Following the Line from Blocks Programming to Robotics

Alexandra-Emilia FORTIS, Teodor-Florin FORTIS

West University of Timișoara,
4 Vasile Parvan Bd., Timișoara, Romania
alexandra.fortis[at]e-uvt.ro

Abstract: *Computational thinking has developed into one of the basic abilities to be achieved during childhood, and not only. Industry 4.0, on the other hand, requires a set of basic skills that can be linked with computational thinking. Although many of the concepts involved in both computational thinking and Industry 4.0 are difficult to be achieved by using a traditional learning path, there are alternatives, especially for young learners, to get started and advance with computational thinking, and get accustomed with some of the new skills required by the new industrial revolution. There are numerous means to acquire such capabilities, supported by formal and non-formal frameworks of education, and assumed by different types of organizations. We focus our paper on an educational path from the youngest learners to teenage programmers, in the context of CoderDojo non-formal educational activities.*

Keywords: Blocks-based programming, Computational thinking, Robotics, Scratch, K-9 education.

1. Introduction

Computational thinking was first introduced in 2006 by J. Wing as "a universally applicable attitude and skill set everyone would be eager to learn and use" to be added to the basic set of abilities on top of which every child should build their education (Wing, 2006). Since then, numerous contributions emphasizing the different aspects of computational thinking can be counted. While computation-like thinking is not something new for the youngsters, who have become familiar with some basic skills through various building/board games, elementary mathematics, or even easy science experiments, computational thinking can develop over these skills, among others, towards a set of analytical thinking capabilities. The society is rapidly advancing with Industry 4.0, which in a rather simplistic definition can be viewed as an "application of the IoT, cloud computing, cyber-physical systems (CPS), and cognitive computing into the manufacturing and service environment." In this context, computational thinking is a "vital skill for

https://doi.org/10.58503/icvl-v17y202221

empowering employees to address problems critically and systematically. (Chong & Wong, 2019).

Many approaches to introducing computational thinking among the youngest exist, many of which are built on habits acquired in their early experiences (e.g., brick buildings or the easy-to-follow board games). There are different block programming approaches, which address the curiosity of young programmers, such as Scratch Jr or Lego Education WeDo, as introductory tools (Papadakis et. al., 2016; Pinto-Llorente et. al., 2016) and Scratch, VEXcode, Robot C, or Lego Education (Mindstorms) EV3, for some advanced notions and transition to robotics (Ko, 2013; Roscoe et al., 2014; Zhang & Nouri, 2019). As (Hsu et. al., 2019) mention, computational thinking has marked a turnpoint in different tech-based educational initiatives across the world. The introduction of computational thinking in the different fields of education allows the realization of the ambition "to prepare the younger generations for the opportunities and challenges of the future economy where computing permeates virtually every aspect of society. (Hsu et. al., 2019).

Not for profit initiatives play an important role in attracting the attention on the importance of computational thinking for the young generations and, without any constraints in their implementation means, they can afford innovative approaches towards achieving some of the objectives of computation thinking. Such initiatives include CoderDojo, "a global movement of free, volunteer-led, community-based programming clubs for young people", with a focus on "peer learning, youth mentoring and self-led learning", or the "Științescu Fund", a program to support several community foundations to create and manage local funds "to encourage innovative ideas that can make sciences more attractive for secondary education students and that can be applied with support from the community." The content of our paper is based on the experience gained by implementing activities within local CoderDojo implementations, as well as the experience gained by implementing some of the Științescu community-funded projects, with a specific focus on the learning path for the young attendees through the various blocks-based programming languages, a plethora of tools, with a unique goal: achieving and improving the computational thinking abilities of the young learners.

2. Background Information for Block-Based Programming

2.1. Block-Based Languages

Blocks-based languages offer a simple and intuitive approach to programming, usually not linked with previous programming experiences. With the different categories of block types, the user has the ability to construct applications by simply "snapping" those blocks that match each other, an ability that is based on the experience of brick building toys. Apparently, such a programming language is rather simple, with limited possibilities to build complex programs. Many block-

based programming languages currently exist, since the initial version of Scratch was publicly made available, in 2007, with the ambition to improve the "digital fluency". "Digital fluency" includes "designing, creating, and remixing, not just browsing, chatting, and interacting", (Resnick et al., 2009) and it is broader than the "digital literacy", offering the opportunity for learners to "self-select from a range of tools to achieve outcomes, and navigate collaborative".

