Applying Mixed Reality Techniques for the Visualization of Programs and Algorithms in a Programming Learning Environment

Santiago Sánchez, María Ángeles García, María del Carmen Lacave, Ana Isabel Molina, Carlos González, David Vallejo, Miguel Ángel Redondo
Universidad de Castilla-La Mancha
Paseo de la Universidad, 4, 13071 Ciudad Real (Spain)
Email: {Santiago.Sanchez, MariaAngeles.GMarin, Carmen.Lacave, AnaIsabel.Molina, Carlos.Gonzalez, David.Vallejo, Miguel.Redondo}@uclm.es

***Abstract*—Program and algorithm visualization has been a research topic for more than 25 years. Correct graphical representations have a demonstrated impact on how students understand programming concepts. Previous works on visualization tools based on trees and graphs representations tend to be too difficult for teachers to use them in their classrooms and for students to understand how they work. Moreover, new mixed reality learning environments can improve this learning experience thanks to the latest technology on the market. This paper discusses a whole new set of graphical representations used to visualize programs and algorithms through augmented reality devices. It also presents these visualizations integrated into the architecture of a newly mixed reality programming learning feature for the COLLECE 2.0 Eclipse plugin, a collaborative and distributed environment for programming learning. This new approach is expected to improve students' learning experience in introductory programming courses.**

Keywords-Program visualization; algorithm visualization; augmented reality; programming learning; eclipse.

I. INTRODUCTION

Programming learning through graphical representations is a field in which researchers have been working for more than 25 years. The main purpose of these works is to reduce the level of abstraction that programming requires to facilitate its understanding [1], fulfilling the objectives of level 2 of Bloom's taxonomy [2]. Researchers have proven that the cognitive capacities of the human being are optimized to process information in a multimodal way (i.e., visual, tactile, and aural). Nevertheless, computer programs are usually presented in a textual way (one dimension), wasting all the power of our brain [3].

On the other hand, teachers' difficulties to create these graphic representations prevent the results of these works from becoming popular in classrooms [4].
However, in recent years, this work has been intensified and redirected, due to the rise of hardware devices and technological advances that allow much more expressive representations. In this sense, the integration of immersive technologies into programming learning tools can contribute to improve learning results.

The use of these emerging technologies enables a multimodal interaction-based process, which facilitate active learning. Among the different devices that offer this kind of interaction, it is worth highlighting those that integrate mixed reality capabilities. These provide a natural learning environment where the student's actions in the physical world influence the virtual one. Mixed reality glasses Microsoft HoloLens hold a dominant position on the market as a device capable of mixing the physical and virtual worlds, covering most of the continuum of virtuality defined by Milgram and Kishino from augmented reality to augmented virtuality [5], but especially focused on the first one. Thus, it is possible to expand the capabilities of traditional programming learning systems. To do that, new graphic representation techniques and new architectures that enable their manipulation need to be defined.

In this context, the work discussed in this article emerges, encompassed within a more ambitious scenario whose final objective consists in building a new generation of programming learning tools based on interactive technologies [6].

The contribution described in this article represents an approximation for the graphical representation of programs and algorithms through mixed reality, as well as a potential architecture to support it. As a practical application, this approach is integrated into COLLECE 2.0, a collaborative and distributed environment for programming learning through problem solving, which is based on the Eclipse platform [7], currently available for download at http://blog.uclm.es/grupochico/proyecto-iapro/collece-2-0/.

This article shows the proposed architecture as a complete environment oriented to programming learning, highlighting the new visualization capabilities of programs and algorithms. The rest of the paper is organized as follows. In Section 2, some similar solutions are presented, as well as previous works on which this work is based. Then, Section 3 focuses on the proposal of this work and the system architecture. Section 4 discusses the different tests performed to obtain the set of final visualizations. Finally, Section 5 draws some final conclusions and suggests possible lines of future work.

