Proceedings of the 10th International Conference on Applied Informatics
Eger, Hungary, January 30–February 1, 2017. pp. 41–52
doi: 10.14794/ICAI.10.2017.41

A comparison of the instruction-oriented and the problem-type-oriented teaching methods through the usage of Scratch programming language

Péter Bernát

ELTE Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary
bernatp@inf.elte.hu

Abstract

For teaching programming, two widespread methods in public education are the instruction-oriented method and the problem-type-oriented method. Both of the methods are based on solving problems; however, the first one chooses the problems in order to present the various instructions of the programming language, while the second takes the problem types specific to a given programming language as its basis and chooses the problems, and with that, the instructions, accordingly.

In my article I will introduce the two methods in theory and in practice. I have chosen Scratch as the programming language, that I will provide a short description of. The two methods will be evaluated also from a didactic point of view. My dual goal with this article is to uncover the dangers of using the instruction-oriented method, while highlighting the potentials in applying the problem-type-oriented method.

Keywords: teaching programming, programming methodology, problem-type-oriented method, instruction-oriented method, Scratch programming language

MSC: 97Q60

1. Introduction

In public education the primary goal of teaching programming is to develop algorithmic problem-solving thinking by exposing students to programming problems. A further goal is to familiarize students with programming methods and notions necessary for solving such problems. As their tool, teachers use a specific pro-

gramming language, which will be introduced only to the extent set by the above goals.

For teaching programming, two widespread methods in public education are the instruction-oriented method and the problem-type-oriented method. Both of the methods are based on solving problems; however, the first one chooses the problems in order to present the various instructions of the programming language, while the second takes the problem types specific to a given programming language as its basis and chooses the problems, and with that, the instructions, accordingly [1].

In my article I will introduce the two methods in theory and in practice, by presenting, through one example, how each of the methods would approach teaching a specific set of instructions within the same programming language. I have chosen Scratch as the programming language, because it is one of the most widely used languages for teaching beginner programmers. In a separate section, I will provide a short description of the program as well.

The two methods will be evaluated also from a didactic point of view. Even though both methods are admittedly popular in public education, only the problem-type-oriented method can be truly recommended [1]. Therefore, my dual goal with this article is to uncover the dangers of using the instruction-oriented method, while highlighting the potentials of applying the problem-type-oriented method.

2. Scratch programming language and the selected set of instructions

With Scratch we can create multimedia programs applying the object-oriented and the event-driven paradigms. The objects are sprites which can be moved on the screen and whose looks can be changed. Their most important properties are their position, their direction, their size, their visibility, and their current costume. They are able to communicate with each other by sending messages. Traditional control structures, such as sequences, count-controlled loops, infinite loops, condition-controlled loops, and selections, can also be used. Parametered procedures can be created; primitive variables and lists can be introduced. A special object in Scratch, next to sprites, is the stage, where sprites are located. In this case, it is not the looks but the background which can be modified.

My examples will deal with teaching the instructions of event handling and sprite movement, the instructions of changing looks, one instruction of count-controlled loops and infinite loops each, and the instructions of sending messages.

3. Introduction of the instruction-oriented method to teach programming

The aim of the instruction-oriented method is to teach programming through instructions. In order to do so, it groups instructions and presents each group through

problems.

Typically, the themes of the method consist of listing instruction groups. To teach the above mentioned Scratch instructions, for example, this method would set up the following syllabus:

1. the instructions of event handling and movement;
2. the instructions of changing looks;
3. the instructions of count-controlled loops and infinite loops;
4. the instructions of sending messages.

I will illustrate each section of the syllabus with a problem that I took from books or online materials which are pronouncedly based on this method and aim to introduce Scratch to young learners.

3.1. The instructions of event handling and movement

The first couple of problems can be designed to teach the instructions of event handling and sprite movement. The book *Scratch Programming for Teens* [2], for example, introduces the instructions that turn and move sprites forward through a program which makes the sprite appear for 1 second at each corner of the screen, once the green flag was pressed to start (figure 1).

