AlgoPoint as an Original Didactic Tool for Introductory Programming Using Flowcharts

Marcin Stefanowicz and Anna Sasak-Okoń[a]
Maria Curie-Skłodowska University, Pl. M. Curie-Skłodowskiej 5, 20-031 Lublin, Poland

Keywords: Flowcharts, AlgoPoint, Authoring Didactic Tool, Visual Programming, Block-Based Programming.

Abstract: The paper concerns the development of the authoring application called AlgoPoint, which supports high school students in the process of learning programming. AlgoPoint is a block-based flowchart editor. It separates the syntax and semantics of the programming language from the actual control flow of the algorithm. In this way, students can focus on developing problem-solving skills, expressing a solution to a problem by building it using various functional blocks and connections. For more advanced users, mechanisms are available to build more complex data structures, functions, or subroutines, which can then be tested with the use of the built-in console or expressed as a text code in the selected programming language. The usefulness and advantages of the proposed application have been positively verified by students in real high school. The verification was conducted in natural conditions during computer science classes under the supervision of a teacher.

1 INTRODUCTION

Before becoming an exemplary programmer, every individual should comprehend the solid foundations of computational thinking (CT). The first places where a young person acquires this knowledge are primary and secondary schools. Therefore, as a crucial factor in the modern fields of STEM (Science, Technology, Engineering and Mathematics), CT ought to be applied in computer science classes (Henderson et al., 2007; Yadav et al., 2011). The problem-solving approach is often identified as a foundation of programming education and can be developed through the use of appropriate methods and tools (Romeo et al., 2017).

Increasing proficiency in CT and programming can be accomplished through text-based, visual and hybrid approaches. First of all, textual coding skills are useful in many areas, as the demand for programmers has grown rapidly in recent years. However, it can be complicated for novices to start learning directly from writing code. They suffer from a wide range of difficulties, such as a lack of full comprehension of the problem itself, various programming concepts or structures, program debugging, English coding language and its structure (Janpla and Piriyasurawong, 2018; Robins et al., 2003). Different types of visual tools can be helpful in this case. In recent years, games that teach through play have become increasingly popular (Rugelj and Lapina, 2019). One can find games designed for children, but also applications suitable for older students. Another very important tool supporting the process of learning programming is the flowchart. A clear, graphical representation of program components and its dependencies helps students understand the mechanisms of particular algorithms beyond the syntax and semantics of statements. The purpose is to illustrate how algorithms should be logically organised, leading to a proper structural solution to a given problem, which is then translated into a computer program (Weintrop, 2021; Kraleva et al., 2019).

In this paper, we demonstrate a new authoring tool called AlgoPoint, which can be used as a support for self-learning introductory programming or as a didactic tool for computer science teachers. AlgoPoint is a modern application designed for high school students who struggle with learning the basics of programming and algorithmics. Our program offers many functionalities, starting from a user-friendly Flowchart Editor to building algorithms with the use of a different type of blocks and connections. The contents of the blocks can be edited to take advantage of more advanced features of the programming language. Created algorithms can be analysed and tested with the support of built-in console and converted to source code.

[a] https://orcid.org/0000-0002-4593-120X

Stefanowicz, M. and Sasak-Okoń, A.
AlgoPoint as an Original Didactic Tool for Introductory Programming Using Flowcharts.
DOI: 10.5220/0011826300003470
In *Proceedings of the 15th International Conference on Computer Supported Education (CSEDU 2023) - Volume 1*, pages 162-170
ISBN: 978-989-758-641-5; ISSN: 2184-5026

The paper text that follows contains 4 Sections. Section 2 discusses the related works. Section 3 describes in detail the functionalities implemented in AlgoPoint with the example of a particular flowchart representing a simple algorithm and its source code. Section 4 presents the results of research conducted in real high school conditions, which prove the practical utility of our application. Section 5 includes the concluding remarks.

2 RELATED WORK

There are many textual and visual programming languages available today that are applied in the process of learning computer science. This group includes popular programming languages used in industry, e.g. Python and JavaScript Amid other textual languages, some original solutions can be found, for instance, MiniScript (Strout, 2021), Coral (Edgcomb et al., 2019) and SIMPLE (Rababaah, 2020). Most of them are simplifications of already existing programming languages, created to aid students switching smoothly between languages used globally.

In literature, we can trace a variety of games that support learning programming, such as Tuk Tuk (Koracharkornradt, 2017) or AstroCode (Bione et al., 2017). Such games, also called serious games, help students develop computational thinking skills in a friendly game environment. Using stories and maps with different difficulty levels encourage players to practice their problem-solving skills in ways not possible in the real world.