II. RELATED WORK

Works on visualization of programs and algorithms are very varied and provide results both for and against the effectiveness of their use in the educational context. Within the first group, in [8], it was made an evaluation of the extrinsic and intrinsic motivation of the students, resulting that these motivations increase when using visualizations of programs and algorithms in the classroom. In [9], it was proposed the resolution of the problem of the knapsack through a textual animation, in which the students had to identify the problem that was finally implemented, obtaining very satisfactory results. In another work [10], the Alice tool was used to teach how to solve recursive problems through 3D visualizations that represent lines of code. Although the students did not successfully solve the problems, they demonstrated certain facilities to deal with them. Similar results were obtained in [11], where the use of the Jeliot tool improved the students' understanding of control structures and loops. In [12], some experiences were made trying to discover why it is so difficult for students to understand recursive programming; they concluded that using visualization tools to display the trace of the program helps them understand how the programs work and how to solve the exercises better.

Among the works that reflect their skepticism about the effectiveness of visualizations in programming learning, the following are noticeable. In [1], it was shown that the algorithm visualization technology is educationally effective depending on how it is used, rather than on the quality of the visualizations. In [13], it was studied the effectiveness of teaching from the teacher's point of view and learning from the student's point of view. In the first case, it was concluded that the teacher must put too much effort in contrast to what these visualizations actually provide, while in the second case no substantial benefits were achieved. In this work it is concluded that for the visualizations to be pedagogically useful, they must support students' interaction and promote active learning, as stated in [4] and [14]. In this sense, the works of [15] and [16] come together through the idea that the teaching community is quite reluctant to incorporate visualization tools, due to the costs of installation, learning, creation, and maintenance that they imply, as well as the fear of losing control of the classes while the applications are used.

However, there are several tools that try to alleviate these disadvantages, which have been analyzed considering the taxonomy defined by Myers [3], which classifies the visualization of programs according to the information to be rendered (i.e., code, data, or algorithm) and to its nature: dynamic or static. SRec [17] is able to dynamically visualize the trace of recursive algorithms; those studied in [15], for functional programming (Kiel and WinHIPE) and object-oriented programming (BlueJ and Jeliot); JAVENGA [18], used in the visualization of network and graph algorithms; Visual LinProg [19], to visualize algorithms of linear problems; VISBACK [20], for dynamic visualization of recursive backtracking algorithms using trees; ALGOLIPSE [21], to visually represent the execution of algorithms on data structures and recursive algorithms; among others.

All the analyzed applications are framed in traditional interaction systems. Regarding the use of mixed reality techniques for teaching in the classroom, several experiments have been conducted, and, although they are not directly related to the visualization of programs and algorithms, it demonstrates the advantages they offer. Thus, in [22], a mixed reality environment, SMALLab, was created, aimed at primary and secondary school students, which allowed students to express themselves using their own bodies and improving the learning process. In [23], objects of the physical world replicated in a virtual world (i.e., cross-reality objects [24]) were used so that students could remotely work in a digital laboratory; the evaluation performed with the students positively demonstrated the use of this technology [25]. On the other hand, in [26], a system of cameras, projectors, and Cuisenaire rods (wooden sticks with different measures) was used to satisfactorily teach mathematical concepts to children. Some more related experiments with programming were conducted in [27], where a set of augmented reality physical markers were used to answer different programming questions, visualizing different 3D models related to the questions. The students enjoyed the activities and the work concluded that there was an increase in their motivation to learn programming concepts, but not so much to understand them, since more tests had to be done. Finally, a systematic literature review on the topic is conducted in [28], where the authors draw some conclusions related to the advantages of using augmented reality in education, such as learning gains, motivation, interaction and collaboration, and its main purpose, related to explain a topic of interest as well as providing additional information.

As a final remark in this section, it is important to highlight that the work described in this article is based on COLLECE [29], a groupware tool where multiple users can work collaboratively, thanks to a turn-based approach, in a shared source file written in Java. A series of improvements were made on this tool that resulted in COLLECE 2.0 [7], a complete programming learning environment, based on Eclipse, with collaboration capabilities oriented to real-time project editing, version control, communication, and other awareness-related elements. This environment has been extended with techniques of mixed reality, although without including techniques of improved visualization of programs and algorithms.