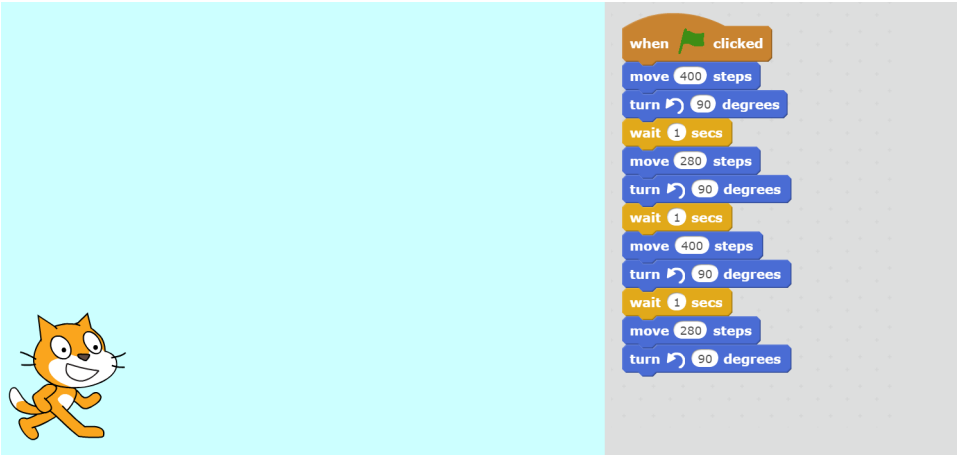

Figure 1: The cat which appears at the four corners of the screen and the program behind it [2]

3.2. The instructions of changing looks

The next problems aim at the instructions that change the looks of the sprites. The notion of costume and the instruction to set the costume can be introduced for instance with a program (figure 2) in which a flying cat is controlled by the

arrow keys, and, depending on the direction of the turn, the sprite will switch to one of its two potential costumes (of looking left or looking right) [3].

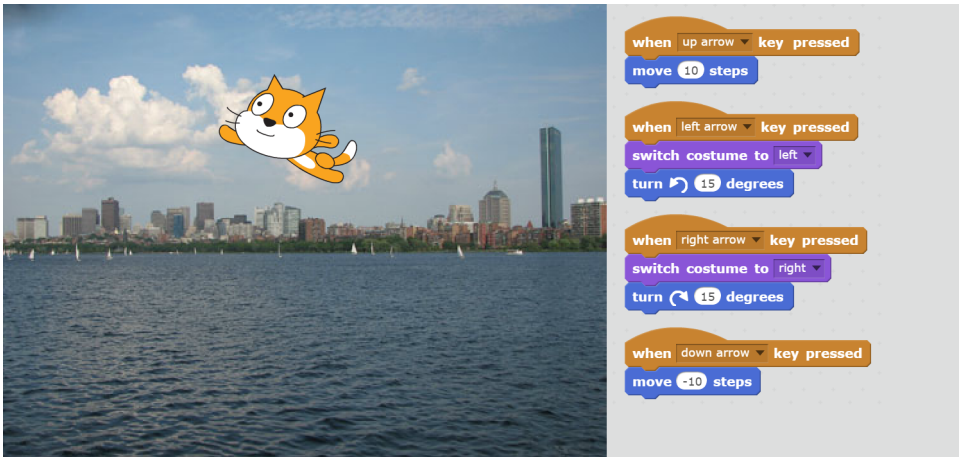

Figure 2: The cat which moves and switches costumes when the arrow keys are pressed, and the program behind it [3]

3.3. The instructions of count-controlled and infinite loops

Some further problems can be used to explain the instructions of count-controlled and infinite loops. For instance, the author of *How to Code in 10 Easy Lessons* [4] decided to introduce the instructions of count-controlled loops with some programs that draw squares. The easiest of these is shown on figure 3, where the sprite is drawing a fixed sized square.

3.4. The instructions of sending messages

With the help of the last problems, messaging can be presented. In the next program [5], the sprite will always move in the direction of the girl that we click on. The chosen girl, Ann or Britney, will send the message "look at Ann" or "look at Britney", respectively, to the sprite, which will make him look and move toward her (figure 4).

4. Evaluation of the instruction-oriented method to teach programming

Even though the instruction-oriented method introduces each instruction of the programming language, its application can lead to a number of harmful consequences. On the one hand, it dedicates approximately the same amount of at-

