VR-OCKS: A virtual reality game for learning the basic concepts of programming[1]

R.J. Segura[2], F.J. del Pino,, C. J. Ogáyar, A. J. Rueda

CEATIC, University of Jaén, Spain

Abstract:

This paper presents a prototype of a virtual reality system to teach the basic concepts of programming called VR-OCKS. The system is inspired by other visual languages such as Scratch or Kodu, and it works by proposing to the user the resolution of simple puzzles in a 3D environment. Several basic commands to a humanoid character, such as advance or turn, together with control flow structures like iteration and conditional selections, are needed to provide a solution for increasingly difficult challenges. Our aim is to attract users, usually children and teenagers, into the world of programming by taking advantage of the appeal and potential of Virtual Reality. In addition to the skills enhanced by educational programming languages, like common sense, creative thinking or systematic reasoning, others such as spatial orientation, autonomy or manual skills are also strengthened. Our experiments with adult and children users have shown a very good acceptance and great potential as educational tool.

Keywords: Computing methodologies->Virtual reality, Applied computing->Computer-assisted instruction, Applied computing->Interactive learning environments

1. Introduction

The teaching of programming at early ages is one of the objectives and recommendations for the introduction of new technologies in education. The latest reports from expert committees of professional organizations advise that the approach to computing in general, and to programming in particular, occurs within the primary education cycle (up to 12 years) and culminates in secondary education (ACM Europe & Informatics Europe, 2017). In recent years several robot kits for children (e.g., Lego Mindstorms®) or similar educational toys have emerged, with simple programming capabilities that include different movement actions, reading form sensors and basic control structures. In the same way

---

[1] This is the peer reviewed version of the following article: **Segura, R. J., del Pino, F. J., Ogáyar, C. J., Rueda, A. J. VR-OCKS: A virtual reality game for learning the basic concepts of programming. *Comput Appl Eng Educ*., 28, pp. 31–41 (2020)**, which has been published in final form at https://doi.org/10.1002/cae.22172.

[2] Corresponding author: Rafael J. Segura. Department of Computer Science, Universidad de Jaén. Campus Las Lagunillas s/n, 23071 Jaén - Spain. Email: rsegura@ujaen.es.

there are some new web-based and mobile-based applications in the market that help to learn programming, such as Grasshopper, Sololearn, Codemurai or CodeCombat. However, most of them are simple programming tutorials focused on learning a specific language (e.g., Javascript, Python, Java, etc.) by proposing small games or puzzles that must be solved with a small program. A better choice to introduce the programming concepts to new students (both children and adults) are the Block-Based Programming (BBP) languages, which have become very popular over the last decade. These languages make it possible to avoid some of the frustrations that arise when learning the syntax of a programming language (relatively complex and far from natural language), and allow the student to focus on the idea of solving problems through a simple computer device. These block-based programming languages are mostly inspired by the LOGO language (Papert, 1980), created in 1967 and inspired by functional languages. Among the most prominent we can mention Scratch (Resnick et al., 2009), Blockly (Lovett, 2017) or Kodu (MacLaurin, 2009), aimed at the introduction of the basic concepts of programming. In some cases, more advanced concepts or applications are also included; for example, Kodu and Blockly are especially focused on the creation of video games. On the other hand, Scratch is more similar to a general-purpose language, and includes event management and even object-oriented features. Taking advantage of the success of these new languages, several initiatives aimed at teaching and disseminating programming have been presented. One of the best known is Code.org, sponsored and supported by important companies, foundations and public figures in the field of computing.

Virtual Reality (VR) has become popular in recent years, mainly due to the maturity reached by sensor and display technologies, and their affordable price. As it was predicted in the 1990s, VR has been successfully applied to fields such as health, education, industrial and architectural design or entertainment. The most affordable category of immersive systems such as Google Cardboard or Samsung Gear VR takes advantage of the powerful features of modern smartphones, such as processing power, high-performance 3D graphics, integrated orientation sensors and large screens. All this makes it possible to have a basic virtual reality system for less than $300. A slightly higher investment allows access to a new class of devices represented by the Oculus Rift, HTC Vive and Sony PS VR. These systems provide a better immersive experience and enable more complex VR applications. This is mainly due to the quality of the screens, the precision and low latency of the position and orientation sensors, and the support of interaction devices. In addition to this, it is necessary to consider the role played by the modern tools of modeling and creation of virtual worlds, and the generalization of acquisition techniques based on scanning, photogrammetry or spherical photography.

In this paper we present a VR system prototype in which users must lead a character through different scenarios. For this, a simple program using basic programming instruction has to be created, so that all the obstacles are avoided to reach the goal. The remainder of the paper is structured as follows. Next section briefly reviews the use of 3D virtual spaces and VR in the teaching and learning process so far. A key aspect of the BBP languages as the representation of the control flow structures is also addressed. The section ends with a description of Scratch and the most relevant BBP environments. In the next two

sections, our VR-based educational game is described, including its mechanics and user interface, along with a more technical explanation of its implementation. Then, the process followed for the validation of the user experience is detailed. Finally, conclusions and future work are presented.

2. Previous work

The use of graphics artwork and games in learning is common from early ages. Drawings, photographs and construction games facilitate learning and stimulate the imagination. In recent years multimedia tools and video games have partially replaced these elements. From a pedagogical and educational point of view, the advantages of interactive 3D spaces versus conventional 2D spaces are diverse (Dalgarno & Lee, 2010, Richards & Taylor, 2015). Existing studies conclude that interactive 3D spaces bring some benefits, such as increasing the complexity of problems, encouraging learning by experience, enabling collaborative learning, as well as having an immersive effect on the user that improves concentration and willingness to solve problems. Moreover, 3D spaces allow training not only programming skills, but also spatial awareness. For all these reasons, the use of VR in education has gained relevance, particularly since 2009 (Mikropoulos & Natsis, 2011). Unfortunately, as Dalgarno (2015) states, most of the development of virtual spaces is based on trial and error, driven by intuition and common sense, and without a formal methodology of development that defines the core elements that must be present.