III. Architecture

The architecture of the proposal presented in this paper is based on the Eclipse development environment. This platform is used by most of the students who learn programming, mainly because of the facilities it offers. It features native support for the Java programming language, syntax autocompletion, project management, program compilation and execution tools, and extensibility capabilities through plugins. This system of plugins enables us to build complete applications that directly benefit from the possibilities offered by Eclipse. Thanks to this feature, and taking advantage of the familiarity of the students with Eclipse, the COLLECE 2.0 programming learning environment was built as a modular Eclipse plugin [30].

COLLECE 2.0 is proposed as a development environment that serves first-year undergraduate students in Computer Science to learn the basics of programming by solving problems, such as those studied in introductory programming courses (e.g., CS101). For this, the environment offers different mechanisms that facilitate learning and collaborative work among students, highlighting project-oriented work sessions, multi-user editing of source code in real time, tele-pointers, blocking of code regions, communication through chat, and statement of the problem to be solved, among others. Regarding the implementation, as stated before, it is based on Eclipse plugin, whose architecture relies on a set of modules that are responsible for different tasks, such as synchronization between users, which follows a client-server network model where a central server takes control and maintains session synchronization among the rest of the clients that connect to it. A server can manage different work sessions at the same time. These work sessions maintain the global context between the connected clients and the server, that is, the data of the users, the status of the associated projects, and the information related to the server itself. All this information and the related interactions are presented to the user through different views developed using the set of Eclipse SWT widgets.

One of the features implemented in COLLECE 2.0 is the capability to visualize programs and algorithms through an external augmented reality device that facilitates the interaction with the system. The device, introduced in Section 1, facilitates the reconstruction of the physical space, identifying typical elements of the environment, such as the floor, walls, tables, and chairs. Thanks to this, we can precisely indicate the position of the physical world where the program or algorithm is required to be visualized, in addition to sharing the visualization with another user who also uses the device simultaneously, or interacting with the visualization through gestures and voice recognition. This interaction allows the user to perform tasks, such as examining the value of the variables, advancing backward or forward in the execution of the algorithm, as if it were a debugger, or discovering certain characteristics of the program when the user physically approaches the visualization, among others.

The integration of the augmented reality device with the environment is done through a new Eclipse plugin that works together with COLLECE 2.0. This new plugin is responsible for performing an analysis of the program to be viewed to extract the relevant information, in addition to establishing and maintaining a network connection between the device and the system to exchange information related to the user's own visualization and physical context.

IV. Definition of Visual Representations

The methodology used to provide COLLECE 2.0 with the representations used during the visualization of algorithms and programs has gone through an exhaustive process of refinement, thanks to the collaboration of several experts, teachers, and students, who have contributed different ideas by conducting surveys. The participants answered several questions, which are now listed:

- How would you graphically represent a condition statement: IF … THEN … ELSE …?
- How would you graphically represent a selection statement: SWITCH … CASE …?
- How would you graphically represent the execution of a loop?
- How would you graphically represent the definition of a function?
- How would you graphically represent the return value of a function?
- How would you graphically represent the evaluation of an expression?

The results obtained were very varied, although most of the participants agreed on the use of flow diagrams to make the representations. Those that did not, contributed certain designs related to boxes (expressions), spirals (loops), telephones (function definitions), and branches (control sentences). From the study of these designs, a representation based on roads was extracted (see Figure 1), sufficiently abstract and scalable to represent any type of program.

This set of roads and traffic signs enables the visualization of the program execution flow in a natural way for the user, since he/she is familiar with them in his/her daily life. The fact that students are familiar with roads and signs facilitates the use of these metaphors to help them understand programs and algorithms through their static representation.

Using this metaphor, a modular set of blocks have been designed to construct the visual representations. These representations are explained below by referencing them numerically according to Figure 1.

The representation associated with the condition statements, IF ... THEN ... ELSE (1), shows a fork with two branches in which the left branch supposes the execution when the condition to be evaluated is fulfilled (THEN), while the right branch involves the execution when this condition is not (ELSE).

