Placing Event-Action-based Visual Programming in the Process of Computer Science Education

Gábor Csapó

University of Debrecen, Faculty of Informatics, Kassai út 26, 4028 Debrecen, Hungary, csapo.gabor@inf.unideb.hu

Abstract: Based on research results and experience, students who finish K-12 education lack the necessary computational thinking skills that they would need to continue their studies effectively in the field of computer sciences. Our goal was to examine the currently used methods and programming languages in K-12 education and to find and present an alternative approach. Using visual programming environments in education to develop students' computational thinking and algorithmic skills is a widespread practice in K-12 education. These environments mostly provide simplified versions of "real" programming languages. In this paper, we present event-action-based visual programming, as an alternative to today's most frequently used methods, which do not restrict the students' development ability to simplified and basic applications while retaining the advantages of visual languages. We organized four workshops in which we presented this programming approach to four distinct groups involved in education. The participants were guided to develop a multiplatform mobile application using Construct 2 event-action-based visual programming. At the end of the sessions we collected data in the form of group interviews and questionnaires on the possibilities of including event-action based visual programming in computer science education. Based on the results, the participants found the method suitable for beginner programmers to help them lay the foundations for more complex, text-based programming languages and to develop a positive attitude towards programming.

Keywords: visual programming; algorithmic skills; computational thinking; computer science education; Construct 2

1 Introduction

One of the most important aspects and goals of computer science education is to develop students' computational thinking and algorithmic skills [1]. These skills are not just important in the context of computer science, but in everyday life as well because they provide the basics for slow thinking approaches [2], [3], [4] used to solve novel problems in various situations. Based on the results of an international research project [5], first-year undergraduate students lack the

required level of these skills to continue their studies efficiently. Therefore, it is not uncommon that these students need reiteration and sometimes complete re-learning of basic topics in order to develop an adequate level of computational thinking on which their following courses can be built.

1.1 Developing Computational Thinking

Computational thinking and algorithmic skills are considered most important in the field of programming and software development. Some educators focus on developing these skills through various forms of programming environments. The text-based programming languages used in education range from modern object-oriented languages (for example: Java, C++, C#) to older procedural examples such as Pascal, which are not widely used and can only be found rarely in the contemporary IT industry. These languages present students with steep learning curves and tend to confuse them with syntax and instructions which seem complex for beginner programmers [6] [7]. Therefore, the teaching efficiency of the topic in K-12 education cannot completely fulfill the requirements stated in the curriculum [8]. To solve this problem, numerous EPLs (Educational Programming Languages) have been developed in order to help students understand the basic concepts of programming logic, often through visual programming approaches. While using these languages to teach programming in education has been a widespread practice over recent years, they have not solved the problem that students who complete K-12 education cannot solve basic tasks that require algorithmic skills [5]. We provide an overview of the most commonly used EPLs in the following section.

In an ideal educational environment, the development of computational thinking is not only focused on the programming topic, but rather integrated into all topics of ICT (Information and Communication Technology) education, such as Sprego and ERM (Spreadsheet Lego and Error Recognition Model), respectively [9], [10], [11], [12], [13], [14], [15]. Teaching birotical software receives great emphasis in the Hungarian K-12 curriculum [8], but most educators miss the opportunity to develop students' algorithmic skills by using such software [16]. Either by assuming that these skills can only be developed in the programming topic, or by being unaware of methodologies that focus on this area of computer science education [17].

1.2 Goals of the Present Work

Following on from the papers by Soloway and Ben-Ari, and on the principles of the Sprego and ERM methodologies, our goal was to analyze the EPLs currently used in computer science education and present an alternative approach for teaching programming which has the potential to increase the efficiency of the

learning processes and the development of algorithmic skills though event-action-based visual programming.

2 Visual Programming

Visual programming uses graphical elements to represent and build up algorithms while focusing on the underlying logic of the application under development. The developers combine various pre-defined elements to construct the visual code. These building blocks differ from each other in terms of their purpose and functionality and revolve around the design of the visual language. While this approach to software development is not new, as a result of continuous progression of visual languages, today various IDEs (Integrated Development Environments) integrate distinct forms of visual programming to aid development processes. These languages help developers construct and define the logic behind their applications without the need to focus on the syntactical details of programming languages. The novel versions of such languages come with easy to understand presentations and self-explanatory instruction sets. These advantages over text-based programming languages make visual programming a compelling tool in the educational field as well. However, it is worth mentioning that while these languages tend to provide an easier and more rapid development experience, they are usually limited in terms of functionality. With some general-purpose exceptions on the market, visual programming languages do not hold the same potential as text-based languages, considering the complexity of the logic they are capable of handling. For this reason, experienced programmers do not favor visual programming throughout the whole development cycle of their applications, but rather use them as supplementary tools. In some cases, the visual programming IDEs offer the opportunity to include custom text-based code alongside the visual elements to customize the projects and to break free of any possible limitations the environments might pose on the developer.

The various forms of visual programming languages create a divergent market, as the user cannot find two identical implementations of this programming method. Furthermore, these implementations are not compatible with each other - the visual code created in an IDE cannot be transferred directly to a different environment as is possible with text-based languages. After analyzing various popular occurrences of visual programming languages on the market, we defined 4 categories into which these languages can be classified [12]. Note that an ideal visual programming language incorporates more than one of the following categories.

- Behavior-based languages: Behaviors are pre-coded scripts that the developers can implement into their projects with minimal effort. The goal of these scripts is to speed up the development process rapidly by removing the need for

developers to define basic functionalities. On the other hand, the behaviors are limited to basic operations with few customization options and are not suitable for constructing complex, custom algorithms. For this reason, this form of visual programming rarely stands alone and is rather accompanied by one of the following, more flexible approaches. The Construct 2 [18], Construct 3 [19] and GameMaker: Studio [20] development environments all include behaviors.