Sharing this idea, Hew & Cheung (2010) carried out an exhaustive review of fifteen works on creation of interactive environments. They detected two main methods for the use of VR in education: descriptive research and experimental research. They also identified that, in most cases, the applications described in the literature only sought to detect the degree of student satisfaction or some social aspects, and were not designed to encourage learning itself. On the other hand, Duncan et al. (2012) presented a detailed study in which they proposed a taxonomy of VR systems according to five criteria: population, educational activities, theory and learning environments, support technology and research area. Based on this taxonomy, an analysis of advantages and disadvantages of using virtual worlds in education is presented, and some areas of research that are not well developed in this topic are identified.

Therefore, one of the most important aspects of VR applied to the learning process comes through the interaction with the virtual environment. Interaction stimulates creativity and discovery, one of the principles of Piaget's constructivist learning theory (Piaget, 1954), which states that students are generators of knowledge by themselves, through their own experimentation and the influence of the social environment that surrounds them. Any domain of knowledge in which the student can develop an understanding of entities with dynamic and complex behaviors, can be exploited to build their personal knowledge. This knowledge grows and improves through exploration and experimentation, by facing increasingly complex problems. Despite its undeniable appeal, constructivism has received some criticism, mainly pointing to the need for the student discovery process to be supervised in some way in order to achieve the learning objectives (Huang , Rauch, & Liaw, 2010).

Both VR and Augmented Reality are becoming valuable tools for teaching, as experiences in different areas of education attest. However, as Merchant et al. (2014) states, game-based learning environments are more effective than the simple use of virtual worlds or simulation environments. This is a key aspect of the application of VR in education, since most limitations that affect its effectiveness are mainly linked to inability to produce an effect of presence and engagement in the user. But the same study concludes that the benefits of VR-based teaching are not only maintained over time, but also facilitate the transfer of knowledge to other contexts. Therefore, the ideal choice would be to design VR experiences in which the user interaction is inherent to the game. In this way, the advantages of both tools would be combined to achieve the objectives. Huang (2010) proposes five learning strategies from a constructivist point of view, which could be followed during the development of Virtual Reality Learning Environments (VRLE):

- Learning in real situations simulated by VR.
- Change of user roles.
- Collaborative learning.
- Problem-based learning (PBL).
- Creative learning.

The system we present incorporates, in one way or another, elements from all these strategies except collaborative learning, that will be addressed in future developments.

In any case, the number of experiences using complete VR systems in education is not high. Most of the developed systems are simple interactive 3D environments in which none of the three elements of VR identified by Burdeau (2003) are present: interaction, imagination and immersion. In most cases, systems that pretend to be virtual reality-based are actually simple 3D environments in which the user can navigate without interacting. Moreover, the existing interaction does not stimulate the imagination. Other works reduce the concept of presence to the possibility of interacting with other users, for example, in virtual worlds such as Second Life (Warburton, 2009), (S. Gregory, Lee, Dalgarno, & Tynan, 2016).

Systems that use VR for teaching programming concepts are even less frequent. Chandramouli et al. (2014) propose a VR system to teach programming concepts to engineering students. This system was developed using the outdated VRML97 for modeling the scene, and as a result, its graphics quality and interaction is limited. Moreover, target users are in general limited to young adults between 18 and 20 years old. Ortega et al. (2017) describe a 3D virtual programming language for beginners and intermediate users, aimed at increasing the number of women interested in programming. The proposed 3D-VPL environment is basically an extension of Scratch to a 2.5D space (and therefore, not a true 3D scene), limiting user interaction to setting up relationships between boxes. Finally, Grivokostopolou et al. (2016) present an interesting work for the teaching of ordering algorithms, but it assumes that the student is able to understand the concepts of iteration, sequence and bifurcation.

2.1. Visual languages applied to the teaching of programming

Traditional educational programming languages like BASIC or LOGO were similar to standard programming languages although using a simplified or more readable syntax. In contrast, influenced by the ideas noted in the previous section, most modern programming languages targeted primarily at children tend to favor the use of graphical notations over traditional code, without without forgetting to enforce widely accepted good practices as structured programming.

Structured programming emerged in the 1960s as a partial solution to the problem of the software crisis. The primary objective was to ease in some way the maintenance tasks of the programs. Its theoretical foundations are found in the Böhm-Jacopini theorem (Böhm & Jacopini, 1966), which shows that any computable function can be implemented in any programming language that has three basic structures: sequence, selection and cycle (iteration). This idea is later endorsed by the famous article by Dijkstra in which concisely (just one page) criticized the excessive use of the unconditional jump instruction, and advocated, as Hoare and Wirth previously did, for the use of the three basic structures (Dijkstra, 1968).

Programming languages designed after 1970 do not incorporate the unconditional jump instruction, or do so in a marginal way. However, the traditional notation of flow diagrams, which is still frequently used today, supports and even encourages the use of unconditional jumps. For this reason, different graphical notations began to appear for the specification of algorithms following the structured approach. The Nassi-Shneiderman diagrams or NSD (Nassi & Shneiderman, 1973) is one of the most widely used. This graphical design representation is based on the use of different blocks, associated with simple actions, conditionals and loops, which can be stacked or nested in different ways to define the algorithm (Figure 1). The construction rules of a NSD make the existence of non-linear execution flows impossible.

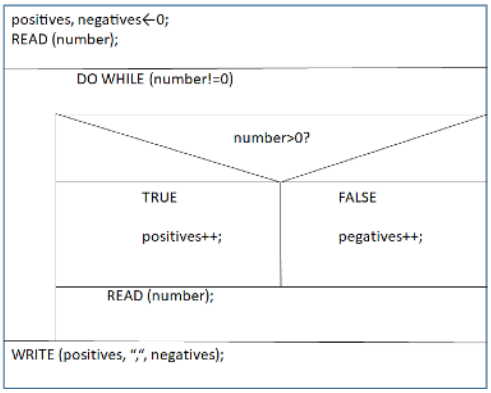

Figure 1: Nassi-Shneiderman diagram: an example algorithm for counting positives and negatives numbers.

