

# Empirical studies of students behaviour using Scottie-Go block tools to develop problem-solving experience

Paweł Gepner<sup>1</sup>[0000-0003-0004-1729], Martyna  
Wybraniak-Kujawa<sup>1</sup>[0000-0001-6335-9481], Jerzy Krawiec<sup>1</sup>[0000-0001-5535-1850],  
Andrzej Kamiński<sup>1</sup>[0000-0002-8759-9786], and Kamil  
Halbiniak<sup>2</sup>[0000-0001-9116-8981]

<sup>1</sup> Warsaw University of Technology, Warsaw, Poland  
{pawel.gepner,martyna.kujawa,jerzy.krawiec,andrzej.kaminski}@pw.edu.pl  
<sup>2</sup> Czestochowa University of Technology, Czestochowa, Poland  
khalbiniak@icis.pcz.pl

**Abstract.** This research is introducing and evaluating the new method supporting grow of programming skills, computational thinking and development of problem-solving approach that evolutionarily introduce programming good practices and paradigms through a block-based programming. The proposed approach utilizes problem-based Scottie-Go game followed by Scratch programming environment implanted to Python programming course to improve the learners' programming skills and keeps motivation for further discovery of computational problem-solving activity. To date, practically little work has been devoted to examining the relationship between beginner development environments and the development practices they stimulate in their users. This article tries to shed light on this aspect of learning programming by carefully examining the behaviour of novice programmers using the innovative block-based programming learning method.

**Keywords:** Computer programming · Problem solving · Computational thinking · Empirical studies · User behaviour · Innovative interaction techniques

## 1 Introduction

In the 21st century the problem solving, and computer programming are perceived as the most critical competencies for students to be used in the solving real-world problems and one of the most demanding area of teaching ranked by educators [2]. It is already understood that programming and coding is not only restricted to computer scientist or professional coders but is fundamental expertise for resolving most sophisticated problems of the era and constructing complicated systems [9]. There are studies describing the relationship of computer science in education that demonstrate the importance of teaching computer thinking from an early age as a fundamental skills. Nevertheless, the same class

of problems are important limitations also outside the educational sector and might be interesting for software development industries.

Developing an effective programming learning concept and curriculum in the right way is a serious and difficult challenge, but many analyses have been done to successfully incorporate programming and coding into the curriculum. Majority of the researches discovered that at the final stages of initial programming courses, most students had problems in decomposition and implementation of algorithms using coding techniques to solve actual problems [9].

Numerous studies have been concentrating on new innovative, alternative approaches and discussing the results of implementing into the teaching program visual programming environment, such as Scratch, Etoys, Alice, RoboMind [15], LighBot, Play LOGO 3D, or Karel, Jeroo, B# [5]. Those studies found positive results connected to visual programming environments and discussed the improvement for novice programmer's engagement in programming with help them grow programming skills [3]. No doubt, that use of visual programming ecosystem has proved its precise benefits to support learning for novice's programmer's [3], but how this is stimulating computational problems solving ability needs to be deeper discovered and better understood. This paper tries to address the gap of this problem and covers two fundamental issues. As the first is to propose a new learning method dedicated to novice programmers, aimed to develop computational thinking and developing a problem-solving approach using a block-based development environment. The second goal is to review the efficiency of solving computational problems by validating three teaching scenarios differing in the sequence of methods and tools used and the visual programming environments in computational problem-solving activities by novice programmers. This paper pursues to address by answering the following research questions:

- Does the new proposed method stimulates the behaviour of the problem-solving and coding thinking skills of novice programmers to solve computational problems?
- Do the novice programmers' computational solutions to the problems are different and they are related, to the way they were learned programming?
- What impact does new method on the novice programmers' perceived difficulty in programming?

It seems obvious that the outcomes of this research can become useful to organizations and individuals responsible for curriculum development for novice programmers to promote programming strategy and computational problem-solving skills.

## 2 Related Work

The aim of this section of the work is to present the fundamental ideas, highlighting the presented method and to present the need for the planned work on improving the problem-solving skills and computer thinking of novice programmers.

## 2.1 Problem solving for programming learning

Computational problem-solving contains the development of computer codes and is considered as the fundamental expertise of computer science education but path to the solution mainly implicating extensive problem-solving skills, instead of basic technical coding activities [15]. According to literature study, problems in the programming process were mainly in the problem-solving phase (such as analysis and design) and the implementation phase and was mainly driven by the lack of problem-solving skills complemented less by lack of semantic knowledge, and weakness in testing the design.