As in the previous representation, the selection sentences, SWITCH ... CASE ... (2), use a fork, but this time with three branches. However, in this case, the central branch which represents the selected case during the evaluation of the expression is exclusively used, leaving the other two as merely symbolic branches. This has been decided in order to

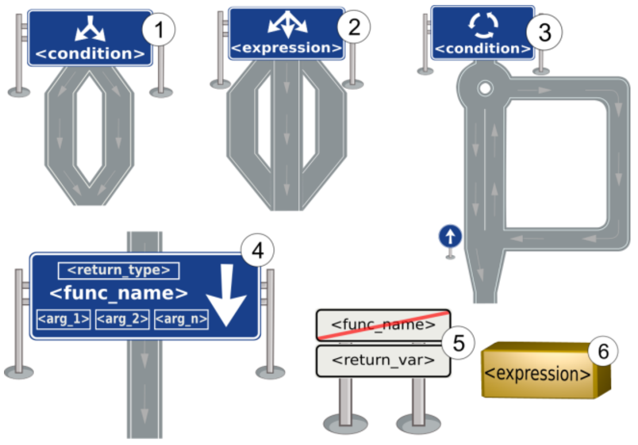

Figure 1. Set of visual representations: (1) condition statement IF … THEN … ELSE …, (2) selection statement SWITCH … CASE …, (3) loop execution, (4) function definition, (5) function return value, (6) expression evaluation.

improve the scalability of the visualization process if the number of cases of the sentence increases disproportionately.

The proposed representations for the loops (3) are based on the metaphor of roundabouts, where a vehicle can travel indefinitely and cyclically. However, the concept has undergone certain modifications to improve scalability (e.g., to support nested sentences), making exhaustive use of the different lanes of the road. Conceptually, the visualization is interpreted through a vehicle (which would represent the step-by-step execution of the program) that would reach the roundabout in the north where the condition of the loop would be evaluated. If this condition is fulfilled, the vehicle would execute the iteration of the loop taking the second exit of the roundabout. Once the iteration has been completed, the vehicle would reach the roundabout in the south, where the condition would be evaluated again. In case this time the condition was not fulfilled, the vehicle would leave the loop taking the third exit of the roundabout.

Regarding function definitions (4), their representation is conceptually based on the traffic signal of exit to city from highway, indicating the beginning of a function, which will be followed by another set of representations indicating the body of the function and, finally, its return sentence. This representation shows information about the type of data that the function returns, the input arguments, and the name of the function itself.

Function returns (5) are represented as a traffic signal that mimics the one existing in real life and that denotes end of city. This representation shows the name of the function from which it is returning and the variable whose value is returned.

Finally, the evaluation of expressions (6), such as, for example, the assignments to variables or the invocations to functions, are represented as a box that contains the expression that will be executed. These boxes are located on the roads, representing the position where they would be in the program.

The representations discussed here are used to display the program statically in order to provide an overview of its structure. The system makes a direct association between certain sentences of the language and their corresponding representation. The set of sentences to be visualized is rich enough to represent any program with them. However, no distinction is made between the types of loops, such as the classics "for", "while" and "do ... while", and their different variants, but all of them are encompassed in a single representation, thus abstracting the user from the language implementation details.

These representations have been evaluated through a pilot test with a small sample of student. Two questions have been presented for the students after they have worked with the representations:

- Q1: I think the proposed notation can be motivating for those who are learning to program.
- Q2: I like the proposed representation to model algorithms.

These preliminary obtained results showed a positive feedback from the students, who found the representations useful and easy to understand. However, in depth evaluations have to be conducted in order to better analysis of the results.

V. Example of Applying Representations

To test the set of representations, a visualization of a function was made. The code for that function is showed in Figure 2, which checks whether the numbers in a list are even, and in that case, increases them by one unit; otherwise, it decrements them. This function is visualized through a graphical representation in Figure 3.

```
public static int [] changeNums(int [] nums) {
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] % 2 == 0) {
            nums[i]++;
        }
        else {
            nums[i]--;
        }
    }
    return nums;
}
```

Figure 2. Sample code listing for further program visualization.