Most of existing BBP languages are inspired by the basic programming principles that we just presented. In this way, the use of code blocks is clearly inspired by the N-S diagrams. Among all the BBP languages, the most widely used is Scratch, a visual programming language developed by the MIT Media Lab (Resnick et al., 2009). This language is very popular in education because it allows students to approach

the world of programming by using a visual and interactive 2D programming environment. It allows the developing of quite complex programs in an entertaining manner by proposing the creation of games, which stimulates children in the achievement of objectives. Its interesting features and its help to the development of an algorithmic thinking has made it a widely used programming language for people of all ages. Initiatives such as the Hour of Code have also facilitated its dissemination.

Scratch allows not only sequential programming but also event-driven programming with multiple active objects called sprites. The list of commands associated to a sprite is specified through an interactive graphical interface by stacking blocks to define a sequence, or placing some blocks into others to represent iterations and loops (see Figure 2). Each block represents a specific command from several predefined categories: *Motion*, *Looks*, *Sound*, *List*, etc. Scratch can also provide a version of the block program in Javascript, since one of its goals is to show the user that the solution can be achieved not only through the graphical interface, but also by writing code in a programming language.

Undoubtedly Scratch is the most popular programming language of its kind, but there are many other languages and platforms to teach programming at various levels and with different objectives. Among the platforms that are currently active in this area we can highlight:

- **MineCraft Code Builder**. It is one of the latest developments in Minecraft Education Edition. It allows modifying the world of Minecraft in a simple way through a specific language.
- **Blockly**. The main alternative to Scratch. Developed by Google, and used by the popular web of code.org for its programming tutorials.
- **Kodu**. Proposed by Microsoft for its Programming in School project. The goal, as with Blockly, is the construction of games.

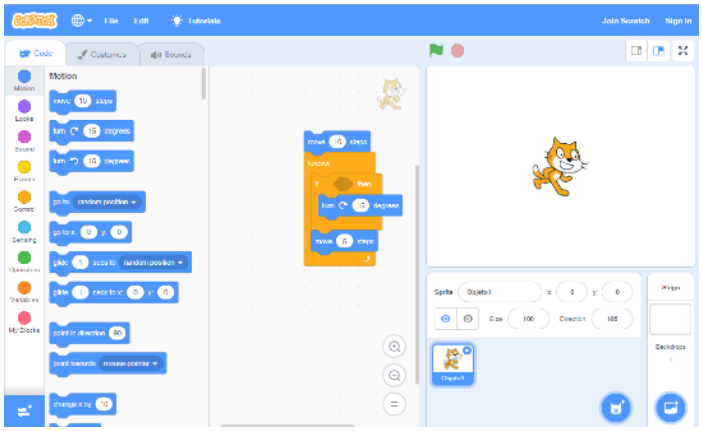

Figure 2: Scratch development environment.

3. Methodology

This work presents a video game prototype based on VR that proposes solving puzzles by using basic programming concepts. These concepts cover simple commands and basic control structures, such as

iteration and conditional selection. The player character (a kid humanoid) has to be guided from a starting point to the end point through a stage, avoiding a series of obstacles. The commands to the character are defined using a block language. Through VR the player is immersed in a virtual 3D environment where his/her spatial skills are also tested. This is the reason because we have named it VR-OCKS (VR-blOCKS). In this virtual environment the effectiveness of the programming challenge is increased, minimizing distractions, and consequently allowing the user to concentrate better on the experience. This way we can analyze how users perceive the virtual environment where they are immersed and how they use experience and logic in order to face the proposed problems.

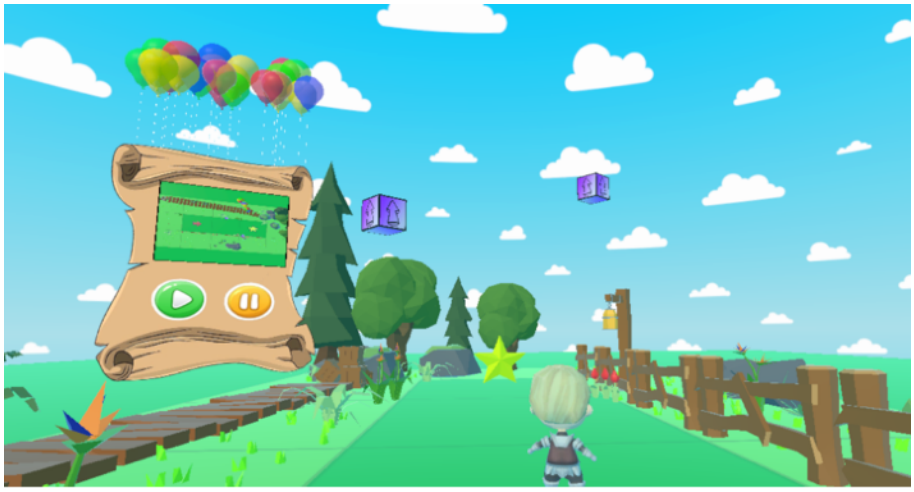

Figure 3: Main view of the game environment, showing the player character, the path to follow in dark green with the designated goal position (the yellow star), the action blocks floating in the center, the action menu at the left, and the execution area behind it, realized as a wooden walkway.