This observation encouraged many researched to propose new schema of teaching; problem-solving first and then programming, they also emphasized the importance of teaching problem-solving independently from programming language. In this approach, students can concentrate on problem solving and testing strategies totally independently from syntax limitations. When the solution is created then particular programming language can be used to translate the solution into code and to test the final program. This idea can be implemented and described by following 8 stages process:

1. Presentation and definition of the problem;
2. Transition from an initial understanding of the problem to a detailed formulation;
3. Building a solution plan using the decomposition of goals into sub-goals, as well as tasks to implement each sub-goal;
4. Developing a solution design from a high-level design to a detailed design by transforming sub-goals into corresponding algorithms;
5. Identify the best type of language to implement the algorithm;
6. Learning the syntax of a programming language;
7. Execution of a detailed design in code;
8. Code testing.

Based on the method described above, there is evidence that problem-solving based learning improves student achievement from 50% to 68% [14]. The study also showed that students in the software development process are enthusiastic about the new learning environment, this significantly affects their enthusiasm and removes blockades related to learning and the use of syntax in programming.

In parallel to the research described above, many scientists also noted that when learning computer-based problem-solving through games, students are more experience pleasure in the learning process than with traditional lectures [15]. Similarly, game development involves both design and programming, and can support the learning of IT concepts by students [8]. It turned out that using educational games, students enjoy e-learning and achieve better learning results as in traditional courses [15]. The use of narrative tools, visual programming and flow modelling tools allowed students to concentrate on algorithms rather than coding [7], as well focus more on the abstract layer of design and problem-solving skills [16]. Research shows that Introduction to Programming courses using object visualization technics and attractive 3D animation environment radically improve student performance [6].

## 2.2 Computational thinking in educational context

Jeanette Wing introduced the concept of computational thinking as the first person, she published the definition first time in 2006: “Computational thinking involves solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science” [13]. Another definition proposed by the “International Society for Technology in Education and the Computer Science Teachers Association” describe computational thinking as: the way how the people use computers to analyse the data, representing data through abstractions and automating solutions through algorithmic thinking. Of the many different proposed definitions, one of the most cited is that of the Royal Society, proposed in 2012 “Computational thinking is the process of recognizing aspects of computation in the world that surrounds us, and applying tools and techniques from Computer Science to understand and reason about both natural and artificial systems and processes”.

According to the observation made by [12], there are several basic computational concepts that programmers have to use:

- Sequence: To build a program, think about the sequence of steps.
- Iteration (looping): Repeat are used for repetition (iterating a series of instructions).
- Conditional statements: if-elif-else checks conditions.
- Threads (parallel execution): Running simultaneously threads-generate independent runs that are executed in parallel.
- Event handling: e.g., pressing a key performs some defined action.
- User interface design: For example, using clickable objects to create buttons.
- Keyboard input: to prompt users to type.

If the developer is unfamiliar with concepts such as conditional statements, data structures, loops, or objects, it will be difficult to plan a suitable solution. Therefore, to build a solution in a specific programming language, basic knowledge of the syntax seems to be necessary [11].

In fact, many researchers, including Jeanette Wing, tried to describe and systematize these processes and based on these explorations following skills have been identified which are important for computer thinking:

- Abstraction is the process of making things more understandable by reducing unnecessary detail [4].
- Algorithmic thinking is a method of getting to a solution via a clear definition of the actions.
- Automation is a process that minimizes the work done, in which a machine-computer performs a set of repetitive commands and tasks quickly and efficiently [4].
- Decomposition is a way of thinking about things in terms of their component parts. The parts can then be understood, solved, developed, and evaluated separately [4].

- Debugging is the systematic process of analysis and evaluation using competences such as testing, tracing, and logical thinking to predict and verify outcomes [4].
- Generalization is a way of quickly solving new problems based on previous solutions to problems and building on prior experience. Algorithms that solve some specific problems can be adapted to solve a whole class of similar problems [4].

In this study, the conventional Introduction to Programming curriculum is enhanced to tackle some problems of learning programming. Consequently, problem solving learning, game-based learning and computational thinking have being implemented to improve the course curriculum. The main idea of this study aims to show novice programmers the “complete view” of software development process, provide a learning environment based on game-based experience, increase student enthusiasm, and remove syntactic obstacles to coding. The method and results of the study are presented below.