As for the block-based approach, one of the most popular tools is Scratch (Meerbaum-Salant et al., 2010). This application enables the creation of simple games and interactive programs. It is designed for children who learn while creating their games and animations. What is more, Scratch is always free and available in as many as 70 languages creating a kind of coding community for children, encouraging self-expression and collaboration. Comparable solutions to this platform are, for example, Alice, App Inventor and SmartBuilder (KAYA and YILDIZ, 2019; Werner et al., 2012). There is also a text-visual hybrid named Pytch created at Trinity College Dublin, which is the environment inspired by Scratch that uses Python instead of blocks (Strong and North, 2021). It helps children to step forward from Scratch's blocks drag'n'drop to the concept of writing code in Python.

Also worth mentioning is a group of applications that are aimed at older students who are not yet proficient in programming but still enjoy the concept of block-based code building. One of them is Flowgorithm. Unlike Scratch, the blocks have different shapes for specific functionalities and are not placed like puzzles on a stack, but are appropriately connected by lines with arrows. The notation of the operators used in the expressions contained in used is mixed, i.e. one operation can be performed using several operators which can be misleading. On the other hand, Flowgorithm allows one to preview the source code properly generated in various programming languages (Cook et al., 2015). Two more applications with similar functionalities are Raptor and Visual Logic. The characteristic feature of Raptor is the ability to choose the level of advancement, from basic to even object-oriented programming. However, certain elements, such as the appearance of the scheme and the language of the application, cannot be changed. Another similar application is Visual Logic. Despite its similarities to Flowgorithm, it lacks some rudimentary features. For instance, the speed of algorithm execution cannot be controlled and exporting a diagram to an image is impossible (Kourouma, 2016). Aside from that, there are other programs such as Microsoft Visio that are used to model and visualise multiple types of information without carefully checking the execution of the algorithm (Yu and Xiong, 2018).

Based on our knowledge and the availability of information about other applications, AlgoPoint stands out from others with such features as extensive personalisation of the flowchart elements, detailed Functions and Data Editor, adjustment of many algorithm execution parameters, as well as a user-friendly modern user interface.

3 AlgoPoint APPLICATION

AlgoPoint was designed to help beginning programmers, especially elementary and high school students (hereinafter referred to as users), develop computational thinking and understanding fundamental programming and algorithmic issues. The intention was to make the process of algorithm development as easy and intuitive as possible. Next to the clear visual process of algorithm building with predefined functional blocks, a few other features are worth noting, such as analysis of the algorithm execution with the possible adjustments of execution parameters, e.g. the duration of a single preview step. Another feature is the construction of a universal language used in all given statements and expressions that could be converted into several well-known programming languages. It is also worth mentioning that both the content of the blocks and their appearance can be freely edited.

The application in question was built in Lazarus.

It is a free cross-platform visual development tool for rapid application development (RAD). The IDE uses the Free Pascal compiler and consequently Object Pascal language (Ribić and Beganović, 2013). Moreover, it provides the Lazarus Component Library (LCL) as well as the core Free Pascal Runtime Library (RTL) and Free Component Library (FCL) where many visual and non-visual components can be found (Gaertner, 2009; Person, 2013).

3.1 Graphical User Interface

The graphical user interface of the main AlgoPoint window consists of five functional areas: Ribbon Menu (on the top), list of functions in the project (on the left), preview of the current state of the declared variables (on the right), console and editor of blocks and connections (in the middle), as presented in Fig. 1. It is worthwhile to note that the program is DPI-aware. Regardless of the screen resolution, this feature causes visual objects to be correctly aligned and scaled on many types of screens (Mishra and Schrawankar, 2014).

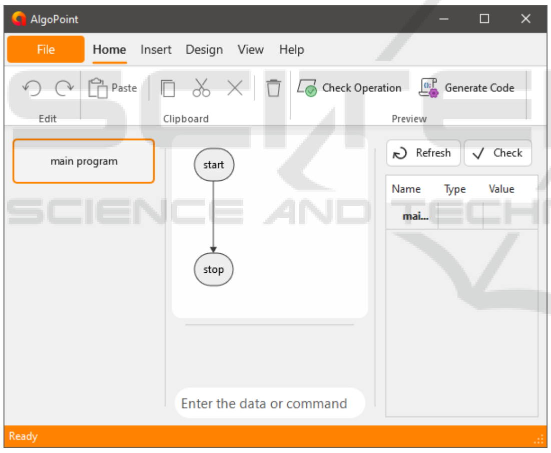


Figure 1: The main AlgoPoint window.

The Ribbon is a graphical menu component representing a set of tools organised by properly named tabs, containing different types of objects (buttons, text boxes, combo boxes, numeric fields), and grouped according to the implemented functionalities. Furthermore, a formatted hint window appear on mouse hover underneath each element with information, such as title, description and shortcut. Such an organisation of a significant part of commands aims to make navigation as efficient as possible. The Ribbon menu includes the following pages:

- Home – a page with basic flowchart editing commands, such as copying and cutting.
- Insert – includes a gallery of shapes from which the user can choose flowchart elements.
- Design – a set of tools for modifying the appearance of blocks and connections.
- View – consists of both controls for adjusting the editor's zoom percentage and changing the visibility of some interface elements.
- Help – has links to the official website and the feedback form.
- Algorithm Testing – a special page that may be activated by clicking the Check Operation button on the Home page. The commands contained here allow the user to control the preview of the algorithm's operation.