- Event-action-based languages: As the name of this category implies, the developers define events in the visual code and assign actions to them which run when the events' conditions are met. The complexity of the logic that can be constructed with this method depends heavily on the available, pre-defined event and action arrays. It provides an easy to understand and transparent visual code, even for more complex projects. From all the visual programming categories, the event-action approach has the smallest professional market share and only a few IDEs can be found which integrate this method: Construct 2 [18], Construct 3 [19], GDevelop [21] and Clickteam Fusion [22]. In education, only the Kodu Game Lab [23] and the Lego Mindstorms [24] environments are known to be based on this method.
- Block-based languages: These languages resemble the syntax of text-based programming and provide building blocks with traditional programming elements to construct the code visually. The developers combine these elements by simple drag and drop means. This form of visual coding is considered a general-purpose approach and rarely limits the user in terms of logical complexity. However, due to the fact that they are based on text-based languages, beginner programmers might find it difficult to start learning with block based visual languages. The Stencyl [25] IDE, the Scratch [26] and Alice [27] EPLs all focus on this approach.
- Node-based languages: The last category of visual languages provides a flowchart or mind map-like experience for the developers, who define nodes from a pre-constructed array and relate them together using various types of connections. This form poses the least limitations on developers and is usually accompanied with the ability to create custom nodes using text-based languages. Working with complex projects using this approach can prove to be difficult because defining complex logic on a flowchart can easily result in an unreadable visual code. Despite this, this method has the largest professional market share as more and more environments integrate it into their code editors. For example, the popular Unreal Engine 4 [28] calls this form of programming "blueprints" and its applications range from creating application logic to designing materials or animations, as well. In 2017 the Godot Engine [29] and the GameMaker Studio [20] also implemented node-based programming. In the current state of our research, we do not know of any EPLs which are based on this type of visual code and would be suitable for low-complexity software development.

2.1 Visual Programming EPLs

Using visual programming languages and environments designed specifically for educational usage to teach students the fundamentals of programming and to develop their computational thinking and algorithmic skills is an accepted practice in contemporary education. While tertiary education focuses primarily on text-based languages, we can find various EPLs that are used in K-12 and higher education. The following environments were designed for beginner programmers and just as with visual programming languages, these development environments also vary in their approach to the topic and in the form of programming they implement.

2.1.1 Alice

Alice [27] is a block-based visual programming environment designed for education. With its help, students can create interactive 3D animations and by design, it serves as an introductory language for object-oriented programming [30]. While it also includes events in its workflow, it is not to be confused with the event-action based visual programming approach as the majority of the code follows the block-based principle. Alice is used in a wide range of educational institutes in secondary and tertiary education, mainly as an introduction to programming. Based on student feedback, measurements and educators' observations, students find the 3D animations made with Alice interesting and entertaining, while the workflow of the software is useful and easy to understand and to get into for beginners [31].

2.1.2 Lego Mindstorms

The idea of building robots and programming them in education is an approach popularized by Lego Mindstorms [24] although it was not the first endeavor in this area. The kits available come with a variety of programmable parts that can be built into the robots, for example motors, sensors, and lights. The students build algorithms to pass instructions to the robots and control them in different situations. This provides several opportunities to teach programming concepts and gives real-world feedback to students. The official visual programming environment available uses a special case of event-action-based visual programming and is only recommended for beginners based on its instruction array. While for advanced users, the robots can also be controlled with text-based languages (C++ and Java), in this paper we only focus on the visual programming aspect of this approach. Lego Mindstorms is used not only in public education, but also in tertiary education [32]. However, despite the advantages and visually engaging experience of this approach, according to research conducted at Hanover College, students learning programming with Mindstorms did not achieve better results than learners who used text-based language IDEs. Also, robot programming seems to offer no additional motivational drive to encourage

students to continue their studies at higher levels in tertiary education [33]. If the goal is to make every student learn to build algorithms, then every learner must be provided with the opportunity to write code, therefore, they must work in small groups. Although the Mindstorms kits are not the priciest, they are still expensive for educational purposes, especially if every student needs a kit. A large number of K-12 institutes do not have the resources to buy even one robot. Another potential problem with teaching with Lego Mindstorms is that the robots have to be built before the programming part of the lessons can begin. This process needs to be either part of the classes with the accompanying sacrifice of time, or teachers have to build all student robots outside of teaching time as additional work. Educators who decide to use Lego Mindstorms to teach programming should keep in mind that while its visual language is easy to understand, it focuses on the concept of controlling robots and is not a general-purpose method of programming. Beyond these concerns, most students in the target groups do not have the necessary background knowledge in physics for building and using the robots.

2.1.3 Scratch

Scratch [26] is a well-known EPL at all levels of education. It was developed by MIT Media Lab Lifelong Kindergarten Group and it uses a block-based visual programming language to construct the logic of interactive stories, games, and animations. It was designed for beginner programmers, even for young (8-16 years old) students who could not imagine themselves as developers before trying out Scratch [34]. It has several design features to aid educational processes; for instance, completed projects can be shared with the Scratch community and students can open every shared work to view its source code. The visual programming approach of this environment even allows learners to construct highly complex projects. The Theme Park God [35] is a prominent example of what can be achieved with this EPL. Note that while building composite projects is possible with Scratch, the resulting source code can be difficult to read and see through. Although students found this environment easy to use, several studies point out potential problems regarding its workflow and effectiveness. In a primary school, learning programming with Scratch did not result in increased problem-solving skills for 5th grade students as opposed to learning with traditional methods [36]. Teaching computer science concepts is only possible with this EPL if it is paired with an adequate, purposefully designed educational context, because learners tend to follow bad programming practices while developing their projects. Instead of focusing on the algorithms, students usually drag and drop into their codes all the blocks they think are necessary to solve the problem. This can result in bricolage projects, instead of a well-analyzed approach to a problem. Overly deconstructed elements without logical coherency are also common in students' work [37]. Scratch does not reinitialize the value of the variables between project executions by default. This leads students to mistaken initialization practices and makes knowledge transfer to future environments

troublesome [38]. While this visual language was designed for education, its popularity inspired developers who created a visual programming environment aimed for production called Stencyl [25], which uses the same programming approach as Scratch.