3.1. Game mechanics

Game mechanics is the set of rules and methods designed for the player to interact with the system. Therefore, it defines what the player can do and the response of the system after each action. The mechanics of VR-OCKS can be summarized as follows: the character starts at the begin of a path with several obstacles, and has to decide which advance or turn actions should be taken to reach a designated goal position. The available actions are represented by different pieces floating around the player character, and the user has to select and place a set of them in the right order to define a simple algorithm that solves the puzzle (see Figure 3). Once the user believes a solution has been reached, it can be tested by pressing a run button in an action menu. This moves the character following the defined actions; if it arrives safely at the goal position, the level is completed. The different aspects of the game mechanics are elaborated below:

- **Game board**. The puzzle to be solved is presented as a board of rectangular tiles. The tiles that belong to the valid path are shown with a different color.
- **Game goal**. The objective of the game is to make the player character walk the path to the goal tile safely.

- **Game control**. The character movements are encoded in a program that has to be created by choosing among a set of pre-established actions, called action blocks. The collection of actions that forms the program have to be placed in a special place of the scene called the execution area.
- **Valid moves**. The character can move from tile to tile according to the actions defined in the program, namely advance, turn right and turn left.
- **Victory condition**. A level is successfully completed if the character reaches the goal tile. The user cannot test a solution to the problem until there is at least one action in the execution area.
- **Losing condition**. There are two situations that can cause the user to lose the game. The first one occurs when the character goes out of a valid path tile; the second when the program leads the character to a valid tile, but this is not the goal tile designated for the level.
- **Levels**. The game has several levels (see Figure 4), which the user can select through the main menu, as long as they have been unlocked. To unlock a level it is necessary to have successfully completed the previous levels.

3.2. User interface

The game has been designed following a low poly/cartoon look, with the following elements: game board, player character, sky, execution space, decoration elements and action menu. The game boards proposed for the different levels have been designed to force the user to use the iteration and bifurcation control structures, with increasing difficulty. The tiles that are part of valid paths are shaded in a different color, and the goal tile is identified by a floating star, as illustrated in Figures 3 and 4.

Figure 4: Examples of game levels.

In addition to the main elements, decorative elements such as trees, fences, boxes, etc. have been added to make the scene more visually appealing. Its purpose is also to hide part of the board at the harder levels, so that the user is required to make an extra effort of abstraction and eventually use the minimap available at the action menu.

Figure 5: The HTC Vive VR platform (HTC2019).

User interaction is done through the interface devices provided by the HTC VIVE VR platform (Figure 5). The user can select an action with one of the two joysticks. The control of the point of view is provided by the sensors of the Head Mounted Display (HMD). The user position in the virtual world is fixed, so position changes returned by the SteamVR tracking system are ignored.

The player uses the joystick as a selector of the action block that are floating around him. The trigger button grabs an action block and allows the user to examine it and change its properties (e.g., number of iterations in a loop) through a radial menu. When an action block is captured, the allowed positions are highlighted in the execution area, as shown in Figure 6. In this way, the programming interface becomes simpler and more intuitive to the user. Finally, when the trigger is released, the action block is dropped at the position in the execution area pointed by the joystick. If none of the valid positions is selected, the action block returns to its starting position. Allowed positions include at the front, end or in an intermediate point of the list of actions currently in the execution area, but also nested within some action in the execution area that represents a control structure.

The action menu shown to the user is simple and intuitive and has been integrated into the scene (see Figure 3). This menu incorporates the minimap, along with the pause and execution buttons. The position of the camera plays a fundamental role to avoid possible dizziness in the user. The camera is located at the head of the user in a fixed observer position within the virtual world. However, tilt and pan movements are allowed to create immersion and allow the user to explore the virtual environment.

The metaphors chosen to represent the actions that control the main character (walking, turning to the left, turning to the right, loops, conditional, etc.) are intended to be easy to understand. Several sound effects have also been added to improve the interaction of the player with the system.

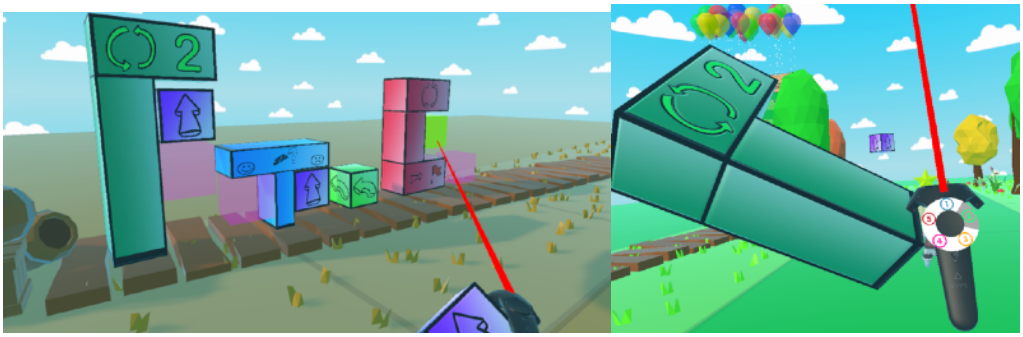

Figure 6: A view of the execution zone and the radial menu

3.4 System architecture and implementation details.

For the implementation of our system we have chosen the Unity engine for several reasons. First, it provides all the necessary tools to create interactive 3D environments in a relatively simple way. Second, it has a complete and well-documented API. And finally, it can run on different platforms such as PCs, consoles, mobile devices or web. For these reasons Unity has gained considerable popularity in recent years. The VR platform chosen is HTC VIVE (Figure 5), which is composed of two SteamVR tracking base stations, two wireless controllers and a headset with two 1080 x 1200 3.6'' amoled screens, integrated earphones, microphone, G-sensor, gyroscope and SteamVR tracking sensor. Steam VR is also used, which is the associated gaming platform required for installing applications. A completely free and functional plugin for Unity has been used for connecting with Steam VR.