Another part of the application is the File menu, which can be opened by clicking on the orange button in the upper left corner of the Ribbon, as illustrated in Fig.1. Based on Fig.2, three areas of the File menu can be distinguished: the orange side panel (on the left), the title bar and the content of the selected side panel tab. The Home page shows up to a dozen recently opened projects. Project Properties (e.g. author and modified date) are listed in the Info section. On the New Project page, the user can search for and select a template to be applied to the new project. In the Print or Export tab, the currently open subroutine schema may be printed or exported to a PDF or image file (in PNG, JPEG, TIFF and BMP format).

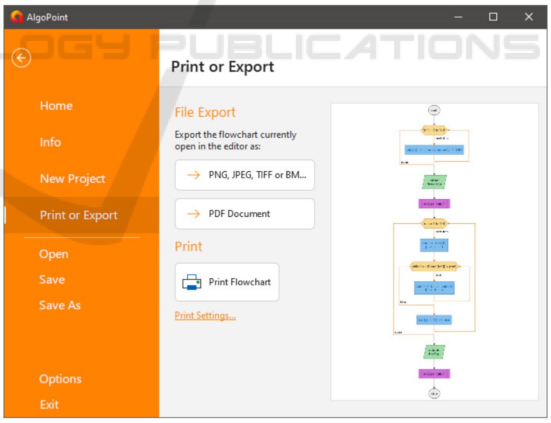


Figure 2: The Print/Export page in the File Menu.

3.2 Flowchart Editor

As previously shown in Fig. 1, the Flowchart Editor is located in the central area of the main window and is the core of the AlgoPoint. This component is designed to ensure smooth project creation and analysis. Blocks and connections contained in it may be formatted using options from the Design page in the Ribbon menu, such as font name, fill colour, border width and style selection from the premade library.

Their content can also be modified, depending on their type. In any design function diagram, the main

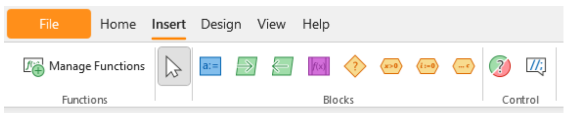
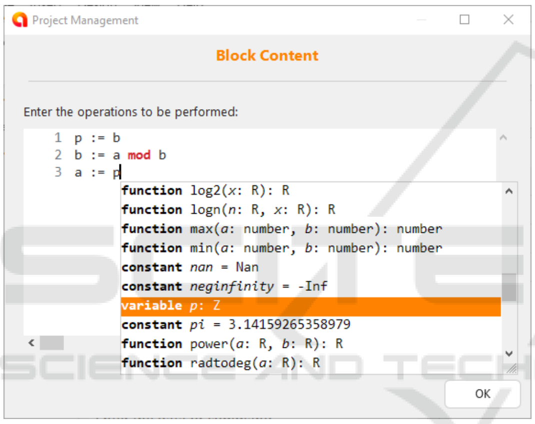


Figure 3: Available elements on the Insert page that can be placed on the flowchart.

part is a start-stop block into which is then extended with other shapes representing certain activities, e.g. data input or subroutine call. Such elements, are inserted by selecting one of those located in the Ribbon Insert page, as presented in Fig. 3. Afterwards, when the user hovers the cursor over the indicated connection, a grey plus symbol appears. At this point, after the mouse click, the given type of block will be immediately added in the designated area.



Enter the operations to be performed:

```
p := b
b := a mod b
a := p
        function log2(x: R): R
        function logn(n: R, x: R): R
        function max(a: number, b: number): number
        function min(a: number, b: number): number
        constant nan = Nan
        constant neginfinity = -Inf
        variable p: Z
        constant pi = 3.14159265358979
        function power(a: R, b: R): R
        function radtodeg(a: R): R
```

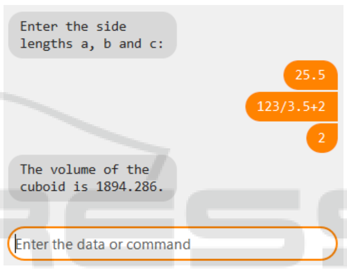
Figure 4: Editing the contents of an operation block.

Each block in the editor has its content depending on its type. To edit it, double-click on the selected element or choose the Edit Block option in the radial context menu. The Project Management window will hereafter be opened, where all possible options for adjusting the individual properties of the block can be found. For instance, as demonstrated in Fig. 4, for an operation block, the user provides instructions in the source code editor with syntax highlighted. In addition, this editor is equipped with code hints and completion as a pop-up window, where all available functions, constants and variables for the currently open subroutine scheme are placed.