Thanks to the rendered visualization, a user can quickly identify the elements of the program. In this case, the visualization includes the definition of a function ("changeNums") that includes a loop with a condition statement and its two possible branches. Finally, it shows

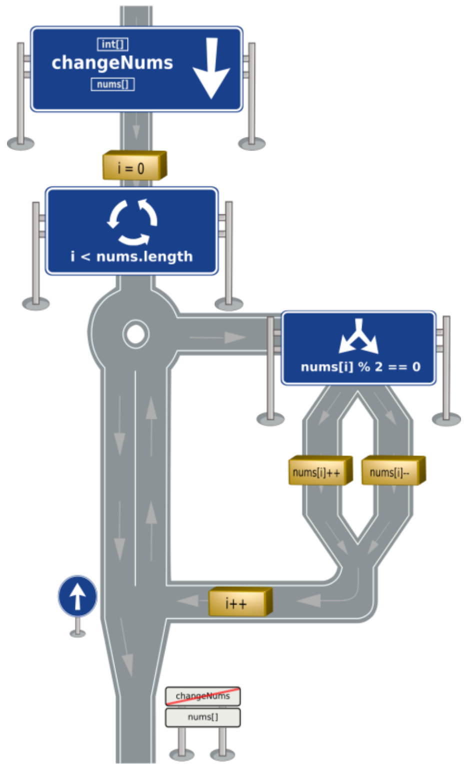

Figure 3. Visual representation of a function definition.

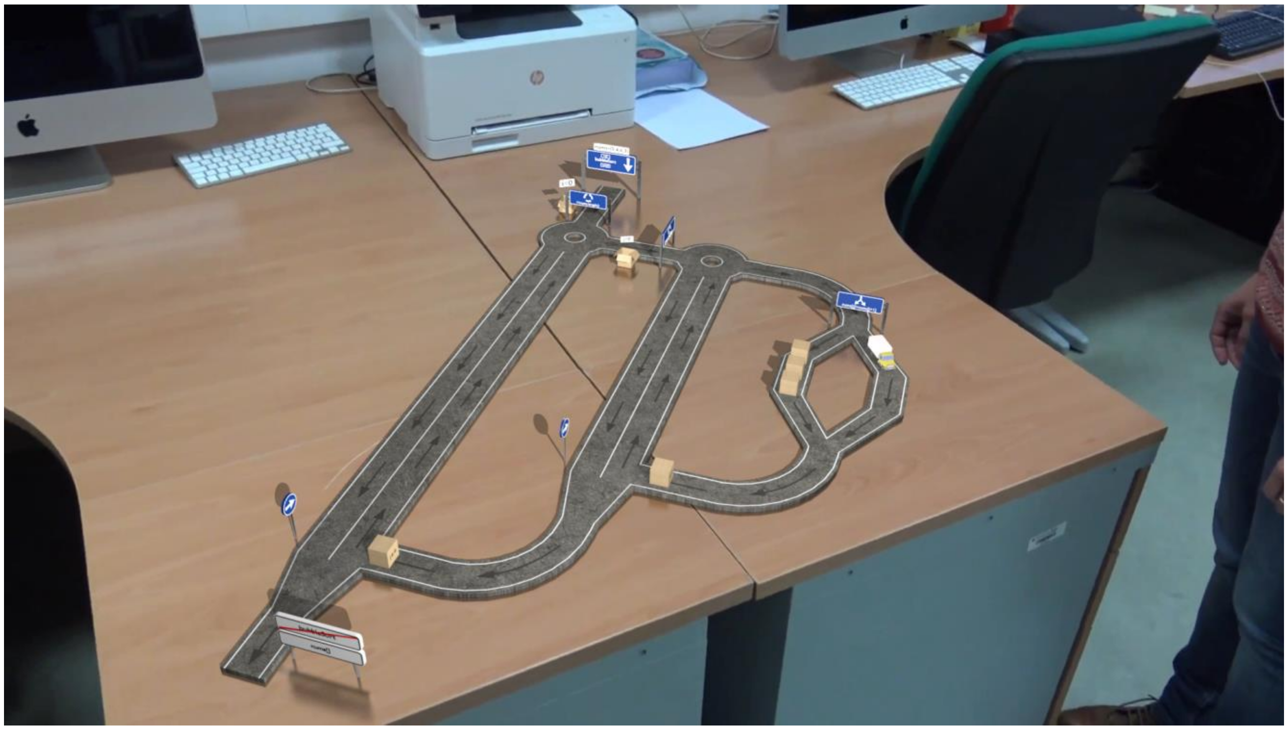

Figure 4. 3D visual representation of the Bubblesort algorithm seen through an augmented reality device.

how the definition of the function ends and the variable that returns ("nums").

