUNG CHILDREN USING TANGIBLE TILES AND CAMERA-ENABLED HANDHELD DEVICES

A. Cardoso[1], A. Sousa[2], H. Ferreira[2]

[1]FEUP (PORTUGAL)
[2]INESC TEC + FEUP (PORTUGAL)

Abstract

Schools are trying to teach programming at an earlier age, but there are some difficulties, namely the cost of having enough computer stations for the kids. We present the tangible system Tactode for young students to learn to program in the classroom, using handheld camera devices. The system was tested with a small focus group of students between 10 and 12 years old, that were asked to draw a regular polygon using the Scratch cat. All students completed the required task although some required help. Both students and teachers reported that they thoroughly enjoyed the experience and would like to repeat. In questionaries following the activities, the students declared that they found the language easy to use, with only 14% deeming it somewhat difficult. We consider these early results encouraging as well as informative for future developments.

Keywords: Programming, Programming Teaching, Programming Learning, Technology, Technologies for Education, Technology in the Classroom.

1 INTRODUCTION

Teaching children to program has several advantages, even more so due to the ubiquity of technology. For example, Chetty [1] found evidence that learning to program at an earlier age leads to future interest in Computer Science. Also, has Resnick et. al [2] pointed out, learning to program expands what children can do with a computer, the range of what they can learn and their problem solving and design capabilities.

Schools have recognized the need and early age programming has entered educational curricula. However, there are some difficulties. Mainly the insufficient availability of technology, as many schools do not have enough computers for all students to use for long periods or in several classes. We consider it more likely for schools to adopt (less expensive) tablets and for the students to be early adopters of smartphones that can be used for education in the classroom.

Inexpensive devices are typically small screen, no keyboard, handheld devices. Using common computer tools for programming in such devices is awkward and frustrating. A large number of visual solutions exist but are hard to manipulate in small screens, are not very flexible and many do not produce generic code, usable elsewhere outside the programming environment.

Besides some well known block based programming languages aimed at children (such as Scratch [6]), there has been increasing evidence supporting the use of tangible languages in the classroom for younger students. Hort et al. [7] present two tangible programming languages as well as the practical advantages of this approach, with initial studies revealing that children between the ages of 6 and 7 were capable of designing chains of actions, with some even finding bugs. In a subsequent study [8], the authors conducted a museum experiment, where the visitors classified the tangible and graphical interfaces as equally easy to use, but were significantly more likely to interact and collaborate with the tangible interface. In [9] Horn et al. propose a hybrid approach between tangible and graphical interfaces in which teachers and students chose the best fit for each situation, given that both have strengths and limitations. Sapounidis et al. [10] conducted a formal study with 109 children between the ages of 6 and 12, where they compared the same programming language with two different interfaces: tangible and graphical. The results showed that children using the tangible interface made less errors, were more likely to effectively debug their errors and, in the case of the younger children, needed less time to accomplish robot programming tasks. Also, when interacting freely, the older children were more engaged, designed programs with higher complexity and used a wider variety of

commands, with the tangible interface.

Unfortunately, the usage of tangible languages is (a) rare, (b) frequently expensive, because they usually rely on electronic components, or (c) limited in terms of capabilities. When it comes to finding a multipurpose tangible language that could be used with different paradigms and robots, our literature review revealed fruitless.

In this work, we propose Tactcode, a programming system based on tangible tiles, similar to puzzle pieces, that can be assembled in a very open and flexible manner in order to create programs that can be exported to different platforms. Given their physical nature, the tiles are adequate for team cooperation and do not require a dedicated computer. They have also been conceptually designed to support different paradigms of programming, as well as different approaches for different age ranges. Additionally, Tactcode reuses the same tiles to target different platforms and robots already available on the market.

To avoid the usage of expensive electronics, we rely on a simple photography of the assembled tiles, that can be taken with a common smartphone. Our software then automatically recognises the tiles and compiles the final program targeting the desired platform. Depending on the platform itself, all computing power can be undertaken by the same smartphone or tablet used for the photograph, and shared along the classroom.

Although Tactcode is currently a proof of concept, the existing implementation already supports a few different target platforms, and provides several working examples. This allowed us to conduct preliminary, quasi-controlled experiments, in order to assess our original hypothesis by measuring their reactions, and simultaneously provide us feedback for future developments. We can conclude that the students overall enjoyed working with the system, and managed to successfully complete the assignments we gave them. We thus believe to have sufficient evidence that our approach can eventually address some (if not most) of the aforementioned difficulties of teaching programming in early education.

2 THE TACTODE PROGRAMMING SYSTEM

Tactode has two main components: a set of puzzle like tiles, each containing an instruction for the robot; and a mobile application that is used to capture the assembled tiles through a photograph and translate them into executable code that is then sent to the destination platform.