The action blocks available in the game are shown in Figure 8. As can be seen, the design of the blocks are inspired by the N-S diagrams, although visual labels and colors have been incorporated following the aesthetics of video games.

a) **Go forward**. Simple action of walking forward. The character always walks in the direction he/she is looking.
b) **Turn left**. This simple action makes a -90º turn around the Y-axis of the player.
c) **Turn right**. Similarly, this action turns 90º around the Y-axis of the player.
d) **Conditional**. As in any programming language, conditionals are present, but here their use is limited. Specifically, they allow the character to take one path or another depending on a boolean variable. By now the only condition that can be checked is: *is there a wall in front of the character?* Nesting conditional instructions is a feature that also remains as future work.
e) **For loops**. The *for* loop repeats a group of actions a given number of times *n*. The parameter n can be set by the user after grabbing the block.
f) **While loop**. The actions that are within a *while* loop are repeated until a certain condition is met. In our implementation, the actions are repeated as long as the player does not reach the goal tile designated for the level. In the same manner as with conditional actions, and due to

the limited scope of our project, *while* loops can not be nested either. Therefore the body of these loops can only be simple or conditional actions.

One of the main problems to be solved in the implementation is to determine how to transfer the program defined by the user, that is to say, the structures of action blocks in the execution area to an execution logic. A command interpreter is needed for processing the actions of the user. It operates like a compiler that translates the instructions represented by the blocks into moves of the game character. The data structure that stores the user program is a multi-list of action blocks (Figure 9). Each branch in the execution flow adds a new nested list to the structure. The execution of the program is implemented by moving a pointer (similar to a program counter) along the list and performing the corresponding actions. A stack is used to restore the pointer position after finishing the execution of a group of nested action in a loop or conditional. Our prototype restricts the maximum nesting level to two, since a higher complexity is beyond the scope of a basic introduction to programming.

Further details of the implementation, including excerpts of the required scripts are available in the Appendix.

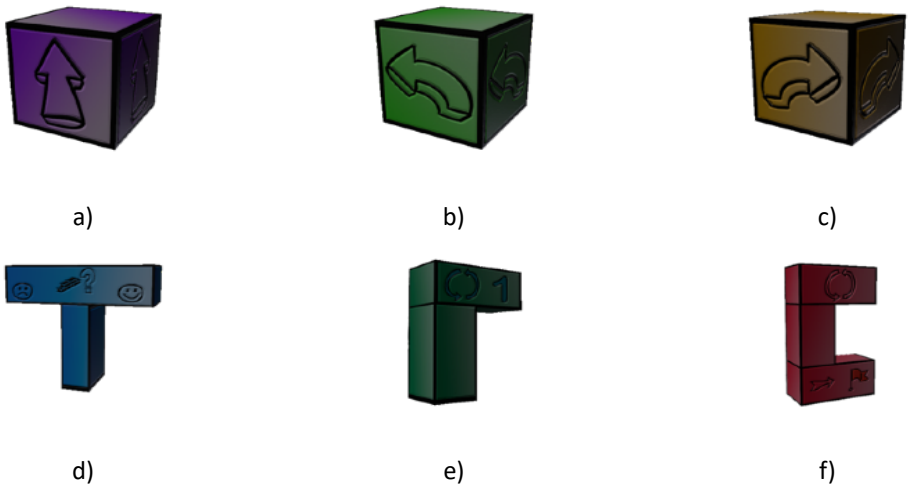

Figure 8: Action blocks defined in the game: a-c represent move commands, whereas d-f represent control structures.

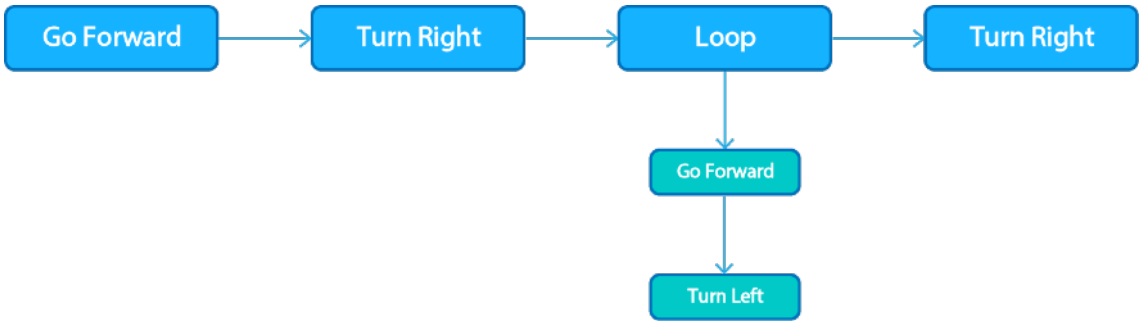

Figure 9: Data structure used for representing a program.

4. Experimentation and results

For the validation of the system, an experimentation in two phases was carried out. The first phase was designed to validate the user interface, detect all possible errors, and improve the overall system. In the second phase, the system was tested by its target users in order to verify the suitability of using our system for teaching basic programming skills.

For the first phase of the experimentation, a group of 20 users aged between 20 and 25 was recruited. We formed two subgroups. The first consisted of 10 users without previous experience with programming or VR concepts (inexperienced users). The second group, with a similar size, consisted of undergraduate and graduate students in Computer Engineering (expert users). In this first phase of the validation, the level of previous experience with VR systems has not been taken into account when making the selection of the population.

During this phase, both subgroups of users tested the system for a pre-set time (around 15 minutes). They explored all the features that the game provides and completed a survey with a standard assessment from 1 to 5 for each question (Miller, 1956). A Likert scale (Allen & Seaman, 2007) has been used, with the item format shown in Table 1. The items of the survey are presented in Table 2. In order to perform the experiment, the user was asked to solve the problems proposed in the different levels, navigate across the different menus, and freely interact with the game. It is important to point out that the number of levels solved by each user was not considered in this validation test, since we are not interested in assessing their programming skills. The survey had questions related to gameplay, design of the game board and environment, menu navigation, interface design, animation and character movement, as well as future improvements in all these aspects.

Score	Label		
1	Never	Strongly disagree	Not important at all
2	Seldom	Disagree	Unimportant
3	Sometimes	Neutral	Neutral
4	Often	Agree	Important
5	Always	Strongly agree	Most important

Tabla 1: Likert item format used in the surveys.

