Doctoral Dissertation

Development and Evaluation of Immersive Systems for Learning Programming

Submitted in partial fulfilment of the requirements
of University of Castilla-La Mancha
for the degree of Doctor of Philosophy
(Computer Science).

November, 2020

Author

Santiago Sánchez Sobrino

Supervisors

Dr. Carlos González Morcillo - Dr. David Vallejo Fernández

Author:
SANTIAGO SÁNCHEZ SOBRINO

Supervisors:
DR. CARLOS GONZALEZ MORCILLO - DR. DAVID VALLEJO FERNANDEZ

This document has been formatted with LaTeX. Images generated with Blender, composed with Gimp and Inkscape.

Abstract

Programming is a field of study that affects other disciplines transversally, either because of its cognitive benefits or because of the high demand for professionals. The latest statistics from 2020 reveal that 6 of the 10 highest paid jobs in the US require programming skills. In addition, it is estimated that by 2029 the number of jobs in the areas of science, technology, engineering and mathematics will grow by up to 8%, while the rest will grow by 3.4%. Specifically, jobs related to Computer Science and Mathematics are expected to increase by 12.1%, making these areas the fastest growing in 2029.

The above data confirms a trend that is occurring in today's world, which is preparing for a fourth industrial revolution (industry 4.0) driven by emerging technological advances in a number of fields that will require programming skills such as robotics, artificial intelligence and quantum computing, among others. Therefore, it is essential to facilitate the training of new professionals in Computer Science. However, learning to program presents certain difficulties due to the high level of abstraction required to assimilate certain concepts of programming, either those related to the structure of the program itself (for example control structures and loops), as other more abstract (for example concurrent programming and recursion). This level of abstraction can be properly managed in the early stages of learning, using mechanisms that facilitate the learning of programming through graphic visualizations and collaborative development environments.

This doctoral dissertation proposes a set of solutions and learning environments oriented to different educational levels, presented through a compendium of publications in indexed scientific journals and international conferences. As a novelty, these proposals are based on the use of emerging technologies for the visualization of programs, algorithms and programming concepts, through the use of Augmented Reality and three-dimensional graphic representations. These visual proposals arise through a new graphic notation, called ANGELA, which is integrated into the other proposals through different adaptations made according to the learning environment where it will be used. The validation of these proposals has been done through various assessments conducted with real students, which have allowed to improve and evolve the initial proposals throughout this research. To do this, different dimensions have been analyzed regarding the impact of user motivation on learning programming, usefulness, ease of use and determination of students to use these proposals.

Finally, an analysis of the results, exposed in the scientific publications, is made, which confirms the initial working hypothesis of this doctoral dissertation: the use of emerging programming learning tools improves the overall learning experience of users.

Resumen

La programación es un ámbito de estudio que afecta transversalmente a otras disciplinas, ya sea por sus beneficios cognitivos o por la alta demanda de profesionales existente. Las últimas estadísticas de 2020 revelan que 6 de los 10 empleos mejor pagados de EEUU requieren competencias vinculadas a la programación. Además, se estima que para 2029 el número de empleos en las áreas de ciencias, tecnologías, ingeniería y matemáticas crezcan hasta un 8%, mientras que el resto lo hagan un 3.4%. Específicamente, se prevé que los puestos de trabajo relacionados con la Ingeniería Informática y las Matemáticas se incrementen en un 12.1%, convirtiendo estas áreas en las de mayor crecimiento en 2029.

Los datos anteriores confirman una tendencia que se está produciendo en el mundo actual, el cual se está preparando para una cuarta revolución industrial (industria 4.0) marcada por avances tecnológicos emergentes en una serie de campos que requerirán habilidades de programación como, por ejemplo, la robótica, la inteligencia artificial y la computación cuántica, entre otros. Por tanto, resulta esencial facilitar la formación de nuevos profesionales de la Ingeniería Informática. Sin embargo, el aprendizaje de la programación presenta ciertas dificultades debido al alto nivel de abstracción que se requiere para asimilar ciertos conceptos de programación, ya sean aquellos relacionados con la propia estructura del programa (por ejemplo estructuras de control y bucles), como otros más abstractos (por ejemplo la programación concurrente y la recursividad). Dicho nivel de abstracción puede gestionarse adecuadamente en las etapas iniciales del aprendizaje, empleando mecanismos que faciliten el aprendizaje de la programación mediante visualizaciones gráficas y entornos de desarrollo colaborativo.

En esta tesis doctoral se propone un conjunto de soluciones y entornos de aprendizaje orientados a distintos niveles educativos, presentado mediante un compendio de publicaciones en revistas científicas indexadas y actas de congresos internacionales. Como novedad, estas propuestas se basan en la utilización de tecnologías emergentes para la visualización de programas, algoritmos y conceptos de programación, mediante el uso de Realidad Aumentada y de representaciones gráficas tridimensionales. Estas propuestas visuales surgen a través de una nueva notación gráfica, denominada ANGELA, la cual se encuentra integrada en el resto de propuestas mediante diferentes adaptaciones realizadas de acuerdo al entorno de aprendizaje donde se vaya a utilizar. La validación de estas propuestas se ha realizado mediante distintas evaluaciones llevadas a cabo con estudiantes reales, que han permitido mejorar y evolucionar las propuestas iniciales a lo largo de esta investigación. Para ello, se han analizado diferentes dimensiones relativas al impacto de la motivación de los usuarios en el aprendizaje de la programación, la utilidad, la facilidad de uso y la determinación del de los estudiantes a utilizar estas propuestas.