As the example in Fig. 1 shows, each tile has three elements: a title that sums what the tile does, an image that illustrates it, and an Aruco [7] tag that allows the tile to be recognized in the photograph.

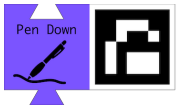

Figure 1. The Tactode tile instructing the Scratch cat to use the pen to write.

The mobile application has a very simple interface. In addition to the settings and the previous programs database, there is a main tab dedicated to capturing (through photograph) and exporting new programs. If the program is successfully captured and it contains no errors, then the students are able to export it. In contrast, when programming errors are detected, they are reported by highlighting the corresponding tile in the picture and showing an error message.

The settings allow the user to choose between getting the picture from camera or from a file and also to specify the robot or platform to be used. Currently, three robots, namely Cozmo, Ozobot Evo and Robobo [8] are supported, plus Scratch and Python as the non-robotic platforms. In terms of computing platforms, the application works on Android, iOS, macOS and Windows.

In the background, the application uses a compiler to translate the pictures into executable code. This compiler is divided into three components. The first component uses the Aruco JavaScript library to detect and organize the tags according to their relative position. The second component generates an abstract syntax tree of the code and is also responsible for detecting errors. The third component is the code generator, and it heavily depends on the output platform, since each robot/system has its

own language.

3 METHODOLOGY

In preparation for the experiments, a guide for the teachers was created, detailing the process of drawing a regular polygon with the Scratch cat. The guide contains a small introduction to programming, focusing on block based programming and on how it significantly simplifies learning. Both the languages Scratch and Tactode are presented, although only the blocks pertaining to the task at hand are detailed. There is also an explanation about regular polygons and their internal and external angles. After these introductory sections, the process of programming the solution in both languages is described in constructive steps.

The choice of the regular polygon activity was based on its appeal, its simplicity, the connection to mathematics and the programming concepts involved, namely loop blocks. Also, Scratch was selected because it allows for some comparison with our language.

The experiment was conducted with 14 students between the ages of 10 and 12. The students where organized into groups of 2 or 3 elements and the objective of the experiment was introduced by their teacher. They where told about the Tactode tiles and application, the Scratch environment, that they would use, and about a strategy for drawing regular polygons. Then, they proceeded to their task of trying to assemble the tiles in order to make the Scratch cat draw a polygon as it walks.