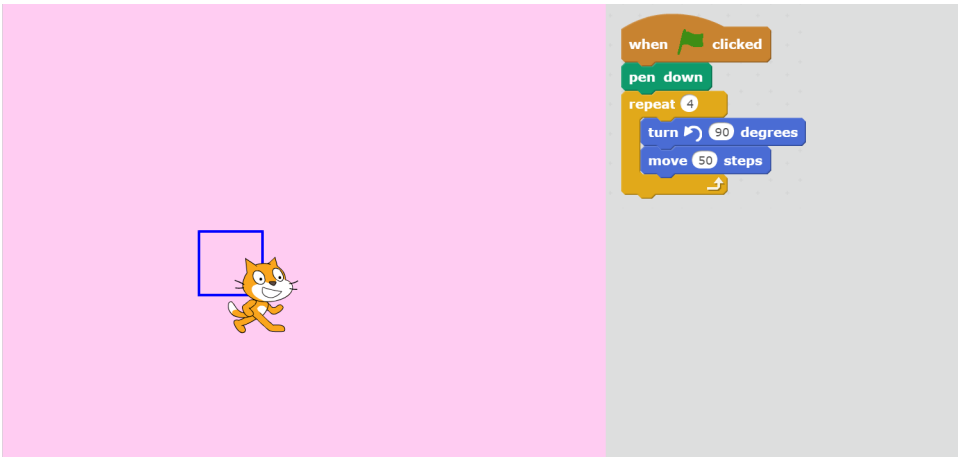

Figure 3: The cat which draws a square and the program behind it [4]

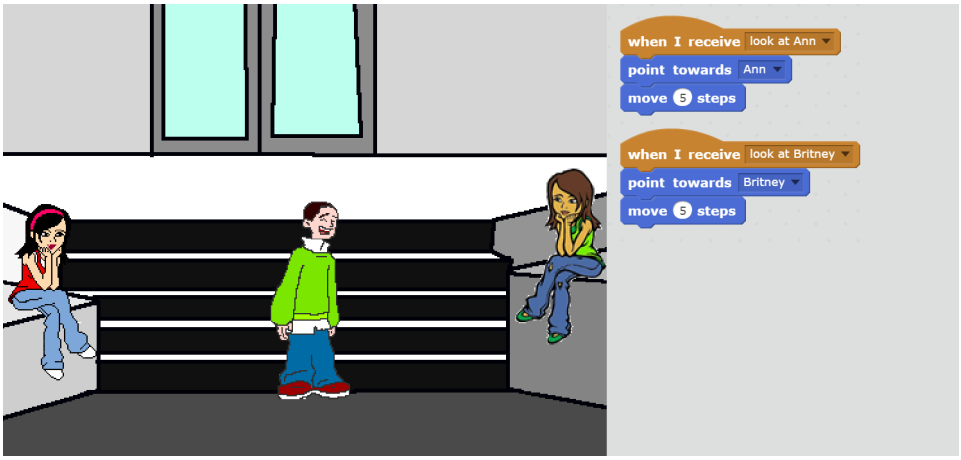

Figure 4: The boy moving towards the girls and the boy's scripts [5]

tention and time to each instruction, while their significance and frequency can vary a lot in reality. On the other hand, it aims to demonstrate the instructions in problems which are as simple as possible, which means that the problems are not getting more and more complex in a gradual fashion, nor is there space for the repetition of the previously learned instructions. Both would be didactically important, however.

In programming practice we identify the necessary instructions based on what a problem requires; nevertheless, the instruction-oriented method works the other way around. It makes up problems for the given instructions. As a consequence, it

is not uncommon that the problems are created only for the purpose of presenting a given instruction, without connecting the problems to a category of an existing genre (like animation or game in the case of Scratch); thus, the problem may feel unreal. Such problems (like exemplified by the program in figure 1) are not too motivational. Similarly, it might happen that the problem makes use of, thus, introduces the instruction in a non-typical way, which is misleading (in Scratch count-controlled loops are more often used to move a sprite or replay music than to draw a square or other symmetric shapes).

Nevertheless, the most severe weakness of the method is that it fails to deal with problem-solving methods which go beyond the scope of instructions. It is so because the instruction-oriented method is grounded on the misconception that to be able to solve more complex problems, all it takes is simply to be aware of the instructions. The truth is, without equipping students with problem-solving methods, we can provide little support for their independent creative work.

5. Introduction of the problem-type-oriented method to teach programming

The problem-type-oriented method approaches problem solving in a fundamentally different way. First of all, it is prepared to accept that even the simplest problems may need to be split up into sub-problems, which in the case of more complex problems is practically inevitable. This operation needs to be repeated on several levels until we get to sub-problems that can be solved with the instructions directly [6]. To solve recurring sub-problems, the problem-type-oriented method offers design patterns which are the reusable, general solutions to these sub-problems.