The block-languages considered in our study cover different characteristics:

Age: Scratch Jr or Lego WeDO for the youngest programmers, aged 6-8; Scratch or Lego EV3 for young learners, aged 8-12; Scratch with advanced features, Snap! (a Scratch spin-off, an extended reimplementation of the language, offering advanced options to "build your own blocks"), or VEX-based approaches for young and teenage programmers;

Approach: programming by playing (Scratch Jr and Lego WeDo), programming by exploration and experimenting (Scratch, Snap!, EV3 or VEXCode), programming by robot experiments (WeDo, EV3, VEXCode, Robot C, also Scratch/Snap! with appropriate extensions);

Interaction: sprite-based, in Scratch Jr, Scratch or Snap!, robot-oriented in Lego-centered or VEX-centered initiatives;

Expressivity: simple interactions in Scratch Jr or Lego WeDO; programming basics and/or advanced programming concepts in Scratch, Snap!, EV3 or VEX-based/ Robot C.

With such a variety of options, Scratch-based approaches can be used to gain basic skills, complemented by advanced elements introduced by the introduction of the different robot-specific devices and sensors.

2.2. Programming Basics

Despite its rather simple appearance and seemingly rudimentary support for data types, the different block-based programming languages abound with less expected features, thus offering a different perspective to the apprentice programmer compared to the traditional means used for first steps in programming. Visual programming approaches like Scratch focus their attention rather on the application logic, instead of the syntactic aspects of the language (Malan & Leitner, 2007), emphasizing that the focus is on the final outcome, not on the particularities of those building blocks. In such a simplification, "programmatic constructs are represented as puzzle pieces that only fit together if syntactically appropriate", offering the means to intuitively discover the basic structures of programs. In fact, according to (Malan & Leitner, 2007), the transition to a high-level programming language was easier for first-time programmers when they developed prior experiences with a visual-programming environment. The development of many block-based languages is based on the triplet "low-floor/high-ceiling/wide-walls", being thus easy to get started, covering various types of users, while offering the opportunity to create complex projects (Jatzlau & Romeike, 2018; Resnick et. al., 2009).

Some of the core concepts and capabilities for block-based programming languages are briefly presented next. However, more details can be found in (Jatzlau & Romeike, 2018), while (Fagerlund et. al., 2020) offer a clear overview of computational thinking concepts, and their mapping onto Scratch-based environments.

Program flow: the ability to manually control the execution path of a program, with the possibility to pause the application, interact with the sprite or robot/device, observe and correct their behavior. (Zhang & Nouri, 2019).

Sensing: the capability of the controlled objects (sprites or robots/devices) to interact with their environment or with the user/programmer, detect and report various information. The relevant computation skills covered include events or coordination /synchronization (Zhang & Nouri, 2019).

Weak data type: the low-floor paradigm cannot be achieved with the data type constraints. Even in a simplified form, this can offer an introduction to the variables / initialization skills for computational thinking (Zhang & Nouri, 2019).

Delayed execution of code: this an interesting capability offered by blocks-based languages, avoiding the application to run too fast.

Drag and drop: in contrast with text languages, all blocks-based languages offer a drag-n-drop support, the programmer will build the application by selecting blocks that fit with the target one.

Block shapes: there are many characteristics blocks can have, and their shape is one of the essential characteristics.

With the different block characteristics (color, shape, induced semantics, domain of use), it is not a difficult task to gradually introduce proper programming-like attitudes to the youngest learners and, consistently, maintain and develop even more programming abilities on top of these. Moreover, as current approaches included in EV3 Mindstorms or VEXCode are fully compatible with Scratch, there will be an easy transition from the young learners to a more advanced user.