Thanks to the representations generated in the physical space and visualized through the augmented reality device, Figure 4 graphically shows the Bubblesort algorithm displayed on a table, as seen by a user of the system. This visualization includes 3D elements associated to each of the 2D representations that have been introduced in the previous section. We can identify the definition of a function with two loops, one of them nested within the other, which also contains a one-branch condition statement.

VI. Conclusion and Future Work

The visualizations discussed in this paper are static representations that need to be rigorously evaluated before obtaining any conclusion regarding their effectiveness. However, its flexibility and scalability to represent programs and algorithms is highlighted as shown in the representation of the algorithms proposed in this paper, involving an advance over other algorithm representation tools, such as those mentioned in Section 2.

The next step will be composing a selection of relevant algorithms with these representations to evaluate their effectiveness with undergraduate students in the first courses of introduction to programming.

In these experiments, the effectiveness of the representations to visualize programs and algorithms will be evaluated, both subjectively and objectively (through eye-tracking techniques).

Acknowledgment

This research has been funded by the Ministry of Economy, Industry and Competitiveness, and the European Regional Development Fund through the project TIN2015-66731-C2-2-R.

References

[1] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, pp. 259-290, 2002.

[2] B. S. Bloom, "Taxonomy of educational objectives. Vol. 1: Cognitive domain," *New York: McKay*, pp. 20-24, 1956.

[3] B. A. Myers, "Taxonomies of visual programming and program visualization," *Journal of Visual Languages & Computing*, vol. 1, pp. 97-123, 1990.

[4] G. Törley, *Algorithm visualization in programming education* vol. 4, pp. 68-80, 2009.

[5] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, pp. 1321-1329, 1994.

[6] M. Ortega *et al.*, "iProg: development of immersive systems for the learning of programming," *Proceedings of the XVIII International Conference on Human Computer Interaction*, Cancun, Mexico, 2017, pp. 1-6.

[7] S. Sánchez, M. A. Redondo, D. Vallejo, C. González, and C. Bravo, "COLLECE 2.0: A distributed real-time collaborative programming environment for the Eclipse platform," *Proceedings of the International Conference Interfaces and Human Computer Interaction 2017*, Lisbon, Portugal, 2017, pp. 136-142.

[8] J. Á. Velázquez-Iturbide, I. Hernán-Losada, and M. Paredes-Velasco, "Evaluating the Effect of Program Visualization on

Student Motivation," *IEEE Transactions on Education*, vol. 60, pp. 238-245, 2017.

[9] C. Kann, R. W. Lindeman, and R. Heller, "Integrating algorithm animation into a learning environment," *Computers & Education*, vol. 28, pp. 223-228, 1997.

[10] W. Dann, S. Cooper, and R. Pausch, "Using visualization to teach novices recursion," *ACM SIGCSE Bulletin*, vol. 33, pp. 109-112, 2001.

[11] R. B.-B. Levy, M. Ben-Ari, and P. A. Uronen, "The Jeliot 2000 program animation system," *Computers & Education*, vol. 40, pp. 1-15, 2003.

[12] C. Lacave, A. I. Molina, and J. Giralt, "Identificando algunas causas del fracaso en el aprendizaje de la recursividad: Análisis experimental en las asignaturas de programación," *Jornadas de Enseñanza Universitaria de la Informática (19es: 2013: Castelló de la Plana)*, Castellón de la Plana, Spain, 2013, pp. 225-232.

[13] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, et al., "Exploring the role of visualization and engagement in computer science education," *ACM SIGCSE Bulletin*, pp. 131-152, 2002.