The goal of the problem-type-oriented method is to teach design patterns and instructions necessary for solving problems that belong to a certain problem group, and within that to a certain problem type. The different problem types, and within those the different problems, are ordered so they can gradually introduce students to design patterns and instructions, while systematically repeating and refreshing them [1].

While teaching Scratch, or generally beginner programming, animation and game development are two popular problem groups. The basic instructions, defined at the beginning of the article, can be introduced through animation. In the following, I will explain my own conception on some of the potential problem types within animation, along with the design patterns and instructions they require. The problems I will present are modified versions of programs shared by users of the official website of Scratch.

Typically, the syllabus that follows the problem-type-oriented method is a list of problem types, just like below:

1. simple interactive animations;
2. one-scene animations;
3. multiple-scene animations.

5.1. Simple interactive animations

Into this problem type I have categorized animations in which a click of the mouse or the pressing of a key triggers that the screen or parts of the screen change appearance or a sound is played.

Figure 5 portrays such an animation (an emoji maker), which allows for changing the background, the face, the eyes, the mouth or the hat by pressing a key [7]. To make one of many funny combinations possible, the programmer needs only two things per each image excerpt (which are separate sprites): to assign a key, which if pressed, activates the instruction and to add the instruction that switches the costume of the sprite. In the present example, sound effect was also added.

Figure 5: Emoji maker and the script of the eyes (for example) [7]

Versions where sprites are constantly switching costumes and thus entering into animation are more spectacular. Take the Christmas card animation in figure 6 for example: for one, the flames in the fireplace are moving constantly; for two, with a click we can activate the bell, which will start ringing (making visual and sound effects), the television, which will play a short cartoon; or the Christmas tree, which will switch on its lights [8]. To translate this into programming terms, the bell has two costumes, one illustrates it tilting left, while the other tilting right. To create the continuous animation of the ringing bell, we need to repeat the instruction of costume-switching in a count-controlled loop with some pauses. Since it is frequent that animations will require *the constant costume switching* of sprites, it is worth referring to this solution as a design pattern that we will return to use from time to time.

The problem type of simple interactive animations can be utilized to introduce the instructions of event handling and costume switching, while with *the design pattern of constant costume switching* we can present count-controlled and infinite loops.

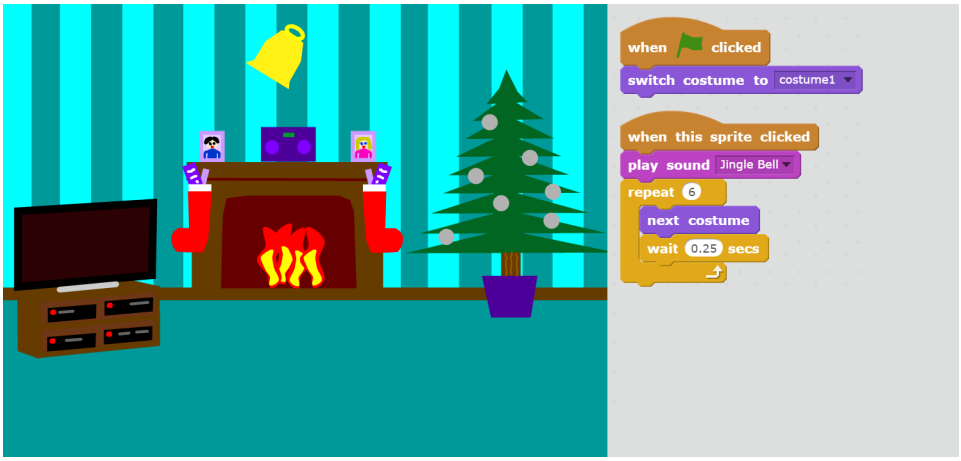

Figure 6: Interactive Christmas card and the scripts of the bell [8]

5.2. One-scene animations

The peculiarity of one-scene animations is that they bring to life short stories with sprites that can already change their positions. In the example below I am presenting another postcard, this time for Valentine's Day [9]. The story goes like this: the princess runs to her suitor, they exchange some words and then a kiss. Simultaneously to the act of affection, a board, decorated with flashing lights, appears with the good wishes (figure 7).