Finalmente, se realiza un análisis de los resultados, expuestos en las publicaciones científicas, gracias al cual se confirma la hipótesis de trabajo inicial que plantea esta tesis doctoral: la utilización de herramientas de aprendizaje de la programación emergentes mejora la experiencia de aprendizaje de los usuarios.

To my parents
Ana Belen and Santiago

Declaration

I hereby declare that this doctoral dissertation entitled "***Development and Evaluation of Immersive Systems for Learning Programming***" is a presentation of original work and is my own personal effort. This work has not previously been submitted and / or evaluated by this or any other university.

The content presented here is based on my reading and understanding and has not been taken from other sources, except where otherwise indicated. All sources are properly recognized within the text as bibliographical references.

Furthermore, I declare myself to be one of the main authors of the work used in this Thesis. In this regard, the following works published in indexed scientific journals have been considered in the presentation of this doctoral dissertation by compendium of publications:

- S. Schez-Sobrino, C. Gmez-Portes, D. Vallejo, C. Glez-Morcillo, and M. Á. Redondo, "An Intelligent Tutoring System to Facilitate the Learning of Programming through the Usage of Dynamic Graphic Visualizations". *Applied Sciences*, 2020 [49]. IF: 2.474 (2019), Q2.

- S. Schez-Sobrino, D. Vallejo, C. Glez-Morcillo, M. Á. Redondo, and J. J. Castro-Schez, "RoboTIC: A serious game based on augmented reality for learning programming". *Multimedia Tools and Applications*, 2020 [51]. IF: 2.313 (2019), Q2.

- S. Schez-Sobrino, M. Á. García, C. Lacave, A. I. Molina, C. Glez-Morcillo, D. Vallejo, and M. Á. Redondo, "A modern approach to supporting program visualization: from a 2D notation to 3D representations using augmented reality". *Multimedia Tools and Applications*, 2020 [48]. IF: 2.313 (2019), Q2.

Acknowledgement
Agradecimientos

Acknowledgement

We have reached the end of this journey in which many people have accompanied me; family, friends, acquaintances and colleagues. I want to dedicate these more personal lines to all of them, especially to my parents and grandparents.

I do not forget who I am, where I come from, or how I started; I know who has bet on me throughout all these years, who has given me their trust and who has supported me until today. To all of them, thank you. Thank you for not losing hope in me despite the mistakes I may have made or the not so good decisions I may have taken.

If I had to mention someone, it would certainly be my PhD supervisors, Carlos González and David Vallejo. For the last 6 years we have known each other, you have always done your best to give me every opportunity you could get and you have always been available for any problem I might have. I could call you friends, but I prefer to call you family.

Also, I would like to mention here Miguel Angel Redondo and Manuel Ortega, for offering me the possibility of sharing 3 years of my life in the research group CHICO, where I have been able to meet and work with real professionals from different areas while participating in the research project that has led to this doctoral dissertation.

I do not forget to thank Dr. Dorothy N. Monekosso for all her confidence in me and for opening the doors of the Leeds Beckett University to me.

Finally, I would like to finish this list by mentioning my partner; the last person who has appeared in my life and who has given me all her support in this final sprint. Thank you, Selma.

And I am not going to close this section of acknowledgement without mentioning Sully and thus closing a cycle, which repeats for the third time since the Final Degree Project in 2014 and the Final Master's Thesis in 2016.

To all of you, again thank you; thanks with all my heart.

Agradecimientos

Llegamos al final de este viaje en el que me han acompañado muchas personas; familiares, amigos, conocidos y compañeros. Quiero dedicar estas líneas más personales a todas ellas, especialmente, a mis padres y a mis abuelos.

No me olvido de quién soy, ni de dónde vengo, ni de cómo empecé; sé quien ha apostado por mí a lo largo de todos estos años, quien me ha brindado su confianza y quien me ha apoyado hasta el día de hoy. A todos ellos, gracias. Gracias por no perder la esperanza en mi pese a los errores que haya podido cometer o las decisiones no tan acertadas que haya podido tomar.

Si tuviera que mencionar a alguien, desde luego sería a mis directores de tesis, Carlos González y David Vallejo. Desde hace 6 años que nos conocemos, siempre habéis hecho lo imposible por brindarme todas las oportunidades que estuvieran a vuestro alcance y siempre habéis estado disponibles para cualquier problema que pudiera tener. Podría llamaros amigos, pero prefiero llamaros familia.

También, me gustaría mencionar aquí a Miguel Ángel Redondo y a Manuel Ortega, por ofrecerme la posibilidad de compartir 3 años de mi vida en el grupo de investigación CHICO, donde he podido conocer y trabajar con auténticos profesionales de distintas áreas mientras participaba en el proyecto de investigación que ha derivado en la presente tesis doctoral.