### 3 Materials and methods

This study explores, how the new method of teaching using Scottie-Go block tools stimulate the behaviour of the problem-solving and coding-thinking skills by novice programmers. To make the comparison three scenarios have been examined and the classic Introduction to Python programming course was used as a reference and starting point.

#### 3.1 Classic course

Typical novice coder, who starts to learn programming, should focus on programming concepts and problem-solving algorithm rather than on language specifics, because they are different from one to another. Python provides the highest level of programming abstraction. So, the student does not have to think about memory management, which is necessary in C++, or class hierarchy, that is unavoidable in Java, or variable types and declarations, that exist in almost each programming language.

In our study we have followed typical curriculum of teaching Python for beginners based on three fundamental development principles which novice programmers need to understand and follow. First, analysing a given problem and determining the best programming strategy to implement, second creating an algorithm to solve the problem, finally converting the algorithm into the Python code. During the course all the programming concepts that novice programmers are expected to know and practically use have been introduced and verified. In addition, numerous of the computation thinking elements and problem-solving exercises and projects have been implemented in final stage of the class to build the foundation for comparison with other analysed teaching methods.

### 3.2 Scratch extension to the classic course

In Poland, teachers have quite a lot of freedom in the selection of tools for learning programming in contemporary K-12 computer science education and among the two most frequently chosen block programming solutions, they use Balti and Scratch. Balti is a relatively unknown environment and, apart from Poland, Czech Republic and Slovakia is practically not used. In Poland, Balti has been deployed in over 4,500 schools and used by 79,755 students.

Scratch is much more often chosen by teachers and has a greater installed base and share in the education sector. Utilising the block-based programming environments like Scratch for introduction to novice programmers basic programming principles and computational thinking is not new and has been analysed and discussed by many researchers [1]. The overall result of this research has already been reviewed and it shows that Scratch enhances learning and problem-solving skills and significantly improves the interest of aspiring programmers to learn coding and develop computer science interest. Other work has shown that Scratch programming is an appropriate way to introduce younger students of all ages to the programming world. The advantage of using Scratch is that it is visually attractive and promotes active learning. Some studies have found a decrease in the number of students who dropped out and lost interest in computer once Scratch was implemented for introduction the concept of programming [17]. Unfortunately, Scratch users may be reluctant and have a difficulty to do transition to the traditional programming environment where syntax becomes an essential and important component. There is, of course, a concern that without the appropriate tools and interfaces to facilitate the transition from block tools introduced in Scratch and methods in the classic programming approach, it can be difficult.

In our scenario, we do not use Scratch as an independent module and we do not start the education process with an introduction to Scratch and then Python, but we rather introduce a specific programming concept (loop, condition, etc.) and outline it in Scratch with nice and easy to understand graphical form, and once the idea is understood and digest the same concept is described in Python. Essentially, Scratch is a nice programming block format to illustrate the idea of a specific programming concept that is much easier for a novice programmer to learn. There is a study comparing students answering multiple-choice programming questions that showed that students cope better with questions and tasks using block representation and graphical methods compared to tasks presented in text form [18].

The same concepts and plans like in classic course were also implemented to the students who participated in the extended course with Scratch enabled, also the same exercises and tests from “BEBRAS” repository have been used to validate the level of adoption for to computational thinking and finally the same testing project has been implemented in the final stage to check computer problem-solving and computational thinking ability and skills.