2.1.4 Kodu Game Lab

The developers of Kodu Game Lab [23] reacted to the increasing popularity of video games and created an environment in which students have the ability to build simple visually appealing 3D interactive games and stories that are rich in multimedia elements and are able to stimulate multiple sensory organs simultaneously. Kodu uses a simplified event-action based visual programming approach. It targets young students, and by its design it promotes learning through independent exploration [39]. This visual language is based on a when-do structure, where students first have to define the events with conditions (when) and then the actions (do) which are run when the corresponding event's conditions are met. This EPL is suitable to teach computer science concepts with its language structure and can be used as a launching point for learning text-based languages [40]. Interestingly, Kodu Game Lab was first developed for the Xbox360 console and was later ported to Windows operating systems, which makes it exceptional in terms of supporting a popular gaming platform.

2.1.5 Summarizing EPLs

The visual EPLs described above are used in practice to teach programming at various levels of computer science education. Most of them focus on games or game-like projects being aware of their effects on educational processes through positive emotions, not exclusively restricted to ICT education [41] [42] [43] [44]. With these environments, students can create their own, visually more appealing projects compared to traditional texts-based IDEs. Similar attempts for content-development-focused approaches can be observed in other fields of education [45] [46] [47] [48]. Content creation can also be suitable in the area of self-learning; however, teachers favoring this approach must take into consideration the difficulty level of such tasks they pose towards students [49].

The educators (just as with every educational tool) have to be well informed and cautious about the limitations of these environments, what can be achieved with them and in what educational contexts they should be used to make developing computational thinking and algorithmic skills more effective. Based on our analyses and feedback from students, learners usually find these environments childish, which make working with them with higher age groups difficult. Moreover, these programming environments were all designed for educational purposes and therefore they are not suitable for use outside of the educational context. Following on, we also have to consider the integration of BYOD (Bring Your Own Device) approaches. Students trying out or developing their own projects on their own devices would be extensively motivating for them [50].

However, we must be aware that most EPLs do not provide functionality to achieve this goal regardless of its potentials. It is important to emphasize that this is also true for other widespread EPLs, which are not necessarily based on visual programming, such as Logo [51]. In the following section we introduce an event-action based visual programming environment that was designed with software development in mind, but has all the elements which are required to make it a compelling candidate for educational usage.

2.2 Construct 2

Developed by Scirra, Construct 2 [18] is an integrated development environment which implements the event-action and behavior based forms of visual programming languages. This environment is designed to develop video games and multimedia web applications, and therefore uses a general-purpose approach in terms of its language and instruction set. After examining several visual programming environments designed primarily for software development and not only for education, we chose Construct 2. Mainly, because its implementation of event-action-based visual programming poses few limitations in terms of possible logic complexity and diversity, and because students can develop interactive projects during the first lessons effortlessly. Furthermore, the design of the environment supports learning through experience and offers convenient help tools. Despite the English language of the user interface, we found it simple to use and, based on our observations, after a few translational explanations, Hungarian high-school students had no problem navigating and using the features of Construct 2.

The environment is based on an in-house developed lightweight HTML5 2D engine and primarily aims for the web platform, with other options available using wrappers. The developers implemented various optimizations in their engine to make the applications developed with it run as efficiently as possible, including optimized GPU (Graphics Processing Unit) draw calls, and optimized collision checks.

2.2.1 The Development Workflow in Construct 2

The interface of Construct 2 is separated into 3 columns. The most dominant central area is the workspace where the majority of the development process takes place. This is where the user designs the graphical layout of the project, as well as defines the visual code on different tabs. On the left side of the interface, the properties of the selected element are listed. In the right column, the project tree is displayed, similar to what in-service developers use in various IDEs focusing on text-based programming languages.

The main building blocks of the projects are the objects; these are the elements which define the usable instruction array (conditions and actions). The System

object is present in every application and handles various system related tasks (for example, managing variables and creating loops). It is important to distinguish local and global objects: while local objects are only available and visible at the parts of the project to which they were added, global objects can be accessed throughout the whole application. Objects cover various functionalities of the underlying engine, such as displaying an image, playing an audio file or processing user inputs.

Everything the end-user sees is placed on layouts. These elements can be interpreted as canvases on which the objects are drawn. Layouts possess all the required functionality to construct visually complex applications, as managing different layers with their own changeable properties make them similar in this field to raster graphics editor applications and students can build upon the knowledge they learned in that topic of computer science education. It is worth mentioning that similarly to other game engines on the market, Construct 2 allows objects to be placed on the layouts outside of their boundaries.

Behaviors (as described in Section 2) are pre-written scripts with a few customization options to allow rapid implementation of commonly used functionalities (for instance 8-directional movement or applying physics to an element). Behaviors have to be attached to objects in order to access their functions and while they provide an easily understandable and swift option to make interactable objects, they are not suitable for developing custom algorithms. However, using behaviors in education can improve student motivation because learners receive spectacular feedback on their work instantly. Note that setting behaviors to objects will expand the available instruction array with the conditions and actions of behaviors.

Figure 1
An example of the event-action based visual code of Construct 2

The event-action based visual programming is used to create the logic of the projects on event-sheets in the work-space area. When the developer creates an event, referencing an already existing object, a condition first has to be selected from the available array, and then the parameters of the condition have to be set. When the first condition is defined, an event block is automatically created with

the condition in it. The next step is to attach actions to this block using the same approach as is used with conditions. The completed event block lists the added condition and actions with object references in an easy to read, natural text format with highlights of the defined parameters. For instance, on Figure 1, the keyboard is the object, the first event has a condition which requires a button to be set to monitor its state when it is pressed down. This form of event-action based visual programming, provides the developers with several options to create high-complexity algorithms (for example: AND or OR logical connections between multiple conditions, sub-events, embedding event-sheets into each other).

While it is obvious that the defined events and actions run in a sequential order and conditions inside an event block are realizing selection as usual, iterations can be confusing for developers who only use text-based programming languages. Construct 2 has several hidden functionalities that make the development progress easier and smoother, but some can potentially interfere with the education of programming concepts. For instance, the third event block in Figure 1 checks for collisions between the player and enemy objects. If the developer places multiple instances of the same object on the layout, the event block watches all of them automatically, which requires a loop using text-based languages. Furthermore, the engine iterates events periodically in a loop to check for the conditions defined at each rendered frame. These particularities of this environment require novel methods for introducing loops, because the educators have to design specific cases in which using the loop events included (for, for-each instance of an object with ordered option, repeat N times, and while) are required.