No me olvido de agradecer enormemente a la Dr. Dorothy N. Monekosso toda su confianza puesta en mi y por abrirme las puertas de la universidad de Leeds Beckett.

Por último, me gustaría terminar esta lista mencionando a mi pareja; la última persona que ha aparecido en mi vida y que me ha dado todo su apoyo en este *sprint* final. Gracias, Selma.

Y no voy a cerrar esta sección de agradecimientos sin mencionar a Sully y cerrar así un ciclo, que repite por tercera vez desde el Trabajo Fin de Grado en 2014 y el Trabajo Fin de Máster en 2016.

A todos vosotros, de nuevo gracias; gracias de todo corazón.

Contents

List of Figures

1 Introduction

The current doctoral dissertation is organized as a compendium of publications where it is offered a review of the results that are published in the indexed scientific journals that accompany this work.

Thus, this first chapter introduces a general overview of the problems, the context of the work and the motivation that make the research conducted in this doctoral dissertation arise. Next, the research questions and the objectives necessary to define a set of actions to answer these questions are formulated. Finally, a discussion of the results present in the main scientific publications is undertaken.

The rest of this doctoral dissertation is organized in two additional chapters. The chapter 2 shows the list of scientific publications that have derived in this doctoral dissertation, including information about each one of them and a brief review indicating how this publication has contributed to this doctoral dissertation, in the case of those publications not directly related to the research topic. Besides, in the chapter 3 the conclusions extracted from the results of these scientific publications are presented, giving answers to the research questions formulated in this chapter, as well as some future lines of work related to the topic of this doctoral dissertation.

1.1 Overview and context

1.1.1 General overview

Methods of learning through distributed real-time collaborative programming have been an important topic of discussion for a long time. The rise of tools

that allow users to work in such a way and the improvements achieved in high-speed networks demonstrate this. However, these new tools try to be too ambitious, building complete systems that offer users an integrated solution in a complete collaborative programming environment, without reaching an optimal result, as demonstrated throughout this section.

These results are achieved by centralizing efforts on low-level tasks such as project synchronization, more specifically that of individual files between collaborators. For example, previous work such as the CoEclipse tool [12, 13] focuses mainly on issues such as maintaining consistency between remote files, i.e. ensuring semantic and syntactic consistency, and not, however, encompassing a larger objective that completes a collaborative programming environment.

A common scenario of real-time collaborative programming is the massive open online courses (MOOCs), focused on software development, which reinforce the concept of remote work by encouraging students to collaborate with each other in order to successfully complete certain course activities. Existing research [52, 24] confirms that real-time collaborative programming is capable of accelerating the process of solving problems, facilitating the creation of better designs and achieving a shorter length in lines of code of the programs developed, thus achieving a higher quality of software projects.

A possible tool of this type would be COLLECE (**COLL**aborative **E**dition, **C**ompilation and **E**xecution of programs) [4], focused on guiding users to solve distributed and paired programming problems.

Among this type of tools, it is worth mentioning Saros [38], for the complete ecosystem it offers on the Eclipse platform. There are also alternatives for other environments that provide collaboration features such as VS Anywhere[1] for the Visual Studio integrated development environment

On the other hand, the rise of web technologies has led to the emergence of some cloud-based tools. Among the most popular are Kobra, CodeShare, or Cloud9 [7]. All of them allow the sharing of documents between users in a collaborative way, with some features such as syntax coloring, and communication via chat or video. However, they lack some capabilities of integrated development environments such as autocompletion, workspaces, or refactoring tools.

Apart from this type of tools, other authors use gamification techniques to improve the learning experience in children with satisfactory results [42, 10].

In the field of augmented reality (AR), there are studies that emphasize the value of this technology as a learning tool. In [20] they study the effects of this technique in the learning process of students, demonstrating the improvement of this form of teaching, with respect to other more traditional

[1] https://vs.componentsource.com/product/vs-anywhere-0

ones. Similarly, virtual reality also shows an upward trend in this area. In [6], for example, a virtual reality platform was developed to teach programming to engineering students.

On the other hand, the use of robots (either virtual or physical) can also facilitate this teaching of programming. In the last years there have been studies of this subject, demonstrating the positive results of this paradigm. Among these studies include [34] and [14], physical and virtual learning environments respectively, and oriented to learning programming in children and adults.

In conclusion, the use of immersive and emerging technologies provides a new paradigm of application in the field of learning programming, which is posed as a challenge to change the way in which new generations learn to program.

1.1.2 Work context

This doctoral dissertation is part of the "iProg: New generation of tools based on emerging interactive technologies for programming education" coordinated research project, specifically in the subproject with the same name as this dissertation with reference TIN2015-66731-C2-2-R. Funding for this project was obtained in a competitive public call in the State Program for Research, Development and Innovation Oriented to the Challenges of Society, within the framework of the State Plan for Scientific and Technical Research and Innovation 2013-2016.