### 3.3 Scottie Go and Scratch extensions to the classic course

Scottie Go products are a series of innovative and unique game-based approaches to introduce novice programmers to the world of programming and creative technology. The game comprises a free to download application, with no ads nor in-play purchases, which features 91 quests of increasing difficulty, a board and 179 cardboard tiles. The app can be installed on all major operating systems. The game covers all basic concepts of programming, ranging from basic instructions, algorithms, and parameters to loops, conditional expressions, variables and functions. To play the game, students start the app and read the challenge of a specific quest. This typically requires the main character to move around the board and to perform specific actions. With the use of the cardboard tiles, players create simple coding instructions for the main character to follow. The instructions are scanned with the app, which then illustrates how the character performs the instructions on the board. This solution gives the players an immediate feedback on whether the instructions were accurate, where the issue may lie, and whether the result can be further optimized. The game requires students to complete a level before moving to the next one. However, teachers are provided with a digital key to let more able students skip specific, less challenging sections. The game enables students to gain basic and advanced programming skills and to boost their logical thinking skills in an interactive and enjoyable approach. At the same time, it allows non-specialist teachers to integrate learning activities in their lessons, which will deliver against the standards of the Computer Science curriculum. Often used to promote teamwork, Scottie Go allows learners to discover programming through a tactile and kinesthetic approach. This means that the computational thinking behind programming is illustrated in an enticing and appealing way. Furthermore, by manipulating the sequences of code with their hands on the actual board, as opposed to typing strings of code on a device, Scottie Go focuses the learner on the actual process of problem solving. The typical try and error approach, so common in traditional programming, is replaced by a more considered, thought-based process and computational thinking.

The purpose of the study was to propose and evaluate the new learning method for a novice programmer, which will incubate the computational thinking and develop a problem-solving ability, rather than only concentrate the efforts on Python syntactic and coding skills. Scottie Go as the block based and gaming-based tool of learning is great candidate to stimulate these abilities. Our method expanded the approach described in the subsection 3.2 with Scottie Go module added on the beginning of the course.

We introduce the selected quest and games from Scottie Go during first 8 weeks of the class to develop the ability of novice programmers to discover and practice planning a suitable solution. The idea is to develop a solution from a high-level design concept of building the path to the solution. This approach requires propose detailed design by transforming sub-goals into corresponding algorithms. Additionally, the system identifies the best type of implemented algorithm and rank it with the proper number of stars. More stars are the higher

ranking meaning better optimized solution to the problem. Competition aspect to plan and implement best solution, stimulates a new programmer to optimize their algorithms and incorporates fun aspect to the learning process.

### 3.4 Participants

This study was conducted at four public schools in Poland, each school had minimum three classes in the same level of education – eighth grade and almost similar size in terms of students in each class. The research sample consists of 361 students from four different schools, from twelve classes. As for gender, 184 are girls (51%) and 177 are boys (49%). Among these 361 students, 121 have taken the classic course, 118 took Scratch extension to the classic course and 122 Scottie Go and Scratch extensions to the classic course. Polish ministry of education formalizes the curriculum and allowing introduction to programming in Python in grade eight but not clarify or define the method how the subject is going to be educate. Most students usually have no prior knowledge of Python programming concepts and skills. Accordingly, their prior knowledge and programming skills are assumed to be equivalent. Four teachers participated in the experiment, each of them had 3 class, one from each type of tested methods, each teacher specialized in teaching computer science and had over 10 years of teaching experience.

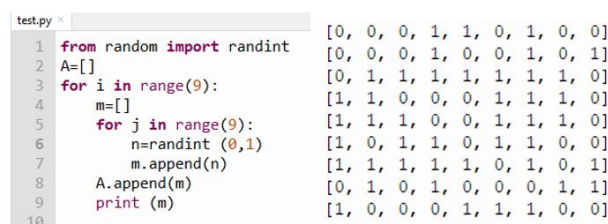
### 3.5 Research method

The purpose of the study was to propose and determine the impact of a new learning method for novice programmers to stimulate the behaviour of computer thinking and problem-solving skills. To quantity the effect of the new learning method and compare it with other tested methods, two tests were developed and carried out at the end of the curriculum as a learning outcomes control mechanism. The ideal idea seems to be to test the solution using a comparative method, this approach allows to better understand the effects of learning the concept of programming developing computer thinking and stimulating problem solving behaviour.

In its classical form, the experimental method assesses the results achieved from the control group in contrast to the results obtained by the treated group. In our scenario, the treatment consisted of implementing the proposed new method compared to the classical Python curriculum. In experimental evaluation, both groups should be equivalent if participants are randomly assigned to control and treatment groups. In our case, it was difficult to randomly assign scholars to different schools, but each school had 3 different categories of tested methods, and the experiment involved 4 different teachers and each used each method. The research procedure for the classic course was embedded with typical curriculum introduction to Python. In the first 34 weeks of the course the basics and extended programming concepts have been introduced to the students according to the curriculum and enabled all the novice programmers with all the competencies from classic course. Through this period of the class, students were