The Editor in question, like a large percentage of other programs, has its context menu, which in AlgoPoint, it has been designed in a rather unconventional way. The respective functionalities are arranged around the back (or close) button inside a clearly framed circle (with the More options auxiliary button). This type of menu, apart from a modern appearance, enables much simpler and faster navigation between actions not only on the computer screen but also on the interactive board.

3.3 Built-in Console

In AlgoPoint, a dedicated console has been developed for interactive communication between the application and the user. Commonplacely, the traditional console is associated by users with a completely black window in which only commands that perform the intended actions are entered. As can be seen in Fig. 5, this component was created differently in AlgoPoint. Messages appear as balloons from two opposite sides. Grey balloons with application notifications pop up on the left edge, while orange balloons with entered data or commands appear on the right. This type of console is more user-friendly due to its readability and resemblance to many classic instant messengers.



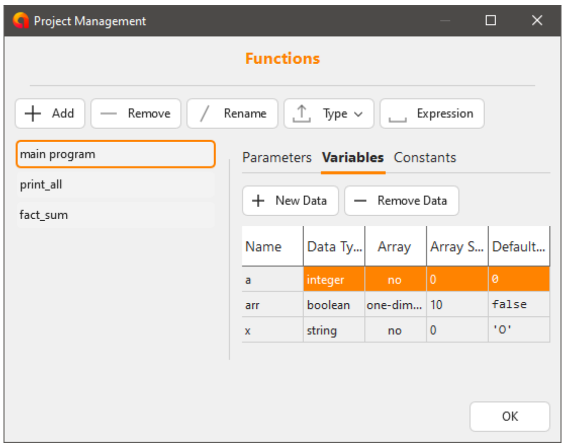
```
Enter the side
lengths a, b and c:
                        25.5
                   123/3.5+2
                           2
The volume of the
cuboid is 1894.286.
```

Figure 5: Sample console messages.

The console has some basic commands built in, e.g. *ap_clear* (clears the console), *ap_action* (executes the given command in the argument), *ap_change_vars* (toggles the ability to change variable values from the console level). Additionally, the user can enter any expression, which will be directly evaluated and passed to the application after pressing the Enter key. Such an attribute of the program simplifies the process of creating a project, as it allows for more complex tests of the algorithm's operation generated manually. For example, in Fig. 5, it could be seen that someone entered the numerical values 25.5, 2 and the expression 123/3.5+2, on the basis of which the cuboid volume was correctly calculated equal to 1894.286.

3.4 Variables and Expressions

The flowchart programming language in AlgoPoint, is statically typed, i.e. that any declared variable must have a predetermined type. Amid the variety of programming languages, a number of variable types may be found. They impose constraints so that the proper operation of the program can remain maintained (Cardelli and Wegner, 1985). In our


Project Management

Functions

main program
print_all
fact_sum

Name	Data Ty...	Array	Array S...	Default...
a	integer	no	0	0
arr	boolean	one-dim...	10	false
x	string	no	0	'0'

Figure 6: Project Functions and Data Editor.

application, there are five built-in data types available: *Integer* ($\mathbb{Z} \cap [-2^{63}; 2^{63})$), *Floating Point Number* ([5E$-$324; 1.7E308]), *Boolean* (true/false values), *String* (any character sequence in UTF-8 encoding), and *Character* (a character from the ASCII set).

An integral element of the application is the Function Management window in the project. To open it, the user should double-click the main start-stop block or once on the Manage Functions button in the Ribbon Insert tab. Based on Fig.6, several areas can be distinguished. At the top, there are buttons for adding, removing, renaming, and editing the function type. Editing is also available for the expression returned by the function. The subroutine is selected from the list on the left. The main (central) panel has a table divided into three categories. In the Variables and Constants sections, the user can modify the following data properties: name, type, array dimension, array size and default value. For the Parameters category, only the first two columns are available. Any changes introduced are updated regularly. It should be noted that it is possible to create one-, two- and three-dimensional arrays. Furthermore, the given default value is always set for each of its elements.