Question	Average inexperienced	Average expert

Selecting actions using the joystick is simple and natural.	4,8	5,0
Highlighting the areas where the actions can be placed is useful.	4,8	4,8
The initial position of the action blocks is adequate.	4,0	4,8
The action blocks used for turn left and turn right moves are intuitive.	5,0	5,0
I know how to nest actions inside loops and conditionals.	3,6	4,5
The visual appearance of the conditional flow action block is correct.	4,0	4,3
It is clear the purpose of the conditional flow action blocks.	3,6	3,8
The visual appearance of the while loop action block is correct.	4,2	4,5
The difficulty of the game increases progressively at each level.	3,8	3,5
The number of actions needed to solve a level is appropriate.	4,2	4,5
The environment has a nice look.	4,6	4,5
The elements of the environment are not intrusive.	4,8	4,0
The layout of the menus of the interface is correct	4,4	5,0
The minimap is useful to solve some levels.	3,2	3,3
Transitions between menus are smooth.	4,4	4,8
The gameplay is intuitive and fluid.	4,6	5,0

Table 2: Survey for the evaluation of the first test.

As it can be observed, most of the scores are greater than 4, and almost coincident for both inexperienced and expert users (less than 4% average difference between both groups). The greatest differences in the scores appear in questions related to the disposition of game elements, with a worse evaluation by inexperienced users. The users also evaluated differently those issues related to game mechanics, which are worse rated by expert users, more accustomed to complex comercial games. The worst ratings were obtained in the questions related to the difficulty of the levels and the utility of the minimap, two issues in which both groups of users agree. However, we decided to keep the difficulty of the level as designed, because the game was aimed at children with no prior knowledge of programming.

The second phase of the experimentation consisted of testing the system with a group of 10 children, aged between 11 and 14, with 5 girls and 5 boys. Parental consent was requested before the tests. The experimentation comprised three steps. First, a brief explanation about the mechanics of the game was provided, through a 2-minute video. This video was shown to all the children in a single session. Then they were allowed to play with the system for a span of 10 minutes. This may seem insufficient, but actually allows obtaining appropriate results for the study without producing the risks associated with

fatigue. At the end of the test, a survey was provided (see Table 3), and the number of levels completed by each child was noted.

	Question	Average	F	M
1.	The headset and joysticks are comfortable	4,10	3,83	4,50
2.	I like the look of the game	4,60	4,67	4,50
3.	It did not take me long to understand what I have to do.	4,40	4,00	5,00
4.	The way to put a box inside another is nice and clear.	4,40	4,50	4,25
5.	I understand the purpose of the conditional action	4,33	4,00	4,67
6.	The loop action (for repeating a set of actions) is easy to use and understand	4,33	4,40	4,25
7.	I got a little dizzy when I was playing	2,00	2,17	1,75
8.	The difficulty of the levels is ok	4,10	4,00	4,25
9.	I used the minimap to solve some harder levels	2,00	2,67	1,00
10.	I wish this game was used at my school	4,60	4,33	5,00
11.	I had a great time playing and I would like to tell my friends	4,80	4,83	4,75
12.	I think programming like that can be fun	5,00	5,00	5,00
	Levels completed	8,2	8,00	8,50

Table 3: Survey for the evaluation of the second test.

As it can be seen, in general the difference between the ratings given by boys and girls is barely significant. In general, the game is well rated in practically all aspects but the children highlight its attractive look and the fun it provides. The number of levels completed in the given time had little variation, although only two children reached level 10, in which the use of conditional structures is required. For this reason, question number 5 was only answered by two students, which does not lead us to any accurate conclusions. There is also no significant difference in the number of levels completed according to the gender, although it is slightly higher in boys.

Regarding the physical sensations, only one of the girls showed slight signs of fatigue during the experiment, although most of the children considered the duration of the experiment insufficient. They clearly wanted to continue using the game at the end of the tests. By analyzing the results, it can be concluded that the system is safe with respect to the physical integrity of the child.

During the experimentation we observed that the time the children spent to learn how to use the controls was shorter than expected. It took many children less than 10 seconds to place the first action block in the execution zone. They used the joysticks without needing further explanation, requiring assistance only for the parameterization of iterations through the radial menu.

One of the most frequent complaints was about the time that the animation of the character associated with the execution took. Most users thought that the execution time was excessive. The children were more anxious to move quickly to the next level than to see whether their solution was actually correct. Similarly, as it can be seen in the survey results, the utility of the minimap got low ratings, as already happened with the users in the first phase of the experimentation. This has led us to consider removing the minimap in the final prototype. A less drastic alternative would be to add a command for showing or hiding it when desired.

5. Conclusions and future work

In this paper, a prototype of a Virtual Reality system for teaching basic programming concepts has been presented. The prototype is aimed at children of approximately 12 years old, with the intention of helping them to develop an algorithmic thinking, and attracting them to the world of programming. The experimentation carried out showed the suitability and viability of the proposal, highlighting a good learning curve of the concepts and a correct set of metaphors for the different actions that can be used in a program and their management.

As future work we plan to increase the number of levels of VR-OCKS and raise the level of difficulty with more commands and less restrictive conditionals. We also want to explore the idea of designing 3D puzzles in which the character is required to climb or jump. This would better exploit the possibilities of VR, while demanding more spatial orientation and abstraction capabilities from the user. Another interesting improvement is to redefine the game as a collaborative environment in which two or more users have to cooperate to define the solution, or different parts of the solution as groups of actions that have to take place at the same time, therefore approaching concepts such as event or concurrent actions. Finally we would like to study how to incorporate more abstract programming elements (e.g., arrays or records).