Developers have the option to include three types of variables, based on their declaration location: global, local, and instance. The latter is associated with an object and follows the object-oriented paradigm, just as managing multiple instances of the same object. Functions are also available in the environment, by using the Function object type which manages defined functions, parameters, calls, and return values through events and actions.

The environment includes a debugger functionality to help developers monitor various aspects of their projects during runtime, and receive performance measurements on used resources. While some of the features of the debugger are not available in the free version of Construct 2, it is a powerful tool which could be used to teach student-project analysis and code debugging.

2.2.2 Supporting Materials for the Educational Use of Construct 2

The developers of this environment and the user community (with in-service teachers included) created numerous resources to help beginner programmers understand the fundamentals of the workflow of the software and to aid its integration into educational processes. The official manual [52] explains the individual elements of the environment in an easy to understand composition with a logical structuring in compliance with the design of the software. The tutorials

listed on the official website [53] are primarily community created materials that cover specific problems and their solution over a wide range of topics. Students can find resources to help grasp the basic development process of Construct 2, but they are provided with tutorials that cover more advanced, platform-specific topics or optimization techniques as well. It is important to note that while these user-created resources are available in several languages, at the time of our research no Hungarian tutorials can be accessed on the website. For those users who prefer learning from video materials instead of written guides, given the popularity of the environment amongst developers, various educational videos are also available. For instance, the Construct 2 Academy channel on YouTube [54] offers the opportunity to learn by creating different types of projects. Professional online courses [55] created for Construct 2 are also an option that further widens the range of supporting materials. The official community forums [56] present a place for English speaking students to post their questions and receive help directly from either a more experienced user or from the developers of the environment. Furthermore, the forums include a section for educational use of Construct 2 where teachers and students can share their experiences regarding the software and provide help and advice for each other. While it is not essential in the context of education, developers who find the functionality of the core environment limiting have the option to install third-party add-ons which are shared in the appropriate section of the forums. Computer science teachers experienced in JavaScript can also create their own add-ons by using the official SDK [57].

Similarly to the service Scratch provides, students have the option to share their completed works on the Scirra Arcade website online [58] for free. This service provides opportunities for learners to optionally share the source code of their projects, as well, and to gather feedback in the form of ratings or comment-based discussions. Detailed statistics are displayed for each shared project to help the developers analyze user traffic, play times, and downloads based on locations. While Scirra Arcade might not be the ideal platform for deploying completed commercial projects, it is suitable for educational usage for students and teachers alike. It allows them to share their prototypes or simple applications with easy to integrate online high-score management.

The free version of Construct 2 comes with a variety of limitations compared to paid licenses. The most notable restrictions are the limited number of event blocks (100), layers (4), no object grouping options or sub-folder creation in the project tree, limited export platforms (only web applications, including publishing to the Scirra Arcade are allowed) and commercial usage is forbidden. It is important to note that the developers of this environment allow educational usage of the free version and, based on our observations and experiences; it can completely cover the requirements of the Hungarian curriculum [8]. For those developers, companies and institutions who find the restrictions of the free version too limiting for their needs, various paid license options are available which all unlock the full potential of Construct 2.

2.2.3 Developing Projects in the Web Browser

While in this paper we focus on the educational possibilities of Construct 2, it is important to note that its successor, Construct 3 has been released which includes further potential features for education. Alongside the new functionalities to aid the smooth development principles which its predecessor embodies, the most notable addition is that the environment runs completely in a web browser as a PWA (Progressive Web App). While managing to keep all functionalities of an integrated development environment with a responsible interface, Construct 3 also opens new possibilities in terms of supported platforms. Every device that has a modern web browser installed is capable of running the environment without the need for additional installation. It is compatible with Construct 2 projects which do not rely on third party add-ons, so students can effortlessly import and convert their work. One of the additions to the new version we would like to point out is the option to translate the user interface into several languages. Consequently, bridging language gaps is now possible for students who might have problems understanding the English interface. While the free version of Construct 3 can also be used in education, all of the licensing options have switched to a subscription based model.

2.3 Visual Programming within the Frame of CogInfoCom

CogInfoCom emphasizes a systematic viewpoint on how modern infocommunication tools can develop synchronously with the cognitive processes of the users [59] [60] [61] [62]. In our current work we focus on the software tools within the scope of the mathability sub-field of CogInfoCom [3] [4] [63].

Using programming with high mathability problem solving aids the educational and cognitive processes with the development of logical reasoning and sequencing skills and abilities [63] [64] [65] [4] [15]. Furthermore, using visual programming technologies also develops the students' spatial visualization abilities [66]. Construct with our concept-based methodology [4] offers the advantages of high mathability visual programming methods in addition with the possibility to extend the capabilities of the human brain through interactive 3D educational environments. Because Construct 3 is built solely on web technologies, similar educational spaces can be created such as the Sprego virtual collaboration space presented in our prior work [67] in MaxWhere [68] [69] [70] [71] [72] [73]. Consequently, developing spaces for teaching and learning with Construct 3 is a compelling future research project. We have to emphasize that our methodology focuses on the core of high mathability product-creation with the use of existing tools [3].

3 Event-Action-based Visual Programming Workshops

In order to accomplish our goals (Section 1.2), four workshops were held during the 2016/2017 academic year, in which the participants created a simple interactive mobile game in Construct 2. We wanted them to experience the workflow of the event-action based visual programming first hand, so during the workshops we provided the conditions necessary for individual work alongside lecturer guidance with a presentation. Depending on the participants' work speed, each workshop lasted between 2 and 4 hours, and finished when the project was completed and was tested by everyone. Because we set out to collect feedback and experience regarding the visual programming approach of Construct 2, and about its possibilities for integration into classes, we targeted four distinct groups with our workshops which are all involved in computer science education at different levels.