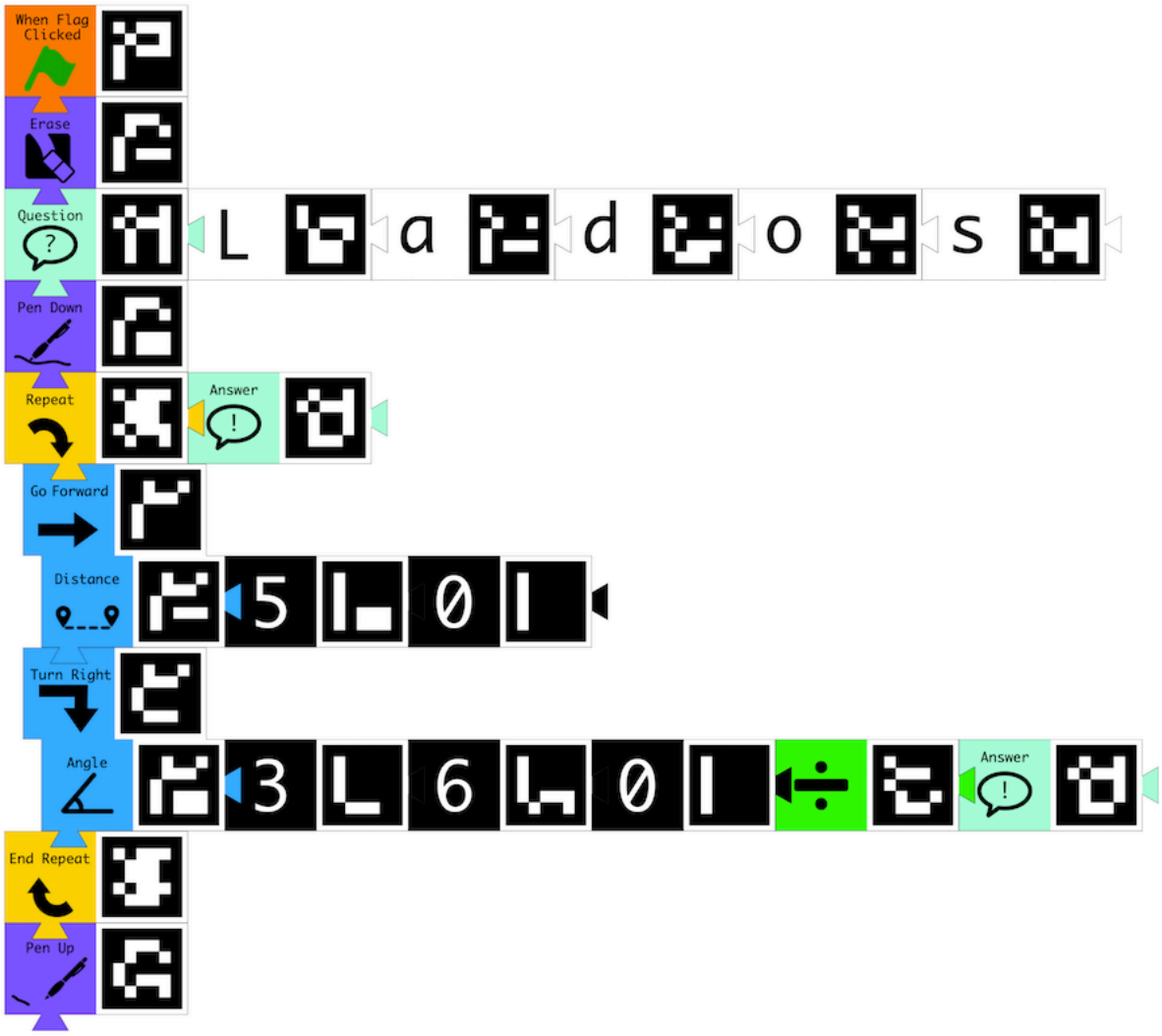

Figure 2. Code for drawing a regular polygon in Tactode.

Fig. 2 shows the final code for drawing a regular polygon, but the students where given the task in smaller steps. First, organize the tiles to make the cat draw a square, without using any repeat block.

Second, understand that for polygons with more sides the task becomes tiresome and the code very long, thus the need for repeat blocks. Third, place the same number in front of repeat and after the divide sign, change this number a few times to obtain different polygons. Fourth, use the question/ answer tool to make a general program where the user decides how many sides (*lados* in Portuguese) the polygon should have.

During our test, the students were divided into two different types of groups. The first group used the Tactode system, while the second used the Scratch system directly. The students then traded places, so that they could test the alternative. There was also some time devoted to free exploration.

4 RESULTS

The total experiment lasted nearly 2 hours, including material distribution, explanations, group activities and questionaries.

All the students completed the basic steps of the required tasks, although some groups required more assistance, especially in the first steps. The final step was not completed by several groups, but this was mainly due to the teacher deeming it too difficult without trying. Figs. 3 and 4 show the students taking a photograph of the completed code and then executing it with Scratch.

Figure 3. Students capturing the code.

In enquiries following the activities, students were asked 5 multiple choice questions. Tables 1 and 2 show the results.

Table 1. Questionnaire results, first part

Question	1	2	3	4	5
Did you understand the objective of the activity? (1 - nothing, 5 - completely)	-	-	-	4	10
Did you find the Tactode language easy to use? (1 - very hard, 5 - very easy)	-	-	3	6	5
Did you find the Tactode application easy to use? (1 - very hard, 5 - very easy)	-	-	-	10	4

Table 2. Questionnaire results, second part

Question	Tactode	Scratch
Which language did you find easier?	5	9
Which language did you prefer to use?	5	9