In this sense, the coordinated project establishes a collaboration between the University of Castilla-La Mancha (UCLM) and the University Rey Juan Carlos (URJC), where the former has undertaken the tasks of development and implementation of various software prototypes and their evaluation at university educational levels, while the latter has been responsible for the evaluation of other tools developed at pre-university levels.

1.2 Background and motivation

Programming is one of the most important disciplines nowadays, both because of the cognitive benefits derived from its learning [43] and because of the high demand for professionals due to the so-called fourth industrial revolution [64]. According to the GlassDoor job portal [16], the latest 2020 reports reveal that 6 of the top 10 jobs in the U.S. required programming skills. By 2029, it is estimated that jobs demanding programming skills just in the STEM[2] fields are projected to grow 8%, thus outnumbering all the

[2]Science, technology, engineering and mathematics

other jobs that will grow a 3.4% [57]. Specifically, jobs directly related to the computer and mathematical field will grow by 12.1%, making it the third fastest growing area of employment by 2029, as shown in the chart in the figure 1.1. However, only 11% of graduates in the STEM areas do so in Computer Science [27]. Similar statistics are being reflected in Europe too, where the European Commision has made a move with initiatives for the learning of programming such as the *EU Code Week* and the *Opening Up Education*, among others [11].

In addition to meeting the needs of future demand for jobs related to digital technologies, programming has a direct impact on young learners by enhancing critical thinking, creative thinking and problem solving skills [63] through so-called computational thinking [62]. Thus, programming can be positioned itself as a transversal tool that facilitates the learning of other disciplines [59].

For all the above reasons, there is a clear need to include programming skills in the curriculum of future generations and to devise new teaching mechanisms that facilitate the learning of programming. In this sense, it would be advisable to include programming learning courses at the basic education levels (i.e. primary and secondary from 6 to 16-18 years-old in the Spanish education system, respectively), which use learning tools adapted to those educational levels. Similarly, the higher education levels (e.g. university) could also benefit from the advantages that programming brings, by improving the learning mechanisms of existing programming or even including them in those studies more distant from STEM disciplines that do not incorporate them. However, it is crucial to pay special attention to university studies in Computer Science, which allow the training of computer science professionals, and the difficulties and challenges they present before proposing new programming learning methodologies in other disciplines.

One of the main difficulties encountered by students in the first years of these studies is learning to program, whose statistics are directly reflected in the dropout rate of first-year students, averaging 51% [35]. One of the major causes of this is the abstraction that programming requires and the serious difficulties it poses for students [31, 3, 15]. For example, according to [33] the main problems are related to the incorrect use of syntactic structures (lack of parentheses, brackets and quotes) and strategic misunderstanding, for example, the choice of the least appropriate loop for the resolution of a given problem [53]. In this context, the application of group learning techniques [54] and the use of metaphors to facilitate the abstraction and understanding of concepts [55] can promote learning in this discipline by encouraging collaboration between students and properly managing the level of abstraction required to understand certain programming concepts, respectively.