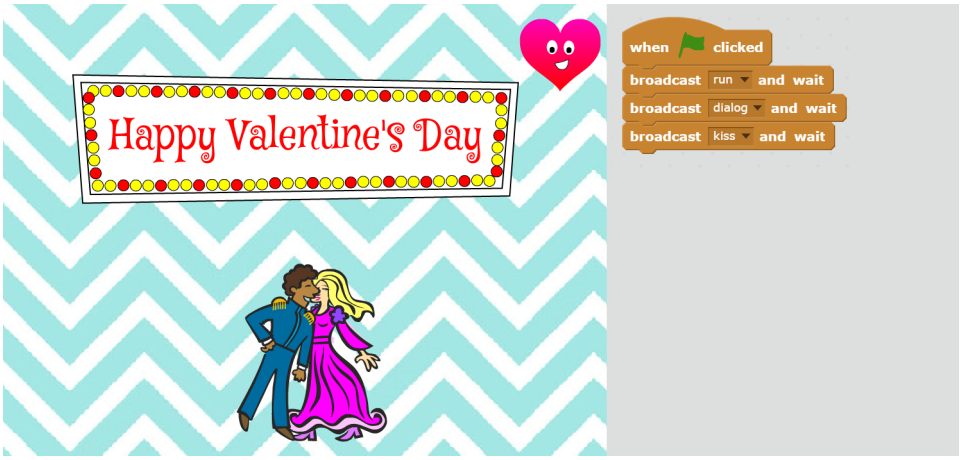

Figure 7: Valentine's Day card and the script of the stage [9]

This problem contains three typical sub-problems of the problem type; it will be best to refer to these solutions as design patterns.

In the process of animation design, we often need to synchronize the timing of the sprites' activities. Staying with the above example, the characters on the postcard need to start their conversation exactly when she arrives. Similarly, the greetings need to appear exactly when he kisses her. We can synchronize the activities by adding meticulously determined pauses but it is more transparent and easier to modify if we split the animation into consecutive steps. In this case, we define the activities of the sprites in each step, and we determine the order of the steps through the messages sent by the stage. As we can see above, the stage sends the messages belonging to the three steps one after another. The general design pattern behind this solution can be referred to as *the design pattern of splitting into steps.*

A further frequent sub-problem of animations is to *move a sprite between two points.* This can be done in several different ways in Scratch, which are worth comparing. In our example, the feet of the running princess are constantly moving, which entails that next to the movement of the sprite, constant costume switching also needs to be taken care of (figure 8).

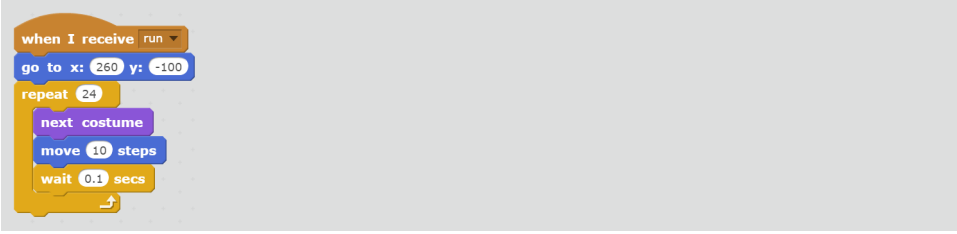

Figure 8: Moving the princess between two points and switching costumes constantly

Finally, *dialogue* is another recurring element of animations. In this case, we need to synchronize the activities, more precisely, the utterances, of the sprites, which seems to be easier by applying pauses, not by splitting into steps like before. That is, while one character is talking, the other is politely waiting (figure 9).

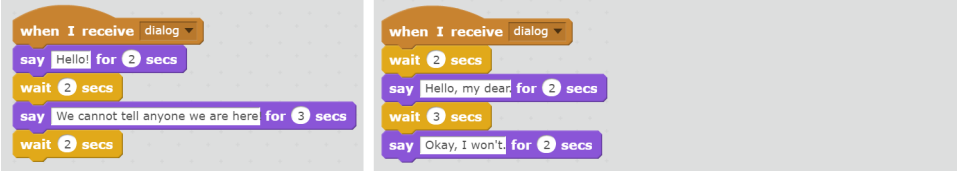

Figure 9: The dialogue between the princess and her suitor

In summary, the problem type of one-scene animations can help introduce *the design patterns of dialogue, moving between two points, and splitting into steps,* and of course through these the instructions of movement and sending messages. Additionally, we need some of the design patterns and instructions introduced

with the other problem type (the constant flashing of the lights on the sides of the Valentine's board can be realized with the help of *the design pattern of constant costume switching*).

Naturally, it is best to introduce the three design patterns of the problem type gradually: first, we can start with a problem where sprites are motionless but they can switch costumes while talking; second, we can move on to a scene where the sprites not only talk but move as well; third, we can arrive to a multi-step animation similar to what was described before.