Next to the Flowchart Editor on the right pane (Fig.1), there is a list of all declared variables. The tree list view shows hierarchically arranged data in the function-variable relationship, divided into three columns: name, type and value. The information presented in this way is much more intelligible and accessible than, for example, a grid layout. There are also two buttons, Refresh and Check, at the top of this area. After clicking on the first of them, the view will be updated immediately, and on the second one, the window for checking the value of any expression will be displayed. To perform a specific action on one or more elements, operators should be used. In AlgoPoint, these are single- or binary-argument constructs that either return a certain result or set the value of a variable. According to the assumptions of our project, they are all part of the universal language of expressions and instructions. The available operators are modeled on those of the Free Pascal language (e.g. $:=$ means assignment, $=$ equality, $/$ float division, *not* stands for logical or bitwise NOT depending on the given values). They can be divided into five groups: arithmetic, logical, bitwise, comparisons and assignments. Each expression and instruction are written in case-insensitive infix notation. It means that for binary operators, the operands are on both sides. To change the order of performed operations, parentheses ought to be applied. During the evaluation of the expression, the notation is changed from infix to Reverse Polish Notation, which makes numerous complex operations of parentheses easier to perform on a computer (Ben-Ari, 2012).

When creating any expression in AlgoPoint, intrinsic functions and constants might be used. The user can choose from a set of 46 functions. Generally, they can be distinguished into the following categories: type conversion (e.g. AsChar, AsDouble), converting units (RadToDeg, DegToRad), type casting (e.g. ToBoolean, ToInteger), mathematical functions (for instance, Abs, Round, Sign), files (FileExists, LoadText), value checking (IsInfinite, IsNan), text and arrays (ALength, SLength, CharAt). Aside from them, AlgoPoint implements four basic floating point constants: *pi* (π), *nan* (an undefined and unrepresentable value), *infinity* ($+\infty$) and *neginfinity* ($-\infty$).

3.5 Block Types and Layouts

Over the years, several standards for creating flowcharts have been developed. Amidst them, the ANSI standard (Chapin, 1970) was proposed in the last century, as well as other forms of schemas, e.g. for structured programming (Nassi and Shneiderman, 1973) and information processing (Rossheim, 1963). Currently, industry and education are mainly using the IBM standard, intermittently with some variations (Cook et al., 2015). Thus, many elements of this standard have been implemented in AlgoPoint. As for the types of blocks, the following were created (the symbols of which are shown in Fig. 3):

- **Operations** – a rectangle with source code representing one or more operations to be performed. They generally change or assign new values to variables. Each instruction must be written successively after the newline character.
- **Input** – allows the user to enter the data. This element has the form of a parallelogram. Inside

is a semicolon-separated list containing variable names with or without indices in square brackets (depending on whether the variable is an array).

- **Output** – displays the result with the given text format, which is similar to the C-style string formatting. The available argument types are *d* (decimal numeral), *e* (scientific notation), *f* (floating point number), *s* (text or character) and *x* (hexadecimal representation). Additionally, the $\%\backslash n$ specifier is used to insert a newline. This type of block has the same shape as the input block.
- **Subroutine Call** – allows the user to call a declared function. The return value can be assigned to a variable of the appropriately matched data type. The form of this process outline is a rectangle with double-struck vertical edges.
- **Decision** – a rhombus-shaped block showing a conditional operation. It has two branches (*false* and *true*) that define the further path of the program's execution.
- **Loop with Precondition** – evaluates a Boolean expression and if true, executes the statements on the *true* branch. The expression is then parsed again. If the result turns out to be false, loop is terminated. It should be noted that any type of loop is represented by a hexagon.
- **Loop with Postcondition** – in contrast to the precondition loop, it evaluates a logical expression at the end. It means that all the statements are executed at least once.
- **Counter Loop** – is a traditional for-loop that increments or decrements the loop counter (previously declared variable) by one in the range of given values. This kind of loop is sometimes an adequate replacement for the precondition loop.

In addition to the aforementioned types of blocks, two more are worth noting. The annotation block is an area where the user can append a description to a given fragment of the function diagram. Another element is the breakpoint. It is used to deliberately suspend the preview of the algorithm's operation in order to analyse its parts in greater detail. After generating the source code of the project, both the first and the second kind of block are presented as comments in the chosen programming language.

Compound blocks, such as loops and conditional statements, have branches on which other blocks can be placed (as illustrated in Fig.7). The individual elements of the branches are executed depending on the fulfillment of a particular condition. In the *If* statement, the *true* branch is on the right, and the *false* is on the left. The difference in the loop layouts is that the *true* branch is in the middle and goes straight down. This arrangement is designed to maintain a coherent and logical representation of the algorithm's flow of action.

3.6 Algorithm Testing

In order to check the algorithm execution process, the user can use the built-in functions. By clicking on the Check Operation button on the Home tab, a special Algorithm Testing tab will open. It contains options to Run, Pause and Stop execution. The user can also control the duration of a single step (default - 1 s, immediate action - 1 ms or another numerical value expressed in seconds) or switch the mode of one step execution. As an example of an algorithm built with the use of the AlgoPoint, consider the flowchart in Fig. 7, which represents the calculation of the nth term of the Fibonacci sequence. As we can see, it was built with the use of four types of available blocks including operations (blue rectangle), input (green parallelogram), decision (yellow rhombus) and loop with the precondition (yellow hexagon). First, the assumption is that *n* is a natural number. If the user enters a valid value of *n*, e.g *n*=10, the text 'Fib(10) = 55' should be presented in the console. However, if an invalid value such as 3.5 or 'txt' is given, the preview will terminate immediately, and the error message 'Invalid input data' will be displayed.