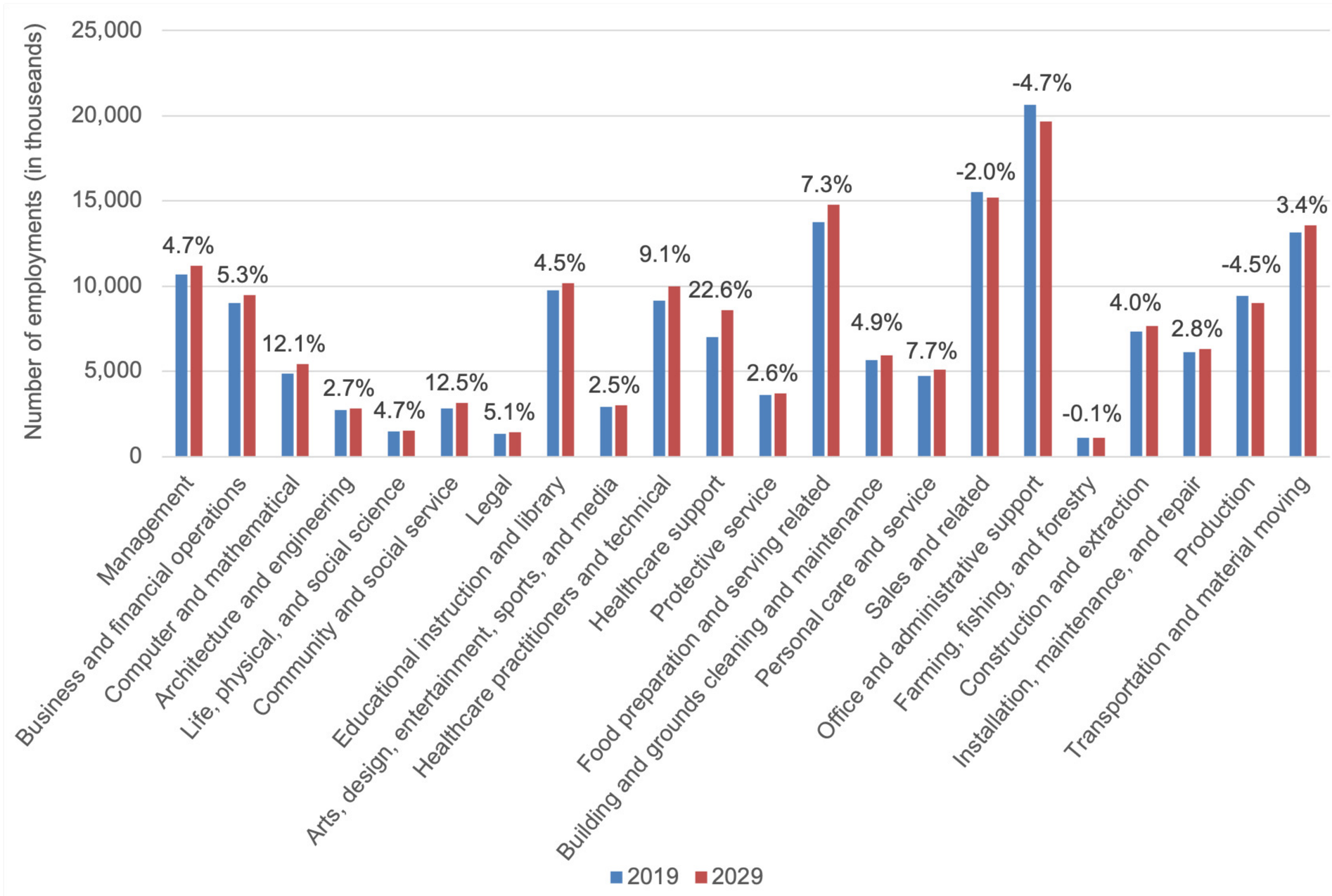


Figure 1.1: Occupational projections from 2019 to 2029 along with their percentages of increase/decrease in jobs. Source: U.S. Bureau of Labor Statistics (2019) [57].

1.2.1 Computer Supported Collaborative Learning (CSCL)

Group learning or CSCL (Computer Supported Collaborative Learning) consists of the application of group learning methodologies (teacher-students or groups of students) with computer support [22]. The CSCL allows the creation of learning spaces that promote the development of individual and group skills, and in which each member of the group is responsible for their own learning, but also the rest of the group. The CSCL can be considered a particular case of application of the principles of the CSCW (Computer Supported Collaborative Work) to the learning domain.

The CSCW is a set of techniques conducted to improve the interaction and collaborative work through the use of computer technologies, preferably of a distributed nature [41], whose final objective is the production of some artifact. The application of the CSCW principles to the field of programming would be related to the paradigm of Collaborative Programming, which is defined as the use of collaborative software to support programming activities, that is, to allow and facilitate that several programmers work together in the same algorithm and its corresponding code [28]. This technique favors the learning of programming, since the problems are solved in less time and the software obtained is of higher quality; in

addition, the students improve their communication skills, reducing their frustration and increasing their satisfaction [61].

1.2.2 Conceptual abstractions and analogies

To facilitate the understanding of programming concepts, we can establish a relationship between the concept to be explained and another concept already known, and less abstract, on which to support such an explanation. Thus, a metaphor is established that allows us to relate two concepts to facilitate the explanation of one of them [30]. By extending this approach to the field of graphic representations, we obtain visual metaphors that relate concepts to graphic content [25].

Thus, the difficulties that arise when learning to program can be addressed through the use of graphic representations that relate programming concepts with others with which students may be more familiar, thus building new metaphors that serve to define a mental model of higher level [19]. The use of graphic representations or 2-D visualizations brings many benefits to the learning process, such as an increase in the motivation and attention of the students [60], their participation in class [56] and the understanding of programming concepts [55], among others. It also makes complex programs more accessible and less intimidating [21, 9, 60].

Furthermore, visualizations based on 3-D graphic representations can bring certain benefits to the user, for example, exploiting the student's spatial memory capabilities to understand and help remember the structure of the programs [5], or displaying a greater amount of information from the program [37].

Using this three-dimensional space, it is possible to employ AR techniques to visualize programs and algorithms directly in the user's physical space and to facilitate natural interaction mechanisms (e.g. gestures and voice, among others), resulting in a more enriching experience for the student by stimulating the activation of brain structures and contributing to a more active learning [1], improving interaction capabilities, learning performance, motivation and collaboration [2, 8].

1.3 Hypothesis and objectives

This chapter introduces the research hypothesis on which this doctoral dissertation is based, formulated through a set of research questions. In a detailed way, these research questions are presented and answered throughout the research articles published and framed in the context of this

doctoral dissertation in the format of a compendium of publications. The research questions are shown in the section 1.3.1.

Once the research questions have been presented, the objectives of the doctoral dissertation are introduced in the section 1.3.2. These objectives are defined by a main objective, which is in turn divided into a set of more specific secondary objectives. These objectives are directly aligned with the research questions and arise as a result of defining a set of tasks or milestones to be completed to answer these research questions.