2.3. Advanced Features in Block-Based Languages

The core concepts for block-based programming languages were discussed in (Jatzlau & Romeike, 2018), and include broadcasting, sensing, prototyping, event-driven programming, weak data types, or delayed execution of code. At the same time, some of the advanced concepts considered in (Hermans & Aivaloglou, 2017) are events, coordination, parallelism, together with the rather traditional ones, like conditionals, loops, operators, procedures, etc. We will offer an overview of some of the unexpected concepts offered by the blocks-based languages used in our approaches.

Concurrency: multi-threading is an interesting feature of Scratch, which allows students to get accustomed with concurrent programming concepts by the means of sprite (the main abstractions from the language) control. As it was presented in (Meerbaum-Salant et. al., 2013), the concepts related with concurrency can be considered at two levels: “type 1 concurrency occurs when

several sprites are executing scripts simultaneously, [...] type 2 concurrency occurs when a single sprite executes more than one script simultaneously". Also, there is a rather lightweight multi-threading mechanism, as the language is missing some of the high-level synchronization mechanisms. Instead, the language offers the complementary messaging system, which offers in turn some basic synchronization means. In fact, there's nothing to worry about race conditions, as the Scratch model was developed "by constraining where thread switches can occur. (Maloney et al., 2010).

In (Fatourou et al., 2018) the authors concluded that "learning tasks were built in a structured approach so that pupils incrementally build knowledge on concurrency issues [...] and not missing the fun of game design". Notice that in the case of some robot-based environments, explicit support for threads/tasks may be used, like in the case of VEX-based projects, when the text version of the coding environments is used.

Messaging and event processing are yet another surprising features of the various blocks-based programming environments. The broadcast-receive pattern is a central feature in Scratch inter-sprite communication, with a default asynchronous behavior, where a message is sent to all sprites capable of accepting it. Additionally, there is a synchronized flavor of this pattern, where the broadcaster will have to wait until all receiving sprites end their message-based processing. With the message passing implementation, the languages expose a rather complex concept through the easy-to-use interface for those who are starting to build their computational abilities. "The concept of message passing is quite complex, combining concurrency, synchronization, and the asymmetric roles of the sender and receiver." (Meerbaum-Salant, 2013).

Event processing is another important concept spanning across the different blocks-based environments, either sprite-based or robot-centered. Event processing can be linked with other capabilities exposed by those languages, like concurrency, messaging, user interactions, or sensing, and can be invoked in various ways: one sprite touching another, sprites touching specific colors, a distance or color sensor attached to a robot, and others.

Sensing: despite its intuitive look, sensing is not an easy concept for block-based programming languages. Sensing is used to describe both the capability of sprites to interact with some environments (touching other objects, distance to other sprites, etc.), or the ability of robotic constructs to interact with the real environment by the means of different sensors (such as distance, light, touch, etc.) The sensing capability can offer the necessary support for an autonomous behavior of the different objects (Jatzlau & Romeike, 2018). Sensing also involves event processing capabilities and may require existing concurrency/multi-tasking support. In order to access the sensing support, a hardware configuration may be needed.

3. Learning Alternatives

The emergence and implementation of the first CoderDojo clubs (dojos) represented an outstanding opportunity for a large number of young learners to get in touch with programming by various technologies. It was the place where two global initiatives intersected, as Scratch was one of the enabling technologies for CoderDojo, to provide an exciting experience to young learners. The local CoderDojo network involved an important number of volunteers, and even more young learners who regularly attended weekly workshops. More than half of them had no previous experience in programming, and almost 25% were enrolled in the first grades of primary school when they started the learning activities. The CoderDojo approach raised important challenges for the mentors, since most of them were IT specialists or students without any teaching background. A methodology for delivering the knowledge was needed such that no one is left behind, regardless of their background and, at the same time, to offer a successful approach for the numerous children that are about to begin reading.

Even if Scratch-based delivery of knowledge was the natural choice, given the success of the Scratch educational path as it was implemented by the CoderDojo Foundation, Scratch adoption was not an easy task, as few quality learning materials were available, and the workshop materials needed to be developed in the meantime. This allowed us to develop a core set of activities, and to complement them by the introduction of additional technologies through specific extensions (e.g., Lego WeDo in Scratch projects) or offer new experiences that were enabled by existing knowledge (like the development of EV3 or VEX IQ interactions).