introduced to the basic programming idea and abstraction such as: if-then conditions, variables, constants, for and while loops, nested loops, operation on strings, arrays, two-dimensional array, functions and use of modules in Python. In week 35 second phase took a place, and the instructor clarified the importance of “BE-BRAS” test and mechanics behind it and special test to validate the ability of the computational problem-solving has been implemented. We used the one of the tests from the repository of “BEBRAS”- International Challenge on Informatics and Computational Thinking customized for eight grade. The implemented 45 minutes online test includes 10 problems to solve and students can achieve from 0-40 points. The test is very intuitional and does not require any knowledge of any particular programming language or any environment to be installed. All the questions in the test were dedicated to verifying the computational thinking and provide clear proof if proposed method enhance novice programmers to solve computational problems. The second testing procedure started week after in week 36 and was dedicated to verifying ability of the problem-solving and computational thinking ability and skills. First, the instructor clarified the idea of test, then students got a Python script (Figure 1) to modify it in the way to solve the problem in most optimal way. We have been ready to see various



```

test.py ×
1 from random import randint
2 A=[]
3 for i in range(9):
4     m=[]
5     for j in range(9):
6         n=randint(0,1)
7         m.append(n)
8     A.append(m)
9     print(m)
10
[0, 0, 0, 1, 1, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 1, 0, 1]
[0, 1, 1, 1, 1, 1, 1, 1, 0]
[1, 1, 0, 0, 0, 1, 1, 1, 0]
[1, 1, 1, 0, 0, 1, 1, 1, 0]
[1, 0, 1, 1, 0, 1, 1, 0, 0]
[1, 1, 1, 1, 1, 0, 1, 0, 1]
[0, 1, 0, 1, 0, 0, 0, 1, 1]
[1, 0, 0, 0, 1, 1, 1, 0, 0]

```

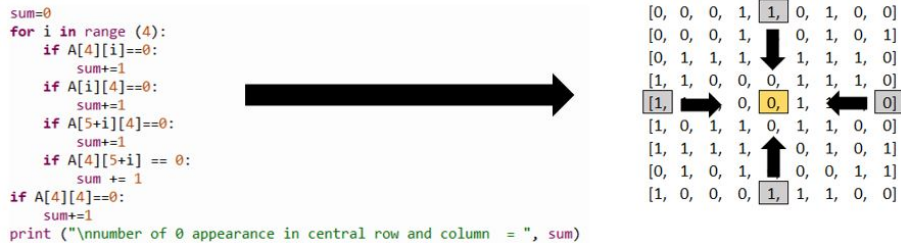
**Fig. 1.** Python code to be modified to solve the problem

programming strategies and implementation expose by the participants could manifest, but idea was noticeably clear the most efficient algorithm is going to win. Finally, we built the ranking of the solution based on the most effective algorithms from computational problem-solving perspective not from most efficient and condensed version of Python implementation. The assignment was to “calculate the number of zero appearance in central row and column in given matrix - table”. We have used the markers of programming actions for problem solving and computational thinking to judge the solution. The indicators were group in two categories:

- Computational thinking practice: simply iteration, nested or multi-conditional iteration, agile style.
- Computational solving practice design: problem decomposition, sequential approach, selective approach.

Based on the indicators and overall algorithm quality (if the problem was solved or not) we have assessed the solution. We have observed different solution to

the problem which can be group in 3 categories. Categories were created based on the way the problem was decomposed and numbers of iteration in the loops. The first category was 9 steps looping and there we have observe two type of implementations two sequential loops of 9 steps or one loop of 9 steps. Second category was loping with 4 steps and there we have seen also 2 subcategories 4 independent loops or 2 loops with 4 steps each. The final most sophisticated category was we called agile or spiral approach where students in 4 steps loop completed the task. Last approach is presented in Figure 2. In addition to the category of implementation we have also judge how the exception and replication have been handled and how the central A[4,4] element of the matrix has been managed (some solutions counted A[4,4] element many times).