3.7 Generating a Code

As we said at the beginning, flowcharts are one of the primary ways to describe an algorithm. Nonetheless, they are not used in program development, as opposed to text-based major programming languages. Once the process of creating flowcharts has been mastered to a great extent, the user should be able to apply this knowledge at a higher level. To transfer elements from the visual representation of the algorithm to the text code, a functionality of generating source codes has been implemented in AlgoPoint. This feature converts all existing flowcharts in the project to the source code in the selected programming language. It aids users to perceive how an algorithm can be coded in the following languages: Free Pascal, C++, C#, Java SE, JavaScript and Python. The source code preview will appear when one press Ctrl+J or click the Generate Code button in the Home tab on the Ribbon. The read-only editor shows code in the selected programming language with syntax highlighting. The buttons at the top allow the user to select one of the available programming languages, copy the entire source code to the clipboard and save it to a file

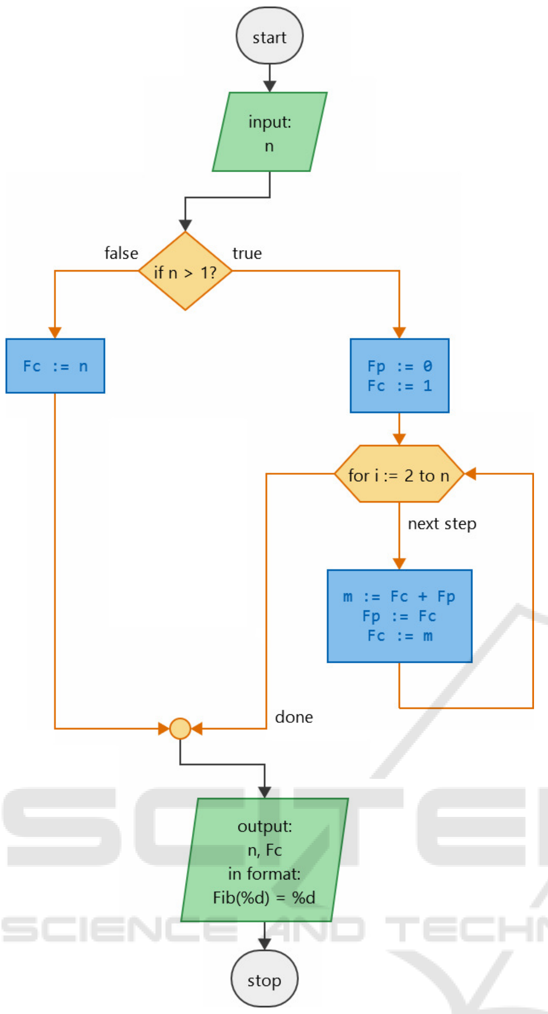


Figure 7: Flowchart for calculating the value of the nth term of the Fibonacci sequence.

with an extension appropriate to the current language. For example, a C++ source code for the flowchart in Fig.7 will be generated as follows:

```
int n = 0; int fc = 0; int fp = 0;
int m = 0; int i = 0;
int main() {
  srand(time(NULL));
  cin >> n;
  if ((n > 1)) {
    fp = 0;  fc = 1;
    for (i = 2; i <= n; i++) {
      m = (fc + fp);
      fp = fc;
      fc = m; }
  } else {
    fc = n; }
  printf("Fib(%d) = %d", n, fc);
  return 0; }
```

4 AlgoPoint IN SCHOOL

After the implementation part of the AlgoPoint was completed, the authors decided to conduct live test in one of the high schools in Poland. Informatic courses in Poland are split into two stages. In primary school, pupils practise basic IT skills. The second and high school stage emphasizes the problem-solving and decision-making skills using an algorithmic approach (Sysło and Kwiatkowska, 2015). However, it should be noted that programming and algorithmics education is universal, thus, the results and conclusions of this research are not limited to the country of the study.

About 100 students from differently profiled classes (with mathematical and IT, mathematical and physical, humanities and law, biological and chemical profiles) in Boleslaw Prus High School in Siedlce participated in the study. During the classes, students and teachers were first introduced to the application, its individual elements and features. Participants were then asked to create a specific algorithm, e.g. finding the greatest common divisor by Euclid's method or calculating the factorial of a non-negative integer. After more than a dozen minutes, everyone demonstrated the flowcharts they had made and shared their first impressions in the survey. The online Google Forms tool was used for its implementation. The questions were in the form of a linear scale 1-6 (the same as the grading scale for this level of education), single choice and short answer text. Among these questions, the following four can be distinguished:

- Which of the functionalities do you consider the most important? Please select 1-4 items.
- How do you rate the ability to export the flowchart to code in six different programming languages on a scale of 1-6?
- Do you think AlgoPoint can help you learn programming and algorithmics? Please tick Yes, No or Don't know.
- Overall, how do you rate the entire application on a scale of 1-6?