3.1. Activity Organization

Regardless of the experience of the mentees (“wide-walls”), each workshop starts with a story and an example, which can easily be adapted to the preferences of each participant. By using such a beginner-friendly environment (“low-floor”) for learning, users can rapidly visualize some results, and easily understand the effect produced by different commands. First workshops are meant for a smooth introduction into the programming environment, starting with games and stories that can be decomposed into simple steps that can be easily transformed into some sequence of action (the algorithm). Such an approach offers the means for an easy introduction to the identification of the flow of the action, and the analysis of the problem, eventually to support problem decomposition once the learner can better understand the process. As each child will further develop the project at their own pace, in the following workshops they will have the opportunity to reach the level of development appropriate for their level of knowledge and, finally, to present their achievements to the others. Periodically, the organization of major events is considered (local events, named MegaDojos, or the global Coolest Project events), bringing together children from different geographical areas, an opportunity

suitable both for the presentation of projects made in previous workshops, and for introductory sessions for newcomers.

First time learners: Many of the first learners are K-5 students, some of them with minimal reading abilities. However, this is not a real difficulty in our approach, as we have the groups of color-coded blocks: blue for Motion, magenta for Sounds, yellow for Events, orange for Controls or green for Operators. Moreover, for these first-time learners we are using a set of Scratch cards, each of them presenting a simple command that will be executed as mentioned. The students are asked to change the different parameters, such that they understand the effects of essential command blocks for this stage: Motion, Looks, Sound and Events. Also, a continuous mentor-mentee relation is a must in order to guarantee a successful learning path. While each child will be happy to see that everything imagined quickly turns into reality, within games, animations or stories, together we are able to make the first steps towards computational thinking. The first-time learner will acquire some CT basic skills:

a) run their first applications, by making a connection between the green-flag and the corresponding yellow-hat block;
b) define a simple algorithm, by stacking a series of blocks, usually blue or purple, at their choice;
c) define a second sprite, and make it active by using the yellow green-flag block;
d) understand and change sprite properties, in a more sophisticated use;
e) for the advanced uses, introduce some decisions, by the corresponding orange if block, and have a basic understanding of some conditions (with the green boolean hexagonal blocks);
f) and, for the most advanced use at this level, play with some wait actions, eventually linked with some purple Looks blocks.

As most of these basic skills do not require a high level of knowledge, the entire learning process can be easily adapted to alternative languages: Scratch JR or Lego WeDo. Worth to mention that, while Lego WeDo can offer an appealing alternative to introductory workshops, it can also be the subject of some complex approaches, if integrated in a Scratch project. Depending on their age and level of understanding, the new learners stage can last up to 4 months. Most of them will be capable of changing their roles, from simple consumers to creators. It is a moment when they want to create their own games, stories, or art projects. They are able to advance to the next level, as there's a clear curiosity to check the use of multiple sprites/stages and increase the complexity of their projects (high-ceiling).

Young programmers: Once the first-time learner stage is over, many of the participating children will continue the block-based learning path. We consider that block-based languages offer enough opportunities, even when faced with new directions of development, as blocks-based environments can be used for Lego EV3 Mindstorms or VEX IQ projects, Raspberry Pi, or even offer some preliminary steps towards mobile application development, via AppInventor. Even

if the activities are still under the umbrella of the weekly workshops, there is more freedom in the learning path for the young programmers. During this stage their curiosity is encouraged, with less interaction with adult mentors, but more interaction with other young programmers and young mentors (usually from the team of young programmers). The young programmers are encouraged to discover the language capabilities and, consequently, build new skills, through thematic projects. New approaches were made available (Lego WeDo/EV3 in Scratch installs, robots-based approaches, like VEX IQ, etc.) in order to offer even more possibilities to develop the new skills, allowing the young developers to create complex and interactive projects, exposing a higher level of CT abilities.