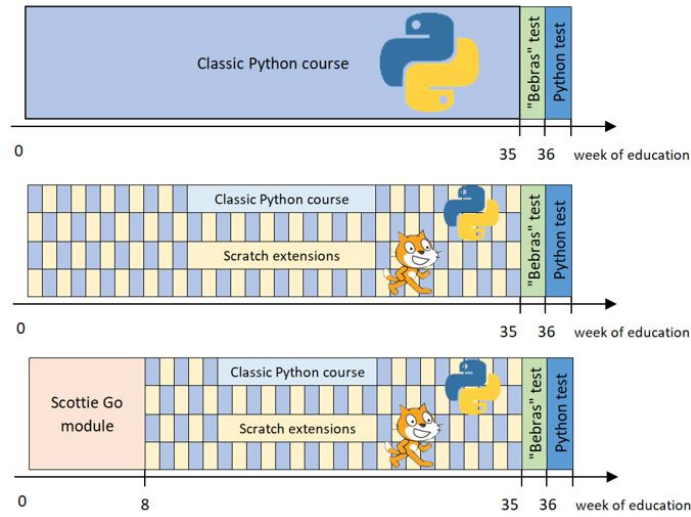


**Fig. 2.** One of the six type of solution: spiral approach with 4 steps loop (centripetal)

Taking into account indicators, type and quality of the proposed algorithm as well how the exceptions have been handled students were able to achieve from 0-100 points from the final test. Of course, they were fully aware that their results from the project would be considered as important component of their mark for the course. The exactly the same components have been used for verification of two other teaching methods, the only difference is that in Scratch extension to the classic course and Scottie Go and Scratch extensions to the classic course, the curriculum has been modified, but testing methodology implemented in week 35 and week 36 remains exactly the same. Figure 3 shows the timeline for all 3 type of courses.

## 4 Results

The data gathered in this study included information about the results of computational problem-solving activities practices, strategies, and as well results of computational thinking test. Collected indicators are representing the computational thinking activities in the action, which is a computational problem-solving task. These indicators could be classified as two dimensions: computational thinking practice and computational solving design. To discover the student's patterns of computational thinking in the action of problem solving, we deployed "BEBRAS" test. In order to verify our postulate that students with



**Fig. 3.** Timeline of all 3 courses

the Scottie Go and Scratch extension to the classic course perform better than students from the classic course (our control group), the "BEBRAS" test was carried out. In addition, there was also an evaluation of the intermediate group, which was based on the adding Scratch modification to classic course. In this evaluation, the dependent variable was student performance determined after taking the "BEBRAS" test. On average, the students in the Scottie Go and Scratch extensions to the classic course performed better ( $M = 26.00$ ,  $SD = 10.16$ ,  $AVG = 24.01$ ) than those in the classic course ( $M = 21.00$ ,  $SD = 10.82$ ,  $AVG = 21.66$ ) and the Scratch extensions to the classic course ( $M = 24.00$ ,  $SD = 10.60$ ,  $AVG = 23.64$ ). The percentage of scores between 20-40 points (best scores) is also important. The highest score is 70% in favor of Scottie Go and Scratch extensions to the classic course. The worst results of this parameter were achieved after completing classic Python course (55%). Score for Scratch extensions to the classic course was better than classic course (63%) (Table 1). The assessment of all courses is performed in identical procedures.

It must be also distinguished that the failure rate for all type of courses is similar. The "BEBRAS" test was not passed by those students who did not attend classes regularly and we do not see any correlation with the version of the course they were assigned.

The proposed test - "calculate the number of zero appearance in central row and column in given matrix - table" examines computational thinking models and problem-solving trials. This procedure resulted in a seven group of results based on the indicators and overall algorithm quality. This solution provided clear distinctions and meaningful explanations for the different models of computational practice. In addition to 3 categories how students solved the problem

**Table 1.** Statistical comparison of the results from 3 courses (“BEBRAS” test).

	Classic Python	Scratch extensions	Scottie extensions
Median	21.00	24.00	26.00
Standard deviation	10.82	10.60	10.16
Average	21.66	23.64	24.01
Percentage of results between 0 and 20 points	45.00	37.00	30.00
Percentage of results between 20 and 40 points	55.00	63.00	70.00

based on the way how the problem was decomposed and numbers of iteration in the loops (described in the subsection 3.5), we also recognized four group, which were not able solve the problem, but they were deploying some of the techniques which can be judged according to our indicators. These group we named and evaluated accordingly: Kamikaze approach, Try and Fail approach, Sequential approach, Selective approach. Table 2 shows 7 groups of students and how they are performing according to our indicators cross the three different teaching methods we have examined.