1.3.1 Research questions

The research questions formulated during the development of this doctoral dissertation serve to validate the contributions of the present dissertation. Thus, the aim is to present here, in a general way, the research questions that have been addressed in the different publications that accompany this work.

- **RQ**. Does the use of programming learning tools based on emerging technologies improve the overall learning experience?
  - **RQ1**. Does understanding and knowing the proposed tools and how they work intrinsically increase interest in programming?
  - **RQ2**. Does using inmersive interfaces, e.g. AR, improve motivation towards programming?
  - **RQ3.** Generally, what is the subjective perception of the ease of use, usefulness and intention of use of the proposed tools?

As already mentioned, these research questions are addressed, either explicitly or implicitly, throughout the research articles included in this doctoral dissertation and, specifically, in reference to the learning environments presented in each of these publications. In the chapter **??**, the results achieved in response to these research questions are discussed.

1.3.2 Main objective and secondary objectives

The main objective of this doctoral dissertation is to "*offer a set of tools and prototypes that facilitate the learning of programming and guide the complete formative process of students with different educational profiles through the use of immersive and emerging technologies that exploit new learning paradigms*".

Based on this objective, other more specific secondary objectives are determined that allow the tasks to be performed in a more constrained

context. The following sections detail each of these secondary objectives and the tasks to be performed to achieve them.

[Obj01] Review of the state of the art

Review of the state of the art regarding the present doctoral dissertation. This review will allow the acquisition of the existing knowledge in the related research lines here exposed, from other works and related tools that can serve for the elaboration of the present proposal.

The specific tasks to be conducted in order to achieve this secondary objective are the following:

- To perform a study of the state of the art by researching the literature on the following topics:
  - Learning tools for programming based on computer graphics and immersive technologies such as AR.
  - Visual analogies and metaphors that serve to represent programming concepts in a way that is more familiar to the user.
  - Existing intelligent tutoring systems oriented to facilitate the learning of programming.
  - Collaborative programming environments oriented to problem-solving.
- To identify potential improvements and innovation pathways in the topics researched.

[Obj02] Development of software systems

Development of advanced software systems for learning programming, oriented to different educational levels and with appropriate technological and didactic approaches.

The specific tasks to be conducted in order to achieve this secondary objective are the following:

- To develop a collaborative tool for remote work oriented to the learning of programming through problem solving that can be evaluated by first year Computer Science students.
- To develop a system of visualization and debugging of programs by means of AR that facilitates the understanding of programming algorithms.

- To develop a serious game for children that introduces them to basic programming concepts such as control structures, execution flow or loops.

[Obj03] Programming metaphors and other abstractions

Definition of a set of abstractions and visual metaphors that facilitate the learning of programming.

The specific tasks to be conducted in order to achieve this secondary objective are the following:

- To propose a series of visual analogies and metaphors that facilitate the understanding of general concepts of programming.
- To propose abstractions on specific programming concepts that help to visualize complex algorithms (such as those related to concurrent programming).

[Obj04] Use of immersive technologies

Integration of immersive and emerging technologies that pose a new paradigm of teaching programming.

The specific tasks to be conducted in order to achieve this secondary objective are the following:

- To explore AR devices that increase the visualization possibilities of the developed software systems.
- To integrate natural interaction mechanisms (such as gestures or voice commands) into the developed software systems

[Obj05] Learning environments

Introduction of each learning environment through use cases that allow to expose the potential of the developed proposals. These use cases highlight the flexibility of the proposals and their scope.

The specific tasks to be conducted in order to achieve this secondary objective are the following:

- To define a series of learning environments where the developed software systems will be used.

- To define a set of use cases aligned with the indicated learning environments.

[Obj06] Proposal evaluation and validation

Evaluation of the use cases presented with respect to different criteria.

The specific tasks to be conducted in order to achieve this secondary objective are the following:

- To undergo an evaluation that considers the intrinsic motivation of the students and the subjective perception regarding the usefulness and ease of use of the proposals.

- To validate the work by analyzing the evaluation results to answer the formulated research questions.

1.4 Work plan

As detailed in the chapter 1, this doctoral dissertation has been developed since September 2016 in the context of the national research project “iProg: New generation of tools based on emerging interactive technologies for programming education (TIN2015-66731-C2-2-R)” and with a view to being completed in an estimated time of 4 years.

During the development time of the doctoral dissertation, several factors have influenced its development, mainly the study and analysis of the existing literature in order to discover potential lines of innovation or improvement, and the constant communication with students with different educational profiles, who have actively participated in the different evaluations conducted to validate the different proposals. The results and comments obtained from these evaluations have served to iterate on the initial proposals, adding new ways of improvement and raising new cases of use.

For all this, a set of publications in indexed scientific journals and conferences both national and international has been obtained. The complete list of publications derived directly from this doctoral thesis will be presented and commented in the chapter 2.

The rest of this chapter presents the work plan followed to be able to perform this series of quick iterations and constant communication with the students and on which the bases of this doctoral dissertation have been built.

1.4.1 Materials

The following are some of the materials used during the development of this doctoral dissertation:

- Books from the UCLM library.
- Digital libraries accessible from UCLM such as Scopus, ScienceDirect, IEEE digital library and ACM digital library, among others..
- Software and hardware available in the laboratories of the CHICO (Computer-Human Interaction and Collaboration) research group.