As shown in the bar chart in Fig. 8, the majority of students said, that AlgoPoint would help them learn programming and algorithmics (81%). AlgoPoint was appreciated by 14% of respondents, but they were not entirely sure that the program would fully meet all their expectations regarding the broadening of programming horizons. Only 5% of people decided that this tool would not be useful during computer science classes. Additionally, to verify the usefulness of individual features of the application, we listed a few of them and checked how the respondents assessed them. They had to choose from one to four features which, according to them, are the most important or useful. In the column chart in Fig. 9, there are the five most frequently chosen functionalities, broken down by the level of interest in IT (high - left black bar; medium - red middle bar; low - right green bar). As can be seen, formatting the appearance of the flowchart elements and its export to PDF were not highly appreciated. On the other hand, the functionalities referring to algorithm execution testing and source code generation, being the most crucial elements of the AlgoPoint, were chosen by the vast majority of students. It is also worth noting a specific dependence among various groups of interests in computer science. Passionate students mainly chose to generate the source code and test the algorithm. However, for respondents with a low level of interest in the field, the key functions turned out to be exporting to PDF and modifying the visual aspects of the flowchart. Moreover, for the remaining group of students, the answers are similar to those given by IT enthusiasts. Therefore, it can be stated that the lower the level of interest in computer science a student has, the more often less vital functions of the application from the programming perspective are used.

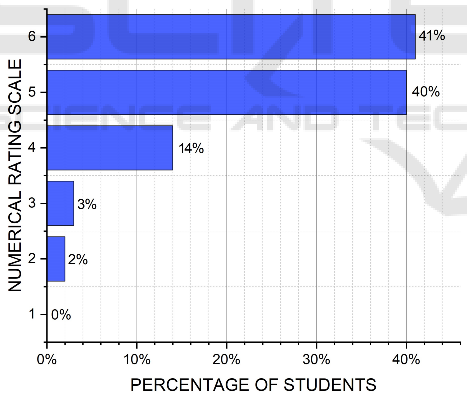


Figure 8: Percentage of students by their overall application rating on a scale of 1-6.

Students were also asked to express their own suggestions and opinions about AlgoPoint. The number of AlgoPoint performance concerns was negligible. An interesting suggestion was to create a built-in tutorial. Such a facility would allow users to check how a given functionality works at any time. As for the positive feedback, apart from the previously discussed functions, the respondents liked the dark mode and particular elements of the user interface, which they associated with popular applications (which can be considered helpful in navigating the program). Furthermore, the teachers found the program well-made and universal enough to be used as a didactic aid in programming and algorithmics classes for students without the extended computer science profile.

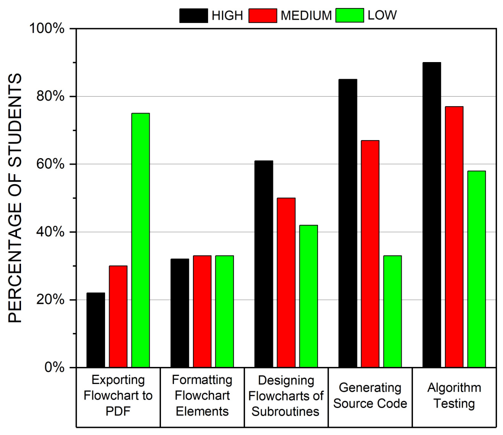


Figure 9: Assessment of the usefulness of individual functionalities of the application, broken down by interest in computer science among students.

Based on the analysis above, it should be noticed that the majority of respondents assessed the AlgoPoint application very positively, regardless of the class profile and the level of interest in IT. Among the most repeatedly mentioned advantages of the application are functionalities such as checking the execution of the designed algorithm, along with the ability to generate the source code in different programming languages. Taking into consideration all the collected opinions, it can be concluded that our software might be seamlessly deployed in the process of teaching computer science at the high school level.

5 CONCLUSIONS

The paper presents an original flowchart editor named AlgoPoint, which was implemented as a support tool for high school students learning programming fundamentals. Students practice computational thinking and problem solving by building algorithmic solutions with the use of predefined blocks and conections. More advanced functionalities, such as text code generation, function definitions or interactive testing, are also included allowing more advanced users to further develop their skills. The presentation and testing of the application by high school students under the supervision of teachers positively verified the usefulness of the AlgoPoint as a universal didactic aid in computer science classes. Moreover, as much as 81% of students claimed that AlgoPoint would help them learn programming and algorithmics. Further development of AlgoPoint would include several user-suggested changes and the introduction of new functionalities such as support for extensions, turtle graphics, and cooperation mode. We would then conduct more detailed live tests and analysis to compare the usability of AlgoPoint against other popular programs (like Flowgorithm) and how it translates to students performance in coding.