**Table 2.** The results of the Python task divided into 7 groups of solutions.

Name of group	Points	Classic Python	Scratch extensions	Scottie extensions
Kamikaze	0-5	6	5	5
Try and Fail	6-15	15	12	13
Sequential	16-34	32	28	26
Selective	35-64	41	43	44
9 Loops	65-84	13	12	13
4 Loops	85-94	11	13	15
Agile/Spiral	95-100	3	5	6

The dependent variable in this comparison was the number of points obtained after completing the test “calculate the number of zero appearance in central row and column in given matrix - table”. On average, students accomplished the Scottie Go and Scratch extensions to the classic course much better ( $M = 44.00$ ,  $SD = 28.41$ ,  $AVG = 47.77$ ) than these in classic Python course (control group) ( $M = 42.00$ ,  $SD = 26.92$ ,  $AVG = 42.83$ ) and also better than these in Scratch extensions to the classic course ( $M = 41.00$ ,  $SD = 27.99$ ,  $AVG = 46.86$ ). The highest percentage of scores between 20-40 points is for Scottie Go and Scratch extensions to the classic course – 44%. The worst result of this parameter was achieved after completing classic Python course (39%). Score for Scratch extensions to the classic course was better than classic course (42%) (Table 3).

Finally, after two test we have collected feedback from the classes where Scottie Go and Scratch extensions to the classic course has been implemented and

**Table 3.** Statistical comparison of the results from 3 courses (Python exam).

	Classic Python	Scratch extensions	Scottie extensions
Median	42.00	41.00	44.00
Standard deviation	26.92	27.99	28.41
Average	42.83	46.86	47.77
Percentage of results between 0 and 20 points	61.00	58.00	56.00
Percentage of results between 20 and 40 points	39.00	42.00	44.00

students have declared that Scottie missions cause them to spend more time and to be more concentrated on the curriculum, also this extension motivated them to solve and complete the mission at home, helped to learn algorithms and programming concepts in the subsequent part of the course, improve creativity, expand problem solving skills and made programming pleasurable more. In addition to these results, this paper looked at students' progress on the Scottie Go and Scratch extensions to the classic course to understand the impact of reducing the length of the classic course by 8 weeks and adding instead a computational problem-solving module at the start of the course. On the one hand, this module enriches the development of these skills, but at the same time reduces the time needed to learn the syntax elements related to Python programming.

## 5 Discussions and Conclusions

The goal of the research was to validate the effect of a new learning method dedicated to novice programmers, specifically designed to develop computational thinking and problem-solving approach utilising a block-based development environment. The impact of the Scottie Go and Scratch extensions to the classic course on novice programmers was evaluated by two research tests (Section 4). Each of the research test was assessed by comparing to classic curriculum Introduction to Programming and also to extended version of this course with incorporated Scratch.

The study shows that problem solving through using a block-based development environment is effective way to support new inexperienced programmers to absorb coding and computational design concepts. The results obtained during the study show a significant improvement in the skills of computer understanding, thinking and solving computer problems at this stage of education. This suggests the possibility of recommending to the education authorities to introduce block tools in the core curriculum on Introduction to Programming.

This article also contributes to our understanding and perception of the relationship between designing development environments and leveraging the development practices they generate. It helps us analyse the relationship between design and learning, especially with regard to programming. Nevertheless, the authors recognise a further need to explore the relationship between the ability

of programmers to better model physical phenomena and the modelling of the natural world and the usage of visual programming tools and good practices.

The analysis in this study also recognized different computational practice patterns and design strategies for solving computational problems - we observed three categories of proposed solutions. Taking into account the difference in solving problems by the participants, further research into the use of block design development environment should take into account such individual differences. However, the areas of computational problems in this study are specific, which may reduce other behaviour patterns and project strategies. In such case future research should analyse different aspects of novice programmers' processes of using a block-based development environment.

It is worth analysing potential actions taken by novice programmers during the development process, such as reviewing bugs, improving plans, and tracking code, that could provide better insight into behaviour patterns and design strategies presented by novice developers. It is essential to have a comprehensive and holistic view of how these design choices affect newcomers [19].

Taking into consideration the feedback given by students and teachers at the end of the course it would also be valuable to integrate game elements into the course, to see how this advancement motivates students to increase computation problem solving capability.

In future research it is also valuable to investigate, if possible, the introduction of an admission test to better understand the level of knowledge and skills of the groups at the beginning of the course and potentially improve the adaptation of the method used.