The young programmer will have the opportunity to acquire additional computation thinking skills, based on:

a) *creativity*, via pen tools, by replicating turtle graphics programming languages, adding graphic effects, animations and even arts to the projects;
b) *collaboration and interaction*, as they are encouraged to periodically present their projects, and participate in larger scale events;
c) *application flow*, with improved text inputs, written or spoken, exposing the story and improving the general interaction level of the project;
d) use some *advanced features* for mathematics (e.g., random numbers), variables and lists (storing, using and showing relevant information for their projects, e.g., time, scores, lives, items, essential in the case of a game);
e) *problem analysis and decomposition*, by defining their own blocks (procedures, for text-based languages), defining different scenes (stages) and/or characters (sprites);
f) *message passing and synchronization*, by defining messages and message handlers, as well as advanced interactions with the script;
g) *application control*, with the capability to use different loop blocks and fully exploit the if-then-else blocks.

Young programmers usually spend up to 6 months developing the full range of skills. At this stage their relationship with mentors is still very important, but only for introducing new notions and for the identification of structural or logical errors in application. Even if learning cards can be used during the first weeks, this type of information passing is rather replaced by an increased collaboration between attendees, especially when robots come into the scene.

Young developers: A significant number of participants will continue the activities in the workshops, either to further develop the applications made previously or to provide support to the freshmen. After their introductory year with Scratch-like approaches, they are either moving to different technologies, or continue to improve their knowledge with block-based languages and start their robotic adventure. Either way, they are going to improve their CT abilities, and develop really complex projects, usually as a team collaboration. The relationship

between the young developers and mentors is more specialized now and directed to the new categories of devices (robotic brain, motors, sensors, etc.) and new categories of requirements for project development. Moreover, the young developers can get in touch with some advanced topics, e.g., related to Neural Networks, Artificial Intelligence and Machine Learning.

Their computational thinking abilities will reach maturity, for the age group they belong to, and most of their projects will exhibit those advanced skills:

a) capability of using the cloning, and control the behavior of cloned sprite;
b) possibility to start multiple stacks (scripts) as a response to various events;
c) fully control the messaging support offered by the language, and influence the behavior of other objects (sprites, backdrops) by messages;
d) understand and use the full range of control blocks;
e) use the full range of interactive blocks (such as Looks, Sound, Sensing);
f) use automation and debugging for applications, have the capability to identify simple structural or logical errors in scripts;
g) load external libraries of blocks for custom executions, or use other blocks-based languages (like WeDo/EV3 support in Scratch, the EV3 Mindstorms or VEX IQ environments);
h) investigate alternative approaches, such as the VEXCode or Robot C text-based editor, where fine tuning of blocks-based projects will be possible.

It is difficult to have an estimate of the time it takes for young developers to understand most advanced concepts of blocks-based language, given the fact that some of them are difficult for university students to understand. However, at this stage a good part of the efforts of young developers are oriented towards the development of competitive projects, which will be presented in local events (MegaDojo) or global (Coolest Projects).

3.2. Assessing Computational Thinking Abilities

Given the non-formal nature of educational activities in the context of CoderDojo, they will not include activities to assess the progress made by participants, the development of each following its own pace, without constraints (the “low-floor” paradigm). However, it is necessary to understand and identify when a certain level of maturity is reached for the language used. And assessing progress can be important to reach the “high-ceiling”: for preparing complex projects, participating and recognition of their progress at the local or global events as mentioned before. Several tools exist for assessing the computation thinking level achieved by a young learner, based on the content of his/her Scratch projects. The evaluation of a set of Scratch-based projects presented at one of the nation-wide dedicated events revealed the most difficult to achieve CT capabilities. The following information was extracted:

a) Most of the projects were of *Master* quality (a total of at least 17 points from the 7 categories): *Master* - 56.25 %, *Developer* - 31.25%, *Beginner* - 12.5%;
b) No project was able to reach the *Master* level for the *User interaction* chapter (usually by specific event/message handlers, little or no use of video/sound blocks);
c) There were difficulties in getting high score in the *Data representation* and *Abstraction* chapters (suggesting difficulties in understanding lists operations, or advanced sprite usage);
d) Most of the projects were able to get high scores at *Synchronization* and *Parallelism* (good command of events and messages handlers, as suggested by *User interaction*, including for sprites or backdrops; the ability to initiate several stacks or scripts at specific events).