REFERENCES

Ben-Ari, M. (2012). *Mathematical logic for computer science*. Springer Science & Business Media.

Bione, J., Miceli, P., Sanz, C. V., and Artola, V. (2017). Astrocode: a serious game for the development of computational thinking skills. In *9th Int. Conf. on Education and New Learning Technologies (Barcelona)*.

Cardelli, L. and Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys (CSUR)*, 17(4):471–523.

Chapin, N. (1970). Flowcharting with the ansi standard: A tutorial. *ACM Comp. Surveys (CSUR)*, 2(2):119–146.

Cook, D. D. et al. (2015). Flowgorithm: Principles for teaching introductory programming using flowcharts. In *Proc. American Society of Engineering Education Pacific Southwest Conf.(ASEE/PSW)*, pages 158–167.

Edgcomb, A. D., Vahid, F., and Lysecky, R. (2019). Coral: An ultra-simple language for learning to program. In *2019 ASEE Annual Conference & Exposition*.

Gaertner, M. (2009). Lazarus for cross-platform development. *Linux Journal 2009*, 3(185).

Henderson, P. B., Cortina, T. J., Hazzan, O., and Wing, J. M. (2007). Computational thinking. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 195–196.

Janpla, S. and Piriyasurawong, P. (2018). The development of problem-based learning and concept mapping using a block-based programming model to enhance the programming competency of undergraduate students in computer science. *TEM Journal*, 7(4):708.

KAYA, K. Y. and YILDIZ, İ. (2019). Comparing three free to use visual programming environments for novice programmers. *Kastamonu Eğitim Dergisi*, 27(6):2701–2712.

Koracharkornradt, C. (2017). Tuk tuk: A block-based programming game. In *Proceedings of the 2017 Conf. on Interaction Design and Children*, pages 725–728.

Kourouma, M. K. (2016). Capabilities and features of raptor, visual logic, and flowgorithm for program logic and design.

Kraleva, R., Kralev, V., and Kostadinova, D. (2019). A methodology for the analysis of block-based programming languages appropriate for children. *Journal of Computing Science and Engineering*, 13(1):1–10.

Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M. (2010). Learning computer science concepts with scratch. In *Proceedings of the Sixth ACM int. workshop on Computing Education research*, pages 69–76.

Mishra, P. and Schrawankar, U. (2014). Desktop operations procreate ease for visually impaired. In *2014 IEEE Int. Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 222–227. IEEE.

Nassi, I. and Shneiderman, B. (1973). Flowchart techniques for structured programming. *ACM Sigplan Notices*, 8(8).

Person, R. (2013). *Getting Started with the Lazarus IDE*. Packt Publishing Ltd.

Rababaah, A. R. (2020). A new simple programming language for education. In *2020 15th International Conference on Computer Science & Education (ICCSE)*, pages 145–149. IEEE.

Ribić, S. and Beganović, M. (2013). Cross compilation under lazarus ide. In *2013 21st Telecommunications Forum Telfor (TELFOR)*, pages 1007–1010. IEEE.

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

Romeo, M., Lepage, A., and Lille, B. (2017). Computational thinking development through creative programming in higher education. *Int. Journal of Educational Technology in Higher Education*, 14(1).

Rossheim, R. J. (1963). Report on proposed american standard flowchart symbols for information processing. *Communications of the ACM*, 6(10):599–604.

Rugelj, J. and Lapina, M. (2019). Game design based learning of programming. *Proc. Int. SLET, CEUR Workshop*.

Strong, G. and North, B. (2021). Pytch — an environment for bridging block and text programming styles (work in progress). In *The 16th Workshop in Primary and Secondary Computing Education*, pages 1–4.

Strout, J. (2021). Miniscript: A new language for computer programming education. In *2021 6th Int. STEM Education Conference (iSTEM-Ed)*, pages 1–4. IEEE.

Sysło, M. M. and Kwiatkowska, A. B. (2015). Introducing a new computer science curriculum for all school levels in poland. In *Int. conference on informatics in Schools*, pages 141–154. Springer.

Weintrop, D. (2021). The role of block-based programming in computer science education. *Understanding computing education*, 1:71–78.

Werner, L., Campe, S., and Denner, J. (2012). Children learning computer science concepts via alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 427–432.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., and Korb, J. T. (2011). Introducing computational thinking in educational courses. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 465–470.

Yu, Z. and Xiong, Z. (2018). Comparative analyses for the performance of rational rose and visio in software engineering teaching. In *Journal of Physics: Conference Series*, volume 1087, page 062041. IOP Publishing.