3.1 The Characteristics of the Target Groups

The first workshop was organized for high-school students (N = 2, the other students who registered did not show up) at a local institute in Debrecen, Hungary. Choosing this school was a compelling option because it focuses mainly on humanities and its primary profile is drama education. Therefore, based on our prior experience with several classes, the students do not view computer science as an important subject and only a few of them have experience with programming or software development outside the classes based on the curriculum [8]. Our second target group was undergraduate students (N = 14) at the University of Debrecen Faculty of Informatics. We held the workshop during the Professional Days event organized in each semester. The students in this group are taking computer science courses; therefore, they have an overview and experience of software development and programming languages. While there are four majors available at the institute each specialized for a different area of computer sciences, we did not filter the students by the courses they had taken, or by their terms. The third experiment group was pre-service teachers of informatics (N = 5) studying at the University of Debrecen Faculty of Informatics. Developing the workshop project with these participants using Construct 2 was an obvious choice as these students had tried various forms of educational programming languages during their studies and could provide feedback on the educational possibilities of the event-action based visual programming language from a contemporary perspective. While all the students from the chosen group participated in the workshop, their small number can be justified by the fact that only a few students regularly enroll for informatics (computer science) teacher education in the institute. We targeted in-service computer science teachers (N = 19) with our latest workshop at a postgraduate training organized in Zamárdi, Hungary. We counted on the participants' field experience and the wide range of

their knowledge of teaching programming for our data collection. Similarly to the previous pre-service computer science teacher group, the participants worked with several EPLs, but they also view these tools through their long involvement in the educational processes. In summary there were 40 participants at all workshops.

During our workshops we collected data regarding the possible options for integrating the selected visual programming method and environment into computer science education. We presented questionnaires at the end of each session to inquire about the previous programming (including visual programming) and software development experiences of all participants and the previous programming languages and methods they learned in the student groups. After the questionnaires, we conducted group interviews in which we collected data and feedback about the workflow of Construct 2, the project and the software development potential of the environment, with additional information about the possible integration and placement of the event-action based visual programming method in computer science education in tandem with currently applied approaches. We extended the collected data with our observations on the participants and their reaction to the environment and its workflow during the development process.

3.2 The Composition of the Developed Project

In advance of the workshops, we designed a 2D, interactive, simple, mobile game project targeting the HTML5 platform, supported by the free edition of Construct 2. Besides the goals we described above (Section 1.2), we also wanted the participants to experience that simple applications can be developed with this environment in only a few hours.

While the logic of the project and the accompanying sound files were our design, the assets we used for its visual appearance originated from a package whose license we purchased [74]. The created project also served as an example of multi-platform application development because it was designed to be playable both on desktop operating systems and on touch screen devices. The concept of the project was the following: fish appear at random generated Y coordinates at the left side of the game layout (outside of the visible area) and they swim towards the right side of the screen. The user's goal is to catch the fish by touching or clicking on them. The game counts the captured fish and displays this number for user feedback. While this game does not terminate and as such plays endlessly, it was an appropriate way for us to include and present various programming concepts in a short time. Because each workshop was limited in terms of available time depending on the hosting event and environment, we focused on developing the core functionalities of the project during the workshops. Therefore, functionalities we considered supplementary (for example the game menu, creating particle effects and using the vibrate function of smart devices) were left out as required, in order to provide time for implementing user ideas and functions.

Although due to the time restrictions, the project operated with simple algorithms (Figure 2), it allowed us to provide an example for the participants of how the following programming concepts can be implemented into a project with this environment: declaring and initializing global variables, setting variable values, displaying and changing strings on the screen during runtime, playing sound files, creating and destroying object instances with code, defining conditions on stored values, generating random numbers in a pre-defined value range, listening to user touch inputs, and using platform specific functionalities (in this case, the vibration function of smart devices).

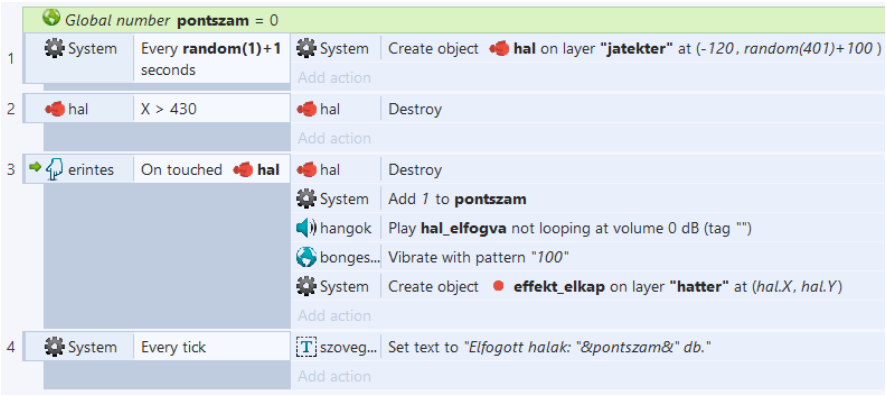

Figure 2

The visual source code of the core functionalities of the workshop project

Note, that in this chapter we do not describe the basic and self-explanatory functionalities of the software that were necessary to create the project (for instance adding new objects to a layout, or setting behaviors). We also touched briefly on the optimization topic of software development by showing the participants the debugger and creating the 2nd event block seen in Figure 2 to avoid fish object instances that left the visible game area filling up memory.

4 Summary

In this section we highlight the results of our research in a summarized form. Based on the data we received from the questionnaires relating to the participants' prior knowledge and experience regarding text-based or visual programming languages, only four undergraduate students had no programming knowledge before the workshop. The participants who learned text-based programming languages divide equally between procedural (N = 26) and object-oriented (N = 26) languages. While the undergraduate students had more experience with procedural languages, the in-service teachers of informatics tend to work with object-oriented programming practices (whether for educational or personal

purposes was not specified in the inquiry). There was no difference between the proportion of the two types of programming languages in the high-school students' and in the pre-service teachers' groups. It is worth mentioning that several participants (N = 11) listed web, data-management and special-case programming languages beside the aforementioned two categories.