5.3. Multiple-scene animations

Animations that narrate longer stories generally take place on various locations. In this problem type what is new is *splitting into scenes.* In Scratch, sprites perceive a specific background setting as an event; therefore, scenes can be timed simply by setting the appropriate background. The specific scenes, then, can be split into steps applying the design method introduced already with the previous problem type. The animation illustrated in figure 10 takes place in space and on the surface of a planet (own program). Consequently, the stage sets space as the background first, but then when the scene ends, the surface of the planet is set. Since the specific scenes are complex enough, their execution is split into further steps, as the stage messages show.

Figure 10: A science fiction story and the scripts of the stage (own program)

Needless to say, the problem type of multiple-scene animations will require us to apply the design patterns and instructions, introduced before, for designing some of the scenes.

5.4. Overview of the three problem types

The next table summarizes the instructions and design patterns necessary for solving the previously examined problem types. I have attempted to put the problem types in such an order that they can introduce instructions and design patterns gradually, while also refreshing them.

	simple interactive animations	one-scene animations	multiple-scene animations
instructions of event handling	✓	✓	✓
instructions of changing looks	✓	✓	✓
design pattern of constant costume switching	✓	✓	✓
instructions of count-controlled and infinite loops	✓	✓	✓
design pattern of dialogue		✓	✓
design pattern of moving between two points		✓	✓
instructions of movement		✓	✓
design pattern of splitting into steps		✓	✓
instructions of sending messages		✓	✓
design pattern of splitting into scenes			✓
instructions of background switching			✓

Table 1: The instructions and design patterns (in italics) belonging to the three problem types

6. Evaluation of the problem-type-oriented method to teach programming

The problem-type-oriented method approaches gradually more complex problems by splitting them into parts and offering design patterns to solve them. This is why the present method is capable of developing problem-solving thinking, contrary to the first method introduced.

Choosing the problems from within problem types guarantees that they make sense, for one, and that they motivate students, for two. Next to the instructions, this method introduces design patterns specific to the given problem type as well, thus, enabling students to think independently and creatively within a problem type. Last but not least, if the problems are ably selected, we can show students the real potentials of a given programing language.

An important feature of the method is repetition. It uses the instructions

and the design patterns over and over again but in slightly different ways, which supports the gradual process of understanding the terms and methods all the more deeply, guarantees constant practice, and convinces students that what they learned is useful.

When teaching programming by this method, we guarantee that the specific instructions will come up just like in programming practice (meaning both the frequency and the types of sub-problems). That is, if an instruction is frequent in programming practice, so it will appear in the problems we teach too. Certain instructions may not be covered at all, but that is only because those instructions are not that regular in programming practice either.

7. Conclusions

In my article I tried to demonstrate and assess the instruction-oriented and the problem-type-oriented methods through an example. I have concluded that the latter is not only without the shortcomings of the former, but it even has important didactic advantages the other lacks.

It is fair to add, though, that the teachers or the authors of the programming book need to have a great deal of extra skills and creativity, in addition to the knowledge of the instructions, in the case of the problem-type-oriented method. They need to be aware which are the problem types that can be solved smoothly in the given programming language, along with the design patterns they require. Most probably this is why the instruction-oriented method still tends to be applied alongside the problem-type-oriented method in most contexts.

References

[1] Szlávi, P., Zsakó, L., Methods of teaching programming, *Teaching Mathematics and Computer Science* 1/2 (2003), 247–257.

[2] Ford, Jr. J. L. Scratch Programming for Teens, *Canada: Cengage Learning PTR*, 2008.

[3] Code Club Projects, `https://codeclubprojects.org/en-GB/scratch/` [cited 2017 May 31]

[4] McManus, S. Super Skills: How to Code in 10 Easy Lessons, *London: QED Publishing*, 2015.

[5] Scratch Magyarország Portál, `http://scratch.mit.edu/` [cited 2017 May 31]

[6] Kamthane, A. Programming and Data Structures, *London: Pearson Education*, 2009.

[7] Emoji Maker!, `https://scratch.mit.edu/projects/99608650/` [cited 2017 May 31]

[8] Santa Clause Christmas Card, `https://scratch.mit.edu/projects/14888671/` [cited 2017 May 31]

[9] Dialoge love is in the air, `https://scratch.mit.edu/projects/96729686/` [cited 2017 May 31]