4. Conclusion

After several years of activities, the educational approaches developed on top of the CoderDojo approach offer a set of necessary and complimentary activities for introducing programming and computational thinking abilities to the youngest learners. Additionally, a larger number of girl participants were counted during the implementation of these activities, as creative, collaborative coding activities and competitiveness were able to increase their motivation. In our work we presented the learning path adopted in local CoderDojo activities, in order to improve the skill level of young learners, from simple technology users to advanced content creators, a learning path which is failure-free, without any evaluation pressure. Despite its apparent simplicity, the approach used allows the customization of the learning path and the diversification of the tools used in the process of acquiring new knowledge. The different levels of activities also define the different levels of interaction with mentors or other young colleagues, offering as a side effect a certain level of independence in the development of future projects. One can estimate how each of the three levels was achieved by aligning projects to the different skill levels that are associated with computational thinking, thus providing indirect feedback to the implemented learning cycle!

References

Chong, B. & Wong, R. (2019). Transforming the Quality of Workforce in the Textile and Apparel Industry Through Computational Thinking Education. *Computational Thinking Education*. Springer Singapore, (pp. 261–275).

Fagerlund, J., Häkkinen, P., Vesisenaho, M. & Viiri, J. (2020). Computational thinking in programming with Scratch in primary schools: A systematic review. *Computer Applications in Engineering Education* 29, (1) (pp. 12–28).

Fatourou, E., Zygouris, N., Loukopoulos, T. & Stamoulis, G. (2018). Teaching Concurrent Programming Concepts Using Scratch in Primary School: Methodology and Evaluation. *International Journal of Engineering Pedagogy* 8, (4), (p. 89).

Hermans, F. & Aivaloglou, E. (2017). Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering Education and Training Track*. (pp. 13-22).

Hsu, Y., Roote Irie, N & Ching, Y. (2019). Computational Thinking Educational Policy Initiatives (CTEPI) Across the Globe. In *TechTrends* 63(3), (pp. 260-270)

Jatzlau, S. & Romeike, R. (2018). How High is the Ceiling? Applying Core Concepts of Block-based Languages to Extend Programming Environments. In *Constructionism, Computational Thinking and Educational Innovation*. (pp. 286-294).

Ko, P. (2013). A longitudinal study of the effects of a high school robotics and computational thinking class on academic achievement (WIP). In *2013 IEEE Frontiers in Education Conference*, (pp. 181-183).

Malan, D. & Leitner, H. (2007). Scratch for budding computer scientists. In *ACM SIGCSE Bulletin*, 39 (1). (pp. 223-227).

Maloney, J., Resnick, M, Rusk, N., Silverman, B. & Eastmond, E. (2010). The Scratch Programming Language and Environment. In *ACM Transactions on Computing Education* 10(4), (pp. 1-15).

Meerbaum-Salant, O., Armoni, M. & Ben-Ari, M. (2013). Learning computer science concepts with Scratch. In *Computer Science Education* 23(3), (pp. 239-264).

Papadakis, S., Kalogiannakis, M. & Zaranis, N. (2016). Developing fundamental programming concepts and computational thinking with Scratch Jr in preschool education: a case study. In *International Journal of Mobile Learning and Organisation*, 10(3), (pp.187-202).

Pinto-Llorente, A., Casillas Martín, S., Cabezas González, M. & García-Peñalvo, F. (2016). Developing computational thinking via the visual programming tool. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, (pp. 45-50).

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, J. (2009). Scratch. *Commun. ACM*, 52(11), (pp. 60-67).

Roscoe, J., Fearn, S. & Posey, E. (2014). Teaching Computational Thinking by Playing Games and Building Robots. In *2014 International Conference on Interactive Technologies and Games*, (pp. 9-12).

Wing, J. (2006). Computational thinking. In *Commun. ACM,* 49(3), (pp. 33-35).

Zhang, C. & Nouri, J. (2019). A systematic review of learning computational thinking through Scratch in K-9. *Computers & Education*, 141.