Regarding prior knowledge in visual programming, a considerable number (N = 19) of participants had no experience of this type of programming. Amongst those students and teachers who had used some form of visual programming beforehand, the Scratch [26] EPL was the most widespread (N = 16), and 8 other environments with high differentiation were listed. Interestingly, only 1 participant mentioned LabView [75], the well-known environment in tertiary education. As regards software development experience, 10 participants stated that they did not have any background in the area, while 28 claimed that they developed software on their own. Note that the questionnaire did not specify the scope of these applications. It is also important to point out that the participants responded to the majority of the questions with multiple answers and therefore some of the summarized results are above 100%. For instance, the summarized number of participants who had experience in procedural and object-oriented programming would indicate that there were 52 total responders. Furthermore, a varying number of uncategorizable answers were found for each question which were disregarded in the summarized results described above.

The group interviews were focused on the Construct 2 environment and its workflow. After the questionnaires, we interviewed the participants and recorded their answers. We also kept in mind to encourage all members of the groups to speak up and gave everyone the opportunity to express their opinion and feedback. To the question considering the overall experience of the environment, almost all of the groups responded positively, with some exceptions in the undergraduate students' group. These students found the progression slow in the development process, which can be explained by the time needed to understand the basic principles of the software.

As regards using the environment easily, and how difficult it is to learn the basics, the high-school students found the initial learning process troublesome, while the undergraduates stated that it was exceptionally easy to get into, and they made the comparison that Construct 2 feels like a toy. The pre-service teachers found the software easy to use, and in their opinion it would be much less complicated to start learning programming with this approach. However, they also pointed out that they would have liked to see the whole source code of the project as multiple event sheets on one page. The in-service teachers also learned the basics without complication, but highlighted that to complete the basic operations routine, this environment requires more practice and time than we had during the workshop.

Responding to the question about the difficulty level of coding the algorithms with event-action based visual programming, both the high-school and undergraduate

students answered that it was straightforward. The environment seemed a well-structured system and the participants claimed that programming in text-based languages is more complicated. The pre-service teachers of informatics provided more details, by stating that this form of programming should be used before any other programming methods and with this environment the coding part of the learning process can be hidden to help students understand the background processes of their projects. They also added that unlike text-based languages, Construct 2 provides spectacular feedback for the students. The in-service teachers saw this environment as an intermediate step in teaching the topic, because mastering the environment may be difficult for young learners and can be limited for students on more advanced programming levels. However, they saw it as a compelling option for project work that lasts over several weeks.

To the questions about whether this environment could be integrated into computer science education and whether students would be learning or teaching programming with its help, the high-school students', the pre-service and in-service teachers' groups responded positively. The pre-service teachers also stated that they would rather learn and teach programming using this method than text-based languages and that the knowledge gained from this visual programming approach can be utilized in different areas of computer science education. In contrast, the undergraduate students only saw the environment as a starting point to avoid creating negative experiences in beginner programmers. They would be ready to learn with Construct 2, but only if there were a different environment later on.

Based on our observations and experiences with the groups during the workshops, we found that the participants handled the environment with ease, and only a few technical questions emerged at the time of the development of the project. While following the presentation and guidance of the lecturer, the participants enjoyed working with Construct 2 and easily understood its workflow. Therefore, suggestions and new ideas emerged about further expanding the project or trying out new functionalities.

Conclusions

In this paper we presented the Construct 2 event-action-based visual programming environment and the workshops we held to introduce it to four groups involved in computer science education. The data we gathered from the questionnaires, group interviews and from our observations indicate that this form of visual programming has the potential to be integrated into the field of computer science education. Based on the information we received, we see high-school computer science classes and introductory programming courses in tertiary education as ideal affiliations. We view our results as a starting point for the next steps required to achieve this aim. Further work includes developing the methodology for teaching the programming topic with this environment in alignment with the requirements present in the Hungarian curriculum [8]. Because only two students

enrolled for our high-school workshop, we want to expand our data on how students at this level of education view and react to this environment. Therefore, testing our future methodology and this form of visual programming in classes is an important task. We also plan to conduct measurements on the effectiveness of this approach with control groups to obtain detailed results on its potential in comparison with the abilities of currently applied EPLs to develop computational thinking and algorithmic skills.

References

[1] J. M. Wing: Computational thinking, Communications of the ACM, 2006, 49(3), pp. 33-35

[2] D. Kahneman: "Thinking, Fast and Slow" Farrar, Straus and Giroux, New York, 2011

[3] P. Baranyi and A. Gilanyi: Mathability: emulating and enhancing human mathematical capabilities, 4th IEEE International Conference on Cognitive Infocommunications, 2013, pp. 555-558

[4] P. Biró and M. Csernoch: The mathability of computer problem solving approaches, 6th IEEE International Conference on Cognitive Infocommunications, 2015, pp. 111-114, DOI=http://doi.org/10.1109/CogInfoCom.2015.7390574

[5] M. Csernoch, P. Biró, J. Máth and K. Abari: Testing Algorithmic Skills in Traditional and Non-Traditional Programming Environments, Informatics in Education, 2015, 14(2), pp. 175-197, DOI=http://doi.org/10.15388/infedu.2015.11

[6] E. Soloway: Should we teach students to program?, Communications of the ACM, 1993, 36(10), pp. 21–24, DOI=http://doi.org/10.1145/163430.164061

[7] M. Ben-Ari: Non-myths about programming, Communications of the ACM, 2011, 54(7), pp. 35, DOI=http://doi.org/10.1145/1965724.1965738

[8] "Central curriculum framework for year 9-12 students", In Hungarian "Kerettanterv a gimnáziumok 9-12. évfolyama számára" Oktatáskutató és Fejlesztő Intézet. [Online] Available: http://kerettanterv.ofi.hu/03_melleklet_9-12/index_4_gimn.html. [Accessed: 09-Nov-2016]

[9] M. Csernoch: "Programming with Spreadsheet Functions: Sprego", In Hungarian: "Programozás táblázatkezelő függvényekkel – Sprego", Műszaki Könyvkiadó, Budapest, 2014