References

ACM Europe, & Informatics Europe. (2017, May). The {C}ommittee on {E}uropean {C}omputing {E}ducation {R}eport ({CECE}): {I}nformatics {E}ducation in {E}urope: Are We All In The Same Boat? https://doi.org/http://dx.doi.org/10.1145/3106077

Allen, I. E., & Seaman, C. A. (2007). Likert Scales and Data Analyses. *Quality Progress*, *40*(7), 64–65. Retrieved from https://search.proquest.com/docview/214764202?accountid=14555

Böhm, C., & Jacopini, G. (1966). Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules. *Commun. ACM*, *9*(5), 366–371. https://doi.org/10.1145/355592.365646

Burdea, G. C., & Coiffet, P. (2003). *Virtual Reality Technology*. John Wiley & Sons Inc. Retrieved from https://books.google.es/books?id=hMQ8DwAAQBAJ

Chandramouli, M., Zahraee, M., & Winer, C. (2014). A fun-learning approach to programming: An adaptive Virtual Reality (VR) platform to teach programming to engineering students. In *IEEE International Conference on Electro/Information Technology* (pp. 581–586). https://doi.org/10.1109/EIT.2014.6871829

Dalgarno, B., & Lee, M. J. W. (2010). What are the learning affordances of 3-D virtual environments? *British Journal of Educational Technology*, *41*(1), 10–32. https://doi.org/doi:10.1111/j.1467-8535.2009.01038.x

Dijkstra, E. W. (1968). Letters to the Editor: Go to Statement Considered Harmful. *Commun. ACM*, *11*(3), 147–148. https://doi.org/10.1145/362929.362947

Duncan, I., Miller, A., & Jiang, S. (2012). A taxonomy of virtual worlds usage in education. *British Journal of Educational Technology*, *43*(6), 949–964. https://doi.org/10.1111/j.1467-8535.2011.01263.x

Gregory, S., Lee, M. J. W., Dalgarno, B., & Tynan, B. (2016). *Learning in Virtual Worlds: Research and Applications*. (S. (Ed. . Gregory, M. J. W. (Ed. . Lee, B. (Ed. . Dalgarno, & B. (Ed. . Tynan, Eds.). UBC Press. Retrieved from https://books.google.es/books?hl=es&lr=&id=ez_2CwAAQBAJ

Grivokostopoulou, F., Perikos, I., & Hatzilygeroudis, I. (2016). An Innovative Educational Environment Based on Virtual Reality and Gamification for Learning Search Algorithms. In *2016 IEEE Eighth International Conference on Technology for Education (T4E)* (pp. 110–115). https://doi.org/10.1109/T4E.2016.029

Hew, K. F., & Cheung, W. S. (2010). Use of three-dimensional (3-D) immersive virtual worlds in K-12 and higher education settings: A review of the research. *British Journal of Educational Technology*, *41*(1), 33–55. https://doi.org/10.1111/j.1467-8535.2008.00900.x

HTC 2019, HTC VIVE Support, Verifying your setup, https://www.vive.com/us/support/vive/category_howto/verifying-your-setup.html

Huang, H.-M., Rauch, U., & Liaw, S.-S. (2010). Investigating learners' attitudes toward virtual reality learning environments: Based on a constructivist approach. *Computers & Education*, *55*(3), 1171–1182. https://doi.org/https://doi.org/10.1016/j.compedu.2010.05.014

Lovett, A. (2017). *Coding with Blockly*. Cherry Lake Publishing. Retrieved from https://books.google.es/books?id=aF24DAEACAAJ

MacLaurin, M. (2009). Kodu: End-user Programming and Design for Games. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (p. 2:xviii--2:xix). New York, NY, USA: ACM. https://doi.org/10.1145/1536513.1536516

Merchant, Z., Goetz, E. T., Cifuentes, L., Keeney-Kennicutt, W., & Davis, T. J. (2014). Effectiveness of virtual reality-based instruction on students' learning outcomes in K-12 and higher education: A meta-analysis. *Computers & Education*, *70*, 29–40. https://doi.org/https://doi.org/10.1016/j.compedu.2013.07.033

Mikropoulos, T. A., & Natsis, A. (2011). Educational virtual environments: A ten-year review of empirical research (1999–2009). *Computers & Education*, *56*(3), 769–780. https://doi.org/https://doi.org/10.1016/j.compedu.2010.10.020

Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review*, *63*(2), 81–97. Retrieved from http://www.musanim.com/miller1956/

Nassi, I., & Shneiderman, B. (1973). Flowchart Techniques for Structured Programming. *SIGPLAN Not.*, *8*(8), 12–26. https://doi.org/10.1145/953349.953350

Ortega, F. R., Bolivar, S., Bernal, J., Galvan, A., Tarre, K., Rishe, N., & Barreto, A. (2017). Towards a 3D Virtual Programming Language to increase the number of women in computer science education. In *2017 IEEE Virtual Reality Workshop on K-12 Embodied Learning through Virtual Augmented Reality (KELVAR)* (pp. 1–6). https://doi.org/10.1109/KELVAR.2017.7961558

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, Inc.

Piaget, J. (1954). *Mindstorms: Children, Computers, and Powerful Ideas*. Oxon, United Kingdom: Routledge. Retrieved from https://books.google.es/books?id=PpfGxMDZP-4C

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for All. *Commun. ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Richards, D., & Taylor, M. (2015). A Comparison of learning gains when using a 2D simulation tool versus a 3D virtual world: An experiment to find the right representation involving the Marginal Value Theorem. *Computers & Education*, *86*, 157–171. https://doi.org/https://doi.org/10.1016/j.compedu.2015.03.009

Warburton, S. (2009). Second Life in higher education: Assessing the potential for and the barriers to deploying virtual worlds in learning and teaching. *British Journal of Educational Technology*, *40*(3), 414–426. https://doi.org/10.1111/j.1467-8535.2009.00952.x

Merchant, Z., Goetz, E. T., Cifuentes, L., Keeney-Kennicutt, W., & Davis, T. J. (2014). Effectiveness of virtual reality-based instruction on students' learning outcomes in K-12 and higher education: A meta-analysis. *Computers & Education*, *70*, 29–40.