[14] L. Fernández and J. Á. Velázquez, "Estudio sobre la visualización de las técnicas de diseño de algoritmos," *Proceedings of the VII Congreso Internacional de Interacción Persona-Ordenador*, pp. 315-324, 2007.

[15] A. Pérez Carrasco, "Sistema Generador de Animaciones Interactivas para la Docencia de Algoritmos Recursivos," Ph.D. Dissertation, Universidad Rey Juan Carlos, Spain, 2011.

[16] J. Urquiza Fuentes, "Generación semiautomática de animaciones de programas funcionales con fines educativos," Ph.D. dissertation, Universidad Rey Juan Carlos, Spain, 2008.

[17] L. Fernández-Muñoz, A. Pérez-Carrasco, J. Á. Velázquez-Iturbide, and J. Urquiza-Fuentes, "A framework for the automatic generation of algorithm animations based on design techniques," *Proceedings of the European Conference on Technology Enhanced Learning*, pp. 475-480, 2007.

[18] T. Baloukas, "JAVENGA: JAva-based Visualization Environment for Network and Graph Algorithms," *Computer Applications in Engineering Education*, vol. 20, pp. 255-268, 2012.

[19] V. Lazaridis, N. Samaras, and A. Sifaleras, "An empirical study on factors influencing the effectiveness of algorithm visualization," *Computer Applications in Engineering Education*, vol. 21, pp. 410-420, 2013.

[20] J. F. Pérez Mena, "VisBack Herramienta para la visualización de algoritmos de backtracking," B.S. Thesis, Universidad de Castilla-La Mancha, Spain, 2015.

[21] L. A. Fava, M. A. Schiavoni, J. Rosso, A. C. Falcone, and L. Ronconi, "ALGOLIPSE: una herramienta educativa para mejorar la comprensión de algoritmos y estructuras de datos," *Proceedings of the XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016)*, 2016, pp. 1280-1290.

[22] S. Hatton, D. Birchfield, and M. Colleen, "Learning metaphor through mixed-reality game design and game play," *Proceedings of Sandbox 2008: An ACM SIGGRAPH Videogame Symposium, Sandbox'08*, pp. 67-74, 2008.

[23] A. Peña-Ríos, V. Callaghan, M. Gardner, and M. J. Alhaddad, "Remote mixed reality collaborative laboratory activities: Learning activities within the InterReality portal," *Proceedings of the 2012 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology Workshops, WI-IAT 2012*, pp. 362-366, 2012.

[24] J. A. Paradiso and J. A. Landay, "Cross-reality environments," *IEEE Pervasive Computing*, vol. 8, pp. 14-15, 2009.

[25] P. Rios, C. Anasol, V. Callaghan, M. Gardner, and M. J. Alhaddad, "Experiments with collaborative blended-reality laboratory technology for distance learners," *Proceedings of The Immersive Learning Research Network Conference*, 2015.

[26] S. Marichal, A. Rosales, F. G. Perilli, A. C. Pires, E. Bakala, G. Sansone, *et al.*, "CETA: Designing mixed-reality tangible interaction to enhance mathematical learning," *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services, MobileHCI 2017*, 2017, pp. 1-13.

[27] N. Salazar Mesía, G. Gorga, and C. V. Sanz, "EPRA: Herramienta para la Enseñanza de conceptos básicos de programación utilizando realidad aumentada," *X Congreso sobre Tecnología en Educación & Educación en Tecnología (TE & ET)*, Corrientes, Argentina, 2015, pp. 426-435.

[28] J. Bacca, S. Baldiris, R. Fabregat, S. Graf, and Kinshuk, "Augmented reality trends in education: A systematic review of research and applications," *Educational Technology and Society*, vol. 17, pp. 133-149, 2014.

[29] C. Bravo, R. Duque, and J. Gallardo, "A groupware system to support collaborative programming: Design and experiences," *Journal of Systems and Software*, vol. 86, pp. 1759-1771, 2013.

[30] S. Sánchez, M. Á. García, C. Bravo, and M. Á. Redondo, "Sistema COLLECE mejorado para soportar aprendizaje colaborativo de la programación en tiempo real sobre Eclipse," *IE Comunicaciones*, vol. 26, pp. 73-82, 2017.