1.4.2 Stages

First, the different phases undertaken during the work plan that would allow the achievement of the objectives presented in the chapter **??** are presented.

Planning, analysis and organization

In this phase the context of the problem is defined. To this end, a study of the literature is conducted on those scientific contributions that address all or part of any of the proposed objectives. This allows us to detect unresolved problems or unaddressed approaches, as well as to identify information that can serve as a basis for our research.

The achievement of this phase would allow us to reach the objective [Obj01].

Elaboration of the proposal

During this phase, the proposal that poses a solution to the main objective of this doctoral dissertation is established. This phase, in turn, is composed of a series of stages:

1. To select the development environment and the necessary tools (e.g. programming languages and *plug-ins*) for the development of the programming teaching systems.
2. To define the conceptual framework of our development method, establishing which fields will be covered, how the immersive technologies will be implemented, what kind of basic problems will be raised for the teaching of programming and on what environments will be performed.

3. To develop a common ecosystem to serve as a basis for the rest of the tools.

4. To provide a collaborative character to this ecosystem to allow the simultaneous and cooperative work of the students.

5. To prepare a set of visual abstractions and metaphors that serve to facilitate the understanding of specific programming concepts.

6. To define a set of learning environments and use cases where the different proposals will be integrated and evaluated.

The achievement of this phase would allow us to reach the objectives [Obj02], [Obj03], [Obj04] and [Obj05].

Proposal validation

The proposed solution must be verified by applying it to specific cases. To this end, experiments will be conducted with real students at different educational levels.

The achievement of this phase would allow us to reach the objective [Obj06].

Analysis and publication of results

To determine the degree of achievement of the objectives initially set, in order to assess and contrast the formulated research questions. Finally, to publish the results in proceedings of international congresses and scientific journals with impact index.

Elaboration of the doctoral dissertation

To formalize the results obtained throughout the research in the current doctoral dissertation.

Time planning

Temporarily, the work done for this doctoral dissertation began in September 2016 and lasted until December 2020. During this period, the phases mentioned above have been approached in a sequential and parallel manner, depending on the nature of the tasks to be performed.

Firstly, we began with the study of the literature and the current state of the art of the subject, to later proceed with the development of the first collaborative tool for teaching programming (COLLECE-2.0). The development of this tool was continued until December 2018, when it was finalized and ready to be used in a real environment.

Previously, in February 2018, work had already begun on the set of visual metaphors that would give rise to the ANGELA notation of roads and traffic signs. This notation would be continuously improved over several iterations until July 2020, when the notation was already in a sufficiently mature state to be used in different learning environments and use cases.

In parallel, and since January 2018, the evaluation phase with students began. The results obtained from this phase would have a direct impact on the iterations performed on the ANGELA notation and on COLLECE-2.0. Similarly, these results, along with the software architectures of COLLECE-2.0 and the ANGELA notation, were published in multiple national and international conferences, as detailed in the chapter 2.

Additionally, between September and December 2018 a research stay was held at the Leeds Beckett University (United Kingdom) which served to consolidate knowledge in different lines of research and which would directly affect the development of this doctoral dissertation. In the section 1.4.3 this research stay is detailed.

Finally, from February 2020 onwards, the complete results of the research began to be published in indexed scientific journals, leading to the writing of this doctoral dissertation in September of the same year.

The Gantt diagram in the figure 1.2 shows the time planning, including the phases of work mentioned above, in a summarized way.