[10] P. Biró and M. Csernoch: Unplugged tools for building algorithms with Sprego, END2017, International Conference on Education and New Development, Lisbon, Portugal, 2017, in press

[11] P. Biró and M. Csernoch: Semi-unplugged tools for building algorithms with Sprego, International Conference on New Horizons in Education, Berlin, 2017

[12] G. Csapó and K. Sebestyén: Educational software for the Sprego method, International Conference on New Horizons in Education, Berlin, 2017

[13] M. Csernoch: Teaching word processing – the theory behind, Teaching Mathematics and Computer Science, 2009 (1), pp. 119-137

[14] M. Csernoch and P. Biró: Error Recognition Model: End-user Text Management, World Conference on Computers in Education (WCCE), Dublin, 2017

[15] M. Csernoch and E. Dani: Data-structure validator: an application of the HY-DE model, 8th CogInfoCom, Debrecen, 2017, pp. 197-202, ISBN: 978-1-5386-1264-4, IEEE

[16] M. Csernoch, P. Biró, K. Abari and J. Máth: Programming oriented spreadsheet functions, In Hungarian: Programozásorientált táblázatkezelői függvények, XIV. ONK: Oktatás és nevelés – gyakorlat és tudomány, Debrecen, 2014, pp. 463, ISBN:978-963-473-742-1

[17] R. Panko: The Cognitive Science of Spreadsheet Errors: Why Thinking is Bad, Proceedings of the 46th Hawaii International Conference on System Sciences, Maui, 2013

[18] "Create Games with Construct 2" Scirra. [Online] Available: https://www.scirra.com [Accessed: 21-Sep-2017]

[19] "Make Your Own Games - Construct.net" Scirra. [Online] Available: https://www.construct.net [Accessed: 21-Sep-2017]

[20] "Make 2D Games with GameMaker | YoYo Games" YoYo Games. [Online] Available: https://www.yoyogames.com [Accessed: 23-Sep-2017]

[21] "GDevelop - Create games without programming - Open source HTML5 and native game creator" F. Rival [Online] Available: http://compilgames.net. [Accessed: 22-Sep-2017]

[22] "Clickteam - Clickteam Fusion 2.5" Clickteam. [Online] Available: http://www.clickteam.com/clickteam-fusion-2-5 [Accessed: 28-Sep-2017]

[23] "Kodu | Home" Microsoft Research. [Online] Available: https://www.kodugamelab.com [Accessed: 04-Oct-2017]

[24] "Home - LEGO.com" Lego. [Online] Available: https://www.lego.com/en-gb/mindstorms?ignorereferer=true [Accessed: 03-Oct-2017]

[25] "Stencyl: Make iPhone, iPad, Android & Flash Games without code" Stencyl. [Online] Available: http://www.stencyl.com [Accessed: 10-Oct-2017]

[26] "Scratch - Imagine, Program, Share" Lifelong Kindergarten Group. [Online] Available: https://scratch.mit.edu [Accessed: 12-Aug-2017]

[27] "Alice - Tell Stories. Build Games. Learn to Program." Carnegie Mellon University. [Online] Available: https://www.alice.org [Accessed: 08-Oct-2017]

[28] "Game Engine Technology by Unreal" Epic Games. [Online] Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4 [Accessed: 10-Oct-2017]

[29] "Godot Engine - Free and open source 2D and 3D game engine" J. Linietsky and A. Manzur. [Online] Available: https://godotengine.org [Accessed: 10-Oct-2017]

[30] S. Fincher, S. Cooper, M. Kölling and J. Maloney: Comparing alice, greenfoot & scratch, Proceedings of the 41st ACM technical symposium on Computer science education, 2010, pp. 192-193, ACM

[31] E. R. Sykes: Determining the effectiveness of the 3D Alice programming environment at the computer science I level, Journal of Educational Computing Research, 2007, 36(2) pp. 223-244

[32] F. Klassner and S. D. Anderson: Lego MindStorms: Not just for K-12 anymore, IEEE Robotics & Automation Magazine, 2003, 10(2) pp. 12-18

[33] D. C. Cliburn: Experiences with the LEGO Mindstorms throughout the undergraduate computer science curriculum, 36th Annual Frontiers in Education Conference, 2006, pp. 1-6, IEEE

[34] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman and Y. Kafai: Scratch: programming for all, Communications of the ACM, 2009, 52(11) pp. 60-67

[35] "Theme Park God on Scratch" Borrego6165. [Online] Available: https://scratch.mit.edu/projects/93279933 [Accessed: 04-Nov-2017]

[36] F. Kalelioglu and Y. Gülbahar: The effects of teaching programming via Scratch on problem solving skills: a discussion from learners' perspective, Informatics in Education, 2014, 13(1) p. 33

[37] O. Meerbaum-Salant, M. Armoni and M. Ben-Ari: Habits of programming in scratch, Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, 2011, pp. 168-172, ACM

[38] D. Franklin, C. Hill, H. A. Dwyer, A. K. Hansen, A. Iveland and D. B. Harlow: Initialization in scratch: Seeking knowledge transfer, Proceedings of the 47th ACM Technical Symposium on Computing Science Education, 2016, pp. 217-222, ACM

[39] A. Fowler, T. Fristce and M. MacLauren: Kodu Game Lab: a programming environment, The Computer Games Journal, 2012, 1(1) pp. 17-28

[40] K. T. Stolee and T. Fristoe: Expressing computer science concepts through Kodu game lab, Proceedings of the 42nd ACM technical symposium on Computer science education, 2011, pp. 99-104, ACM

[41] P. Gadanecz: The nature of positive emotions via online language learning, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 197-203, ISBN 978-1-5386-7094-1

[42] A. I. Wang and B. Wu: Use of game development in computer science and software engineering education, Computer Games and Software Engineering, K. M. L. Cooper and W. Scacchi (Eds.), Chapman and Hall/CRC, 2015, pp. 31-58

[43] S. Sheth, J. Bell and G. Kaiser: A Gameful Approach to Teaching Software Design and Software Testing, Computer Games and Software Engineering, K. M. L. Cooper and W. Scacchi (Eds.), Chapman and Hall/CRC, 2015, pp. 98-119