Appendix

As discussed before, the basic mechanics of the game consists of selecting and placing several action blocks in the execution area to define a program, which is subsequently executed. Before adding an action to the program the user can modify several properties, depending on its type: number of iterations, number of steps to advance, etc. The management of the execution area is implemented in a software controller called *GameControl*, which maintains a list that stores the actions that define the program. Then, they are grouped to form higher-level structures associated with bifurcations or loops. This game mechanics is defined in the sequence diagram depicted in Figure 7. Nested action blocks represent an additional problem, since higher-level action blocks have to be properly scaled to accommodate the enclosed actions. This has been solved by creating a script called *ControllerPresetHightLight*, responsible for all the management of the *GameObjects* of *HightLight* type.

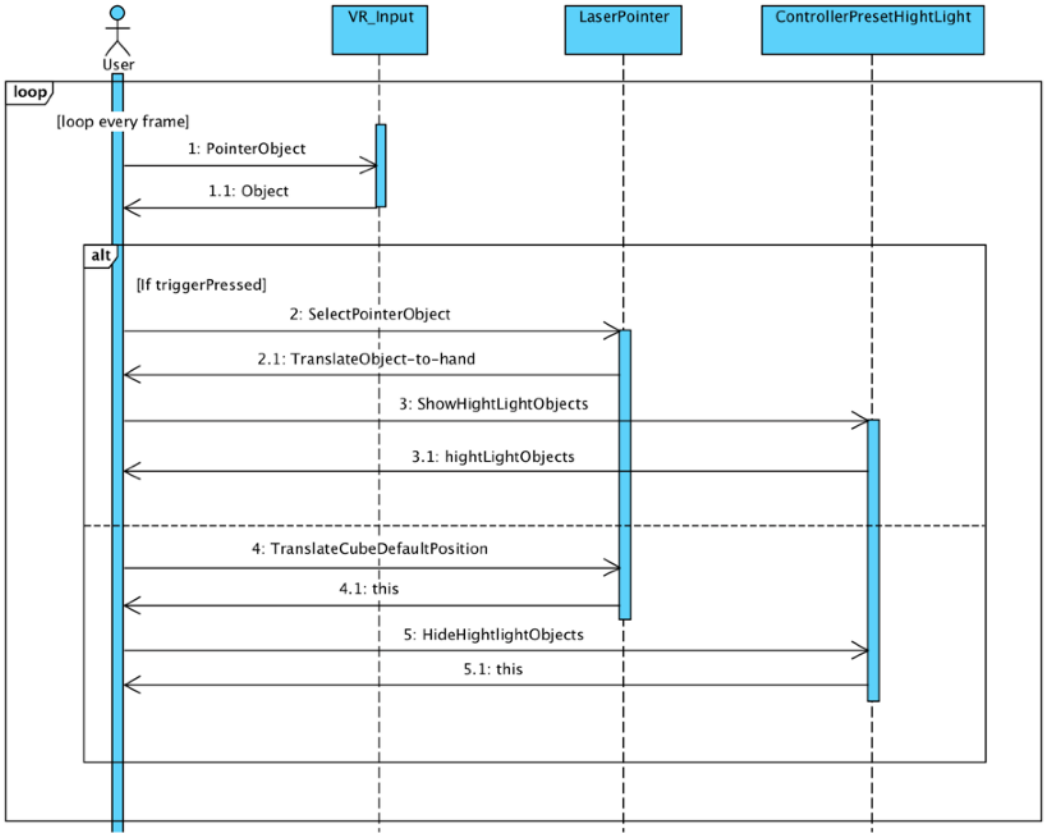

Figure 7: Sequence diagram of the selection and arrangement of actions blocks in the execution area.

Below are the C# scripts to solve the problem of locating the boxes associated with the actions in the execution area, as well as the recalculation of the dimensions of the boxes that are nested in loop structures.

```
var pivot; // Initial position

var pivotInsideLoop; // Initial position inside loop

var actionList; // List of actions in the execution area


WHEN AN ACTION IS TAKEN{

        var selectedHightLight = selectHightLight(objectTaken); // Returns hightlight of object taken

        instantiate(selectedHightLight, pivot); // Instantiate an object hightlight

        IF objectTaken != "loop" THEN // Loops are not nested

                FOR i=0 to actionList.count

                        IF actionList[i] == "loop" THEN

                                // Nesting object into loop

                                instantiate(selectedHightLight, pivotInsideLoop)

                        ENDIF

                        IF actionList[i] == "conditional" THEN

                                IF !action to left of conditional THEN

                                        instantiate(selectedHightLight, leftPosition)

                                ENDIF

                                IF !action to the right of conditional THEN

                                        instantiate(selectHightLight, rightPosition)

                                ENDIF

                        ENDIF

                ENDFOR

        ENDIF

}

WHEN AN ACTION IS RELEASED IN A HIGHTLIGHT POSITION{

        var translation = checkTranslation(objectReleased);
```

```
        // Returns the corresponding transformation of the released object
        pivot += translation // Update pivot position


// THIS SCRIPT CAPTURES THE EVENTS WHEN AN OBJECT ENTERS OR EXITS THE BODY OF THE LOOP
var loopActionList; // List of objects inside loop
FUNCTION OBJECT_ENTER(object){
        IF !loopActionList.contain(object)
                loopActionList.add(object)
                bodyLoop.escale(0,tamObjectEntered, 0)
                bodyLoop.position(0, -tamObjectEntered/2, 0)
        ENDIF
}
FUNCTION OBJECT_EXIT(objectExit){
        IF !loopActionList.contain(objectExit)
                loopActionList.add(objectExit)
                bodyLoop.escale(0,tamObjectExit, 0)
                bodyLoop.position(0, -tamObjectExit/2, 0)
        ENDIF
}
```