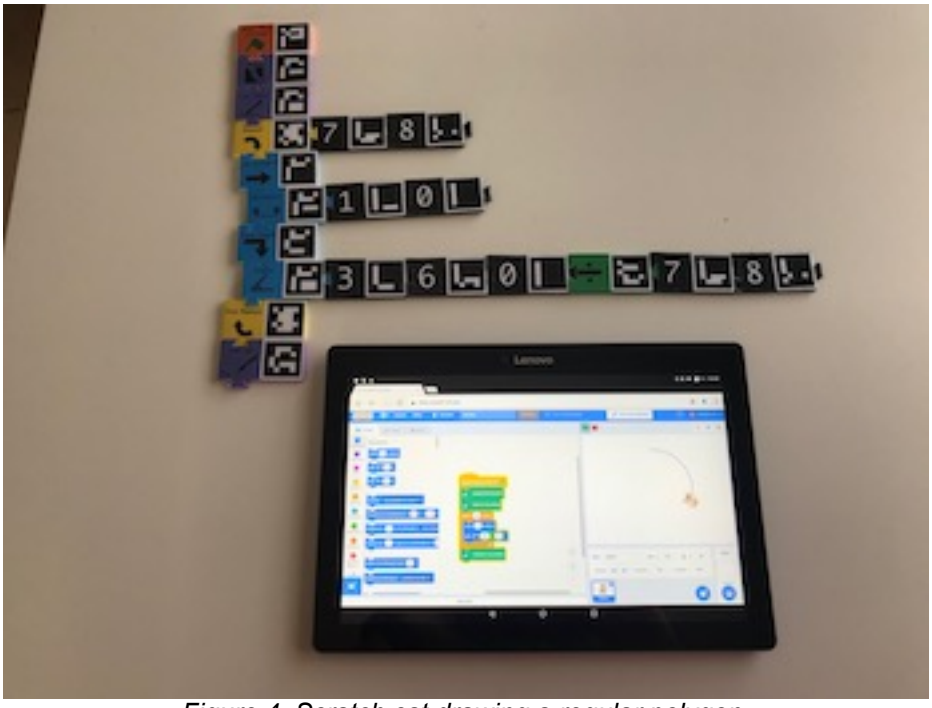

Figure 4. Scratch cat drawing a regular polygon.

Observing the students while they were performing their tasks, there seemed to be more focus in the groups using Tactode, while the other groups experimented more with different commands and objectives.

5 CONCLUSIONS

Given the number of students, our experiment is not statistically relevant. But that was not our goal with this test, we simply wanted to see if the current implementation is working, if the students manage to use it with minimal help and what future developments are more important. In those aims, we were successful. Particularly, it became quite clear that the system works and that students learn while enjoying themselves.

We learned that the quality and speed of the image capturing process needs to be improved since that was one of the main difficulties of the students. Also, the steps necessary to get the compiled code executed could be reduced. Arguably this depends on each platform developer as much as it depends on us, but if nothing else can be achieved, it is worth considering having execution directly in our application similarly to what Scratch does. This because a considerable justification given by students for their preference of Scratch was how fast and easy it was to execute.

Of course it is important to keep in mind that the students in this experiment were using tablets or computers, so they had plenty of screen to conveniently use the drag and drop interface of Scratch. With smartphones this quickly changes and the only thing missing from our system is its own simulator for execution.

Although the students seemed to prefer the graphical interface, particularly the older ones, this would not have been possible if they were using smartphones instead of tablets and computers. Thus, there is still the compelling argument of cost on the side of the tangible interface. In any case, as suggested

in [5], there is probably more to gain from a hybrid approach.

ACKNOWLEDGEMENTS

This paper was financed by FEDER funds, through the Programa Operacional Competitividade e Internacionalização – COMPETE 2020, in the scope of project POCI-01-0145-FEDER-006961, and by Nacional Funds, through the FCT – Fundação para a Ciência e a Tecnologia, in the scope of project UID/EEA/50014/2013.

REFERENCES

[1] J. Chetty, “Combatting the War Against Machines: An Innovative Hands-on Approach to Coding” in *Robotics in STEM Education* (Khine M. eds.), pp. 59 - 83. Cham: Springer International Publishing, 2017.

[2] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. “Scratch: Programming for all”, Communications of the ACM, vol. 52, no. 11, pp. 60 - 67, 2009.

[3] M. S. Horn and R. J. K. Jacob, “Designing tangible programming languages for class room use”, *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, pp. 159 - 162, 2007.

[4] M. S. Horn, E. T. Solovey, R. J. Crouser, and R. J. K. Jacob, “Comparing the use of tangible and graphical programming languages for informal science education”, *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 975 - 984, 2009.

[5] M. S. Horn, R. J. Crouser, and M. U. Bers, “Tangible interaction and learning: the case for a hybrid approach”, *Personal and Ubiquitous Computing*, vol. 16, no. 4, pp. 379 - 389, 2012.

[6] T. Sapounidis, S. Demetriadis, and I.Stamelos, “Evaluating children performance with graphical and tangible robot programming tools”, *Personal and Ubiquitous Computing*, vol. 19, no. 1, pp. 225 - 237, 2015.

[7] F. J. Romero-Ramirez, R. Muñoz-Salinas, R. Medina-Carnicer, "Speeded up detection of squared fiducial markers", *Image and Vision Computing*, vol. 76, pp. 38 - 47, 2018.

[8] F. Bellas, M. Naya, G. Varela, L. Llamas, A. Prieto, J. C. Becerra, M. Bautista, A. Faiña, and R. Duro, “The Robobo project: Bringing educational robotics closer to real-world applications” in *Robotics in Education* (W. Lepuschitz, M. Merdan, G. Koppensteiner, R. Balogh, and D. Obdržálek eds.), pp. 226–237, Cham: Springer International Publishing, 2018.