[44] T. Xie, N. Tillmann, J. de Halleux and J. Bishop: Educational Software Engineering, Computer Games and Software Engineering, K. M. L. Cooper and W. Scacchi (Eds.), Chapman and Hall/CRC, 2015, pp. 113-132

[45] D. Sik and J. Horvath Cz.: Open micro-Content Development with Web 2.0 and Smartphone Environment, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 29-31, ISBN 978-1-5386-7094-1

[46] K. M. L. Cooper and S. Longstreet: Model-Driven Engineering of Serious Educational Games, Computer Games and Software Engineering, K. M. L. Cooper and W. Scacchi (Eds.), Chapman and Hall/CRC, 2015, pp. 59-89

[47] E. Hayes: Game content creation and it proficiency: An exploratory study, Computers & Education, 2008, 51(1), pp. 97-108

[48] L. H. Wong and C. K. Looi: Vocabulary learning by mobile-assisted authentic content creation and social meaning-making: two case studies, Journal of Computer Assisted Learning, 2010, 26(5), pp. 421-433

[49] E. Gogh and A. Kovari: Metacognition and Lifelong Learning, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 271-275, ISBN 978-1-5386-7094-1

[50] K. Nagy, B. Szenkovits, Gy. Molnár, J. Horváth-Czinger and Z. Szűts: Gamification and microcontent orientated methodological solutions based on bring-your-own device logic in higher education, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 385-388, ISBN 978-1-5386-7094-1

[51] "Imagine is here!" In Hungarian: "Itt az Imagine!" Sulinet. [Online] Available: http://logo.sulinet.hu [Accessed: 07-Nov-2017]

[52] “Official Construct 2 Manual - Construct 2 Manual” Scirra. [Online] Available: https://www.scirra.com/manual/1/construct-2 [Accessed: 01-Dec-2017]

[53] “Top game making tutorials - Scirra.com” Scirra. [Online] Available: https://www.scirra.com/tutorials/top [Accessed: 01-Dec-2017]

[54] “ScirraVideos - YouTube” ScirraVideos. [Online] Available: https://www.youtube.com/user/ScirraVideos [Accessed: 27-Nov-2017]

[55] “Construct 2 - From Beginner to Advanced - Ultimate Course! | Udemy” J. Alexander. [Online] Available: https://www.udemy.com/construct-2-from-beginner-to-advanced-build-10-games [Accessed: 27-Nov-2017]

[56] “Index page - Scirra Forums” Scirra. [Online] Available: https://www.scirra.com/forum/ [Accessed: 03-Dec-2017]

[57] “Construct 2 Javascript SDK documentation - Construct 2 Manual” Scirra. [Online] Available: https://www.scirra.com/manual/15/sdk [Accessed: 30-Nov-2017]

[58] “Best Addicting Games - Addicting Games” Scirra. [Online] Available: https://www.scirra.com/arcade/top-addicting-games [Accessed: 28-Nov-2017]

[59] P. Baranyi and A. Csapo: Definition and Synergies of Cognitive Infocommunications, Acta Polytechnica Hungarica, 2012, 9, pp. 67-83

[60] P. Baranyi, A. Csapo and Gy. Sallai: "Cognitive Infocommunications (CogInfoCom)" Springer, 2015

[61] P. Baranyi and A. B. Csapo: Revisiting the concept of generation CE-Generation of Cognitive Entities, 6th IEEE International Conference on Cognitive Infocommunications, 2015

[62] A. Kovari: CogInfoCom Supported Education, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 233-236, ISBN 978-1-5386-7094-1

[63] K. Chmielewska, A. Gilányi and A. Łukasiewicz: Mathability and Mathematical Cognition, 7th IEEE International Conference on Cognitive Infocommunications, 2016, DOI=http://doi.org/10.1109/CogInfoCom.2016.7804556

[64] J. Hromkovič: "Algorithmic Adventures", Springer, Berlin Heidelberg, 2009

[65] S. E. Kruck, J. J. Maher and R. Barkhi: Framework for Cognitive Skill Acquisition and Spreadsheet Training, Journal of End User Computing, 2003, 15(1), pp. 20-37

[66] K. M. L. Cooper and W. Scacchi: "Computer Games and Software Engineering", Chapman and Hall/CRC, 2015

[67] G. Csapó: Sprego virtual collaboration space, 8th IEEE International Conference on Cognitive Infocommunications, 2017

[68] "MaxWhere Store - VR workspaces" MISTEMS Ltd. [Online] Available: http://www.maxwhere.com/ [Accessed: 24-Sept-2018]

[69] B. Lampert, A. Pongracz, J. Sipos, A. Vehrer and I. Horvath: MaxWhere VR-Learning Improves Effectiveness over Clasiccal Tools of e-learning, Acta Polytechnica Hungarica, 2018, 15(3), pp. 125-147, Available: http://www.uni-obuda.hu/journal/Lampert_Pongracz_Sipos_Vehrer_Horvath_82.pdf [Accessed: 12-Sept-2018]

[70] I. Horváth: Evolution of teaching roles and tasks in VR / AR-based education, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 355-360, ISBN 978-1-5386-7094-1

[71] B. Berki: Desktop VR and the Use of Supplementary Visual Information, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 333-336, ISBN 978-1-5386-7094-1

[72] Zs. T. Horváth: Another e-learning method in upper primary school: 3D spaces, 9th IEEE International Conference on Cognitive Infocommunications, 2018, pp. 405-408, ISBN 978-1-5386-7094-1

[73] B. Berki: 2D Advertising in 3D Virtual Spaces, Acta Polytechnica Hungarica, 2018, 15(3), pp. 175-190, Available: http://www.uni-obuda.hu/journal/Berki_82.pdf [Accessed: 24-Sept-2018]

[74] "Kenney • Assets" Kenney. [Online] Available: https://kenney.nl/assets [Accessed: 10-Sep-2017]

[75] "LabVIEW - National Instruments" National Instruments. [Online] Available: http://www.ni.com/en-us/shop/labview.html [Accessed: 02-Dec-2017]