The fundamental goal of this research is that it will help form the next generation of initial computer science learning environments and thus shape the next generation of learners, improving programming skills utilising block-based tools to develop problem solving abilities. This approach has the potential to help students achieve better results when introduced to a programming course, which in turn can have an impact on their performance in future IT projects.

**Acknowledgements** Authors want to thank the anonymous reviewers whose suggestions significantly improved the quality of this manuscript.

## References

1. Aivaloglou, E. & Hermans, F. How kids code and how we know: An exploratory study on the Scratch repository. *Proceedings Of The 2016 ACM Conference On International Computing Education Research*. pp. 53-61 (2016)
2. Bower, M., Wood, L., Lai, J., Howe, C., Lister, R., Mason, R., Highfield, K. & Veal, J. Improving the computational thinking pedagogical capabilities of school teachers. *Australian Journal Of Teacher Education*. **42**, 4 (2017)
3. Chao, P. Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers Education*. **95** pp. 202-215 (2016)

4. Fagerlund, J., Häkkinen, P., Vesisenaho, M. & Viiri, J. Computational thinking in programming with scratch in primary schools: A systematic review. *Computer Applications In Engineering Education*. (2020)
5. Garcia-Peñalvo, F. & Mendes, A. Exploring the computational thinking effects in pre-university education. (Elsevier,2018)
6. Hosseini, R., Akhuseyinoglu, K., Brusilovsky, P., Malmi, L., Pollari-Malmi, K., Schunn, C. & Sirkiä, T. Improving engagement in program construction examples for learning Python programming. *International Journal Of Artificial Intelligence In Education*. **30**, 299-336 (2020)
7. Hu, Y., Chen, C. & Su, C. Exploring the effectiveness and moderators of block-based visual programming on student learning: A meta-analysis. *Journal Of Educational Computing Research*. **58**, 1467-1493 (2021)
8. Israel-Fishelson, R. & HersHKovitz, A. Persistence in a game-based learning environment: The case of elementary school students learning computational thinking. *Journal Of Educational Computing Research*. **58**, 891-918 (2020)
9. LaToza, T., Arab, M., Loksa, D. & Ko, A. Explicit programming strategies. *Empirical Software Engineering*. **25**, 2416-2449 (2020)
10. Mathew, R., Malik, S. & Tawafak, R. Teaching Problem Solving Skills using an Educational Game in a Computer Programming Course.. *Informatics In Education*. **18**, 359-373 (2019)
11. Papadakis, S., Kalogiannakis, M., Orfanakis, V. & Zaranis, N. The appropriateness of scratch and app inventor as educational environments for teaching introductory programming in primary and secondary education. *Early Childhood Development: Concepts, Methodologies, Tools, And Applications*. pp. 797-819 (2019)
12. Papadakis, S. Apps to Promote Computational Thinking Concepts and Coding Skills in Children of Preschool and Pre-Primary School Age. *Mobile Learning Applications In Early Childhood Education*. pp. 101-121 (2020)
13. Pellet, J., Dame, A. & Parriaux, G. How beginner-friendly is a programming language? A short analysis based on Java and Python examples. *University of Cyprus*,(2019)
14. Theobald, E., Hill, M., Tran, E., Agrawal, S., Arroyo, E., Behling, S., Chambwe, N., Cintrón, D., Cooper, J., Dunster, G. & Others Active learning narrows achievement gaps for underrepresented students in undergraduate science, technology, engineering, and math. *Proceedings Of The National Academy Of Sciences*. **117**, 6476-6483 (2020)
15. Topalli, D. & Cagiltay, N. Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers Education*. **120** pp. 64-74 (2018)
16. Visnovitz, M. Classical Programming Topics with Functional Programming. *Central-European Journal Of New Technologies In Research, Education And Practice*. pp. 41-55 (2020)
17. Weintrop, D. & Wilensky, U. Between a block and a typeface: Designing and evaluating hybrid programming environments. *Proceedings Of The 2017 Conference On Interaction Design And Children*. pp. 183-192 (2017)
18. Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L. & Wilensky, U. Defining computational thinking for mathematics and science classrooms. *Journal Of Science Education And Technology*. **25**, 127-147 (2016)
19. Weintrop, D. & Holbert, N. From blocks to text and back: Programming patterns in a dual-modality environment. *Proceedings Of The 2017 ACM SIGCSE Technical Symposium On Computer Science Education*. pp. 633-638 (2017)