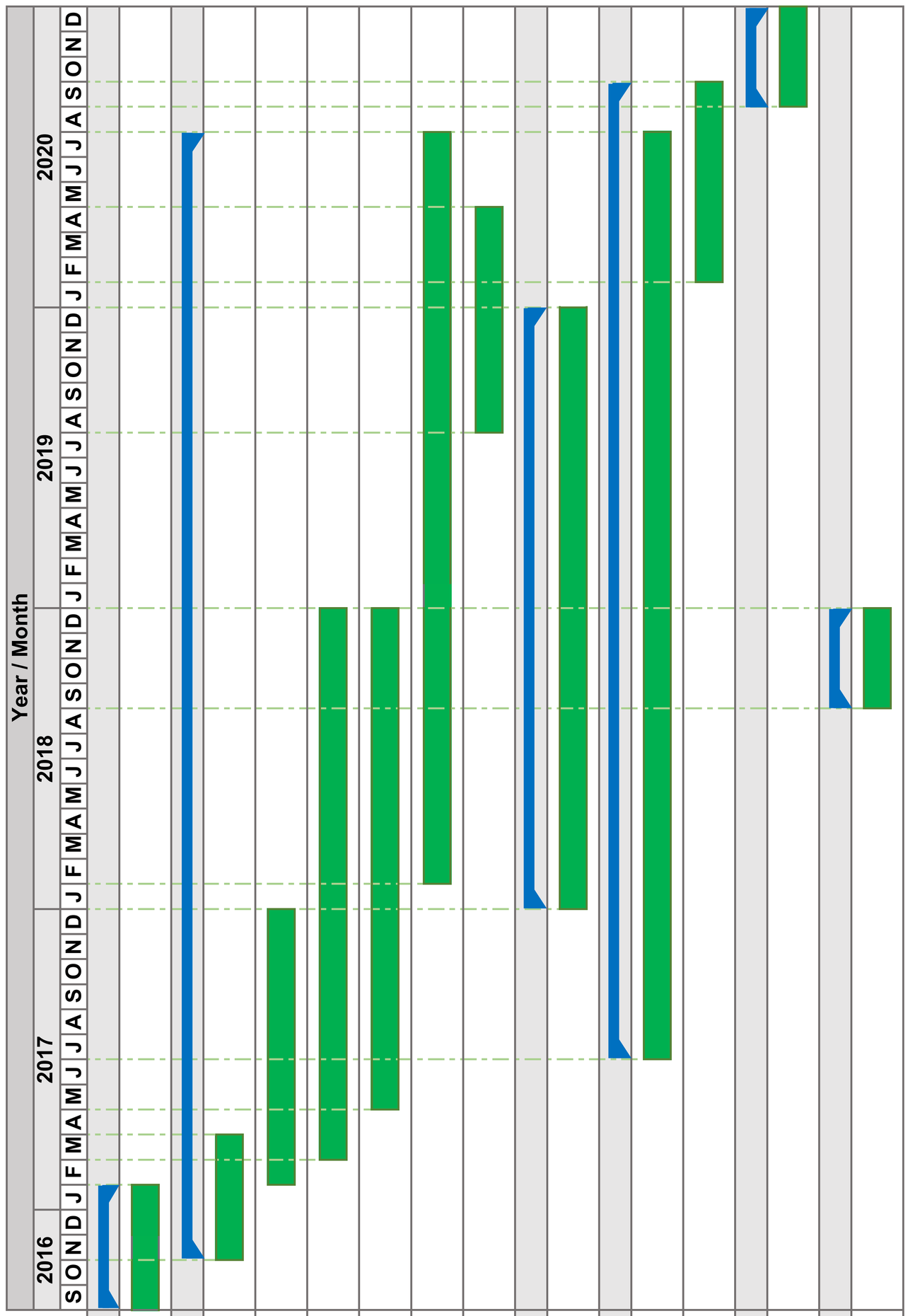


Figure 1.2: Gantt chart of the work plan.

1.4.3 Mobility plan

From 17th September 2018 to 17th December 2018 (3 months) a research stay was held at the Leeds Beckett University, specifically at the Digital Health And Assistive Technology (DHAT) Research and Innovation hub of the School of Computing, Creative Technologies & Engineering, under the supervision of Prof. Dr Dorothy N. Monekosso and with the aim of applying for the International Doctorate mention of the doctoral program related to this doctoral dissertation.

During this stay a participation was made in the MONICA project[3] (Management Of Networked IoT Wearables - Very Large Scale Demonstration of Cultural Societal Applications) from the European Union's Horizon 2020 research and innovation programme and in a telemedicine project aimed at the rehabilitation of stroke survivors[4].

There was also the possibility of meeting different doctors in very diverse lines of research that allowed to gather opinions, comments and experiences about the development of the present doctoral dissertation. Special mention to the members of the DHAT group, specifically Dr Akbar Sheikh-Akbari, for the valuable feedback provided on the line of research of this doctoral dissertation, to Dr Sareen Galbraith for sharing her research and work on technologies to support the diagnosis of dementia in people affected by Parkinson's and to Dr Rasha Ibrahim for her recommendations when evaluating proposals with real users. Likewise, Sanela Lazarevski, lecturer at the Leeds Beckett University, and Dr Angel Jimenez Aranda, Digital Technology Lead at Devices for Dignity (Sheffield), are also mentioned for the exchange of comments on the subject of learning programming during the research stay. These experiences have facilitated the improvement of the quality of the present doctoral dissertation, the widening of the horizon of knowledge related to this research and the capacity of the present PhD candidate to conduct research.

On the other hand, thanks to the research stay, it was possible to participate in the above-mentioned projects, which allowed to acquire experience working in a multidisciplinary and international environment, as well as a greater knowledge about Internet of Things (IoT) and AR immersive devices, and their possibilities in a context of learning programming.

Also, during this stay the results of the advanced work were published in different international conferences and several evaluation models were prepared with users that would serve to validate the work developed later.

Finally, the advanced work on the international projects was published in the proceedings of the 2nd International Conference on Multimedia Analysis and Pattern Recognition (MAPR) [44] and in volume 8 of the journal IEEE

[3] https://www.monica-project.eu
[4] https://www.virtualphysioproject.com

Access [45]. The chapter 2 includes more information on these articles and on the rest of the research articles published during the completion of this doctoral dissertation.

1.5 General discussion

This section discusses the results obtained from the main publications in scientific journals accompanying this doctoral dissertation and other relevant publications in international congress proceedings available in the chapter 2.

The results obtained from the publications are directly aligned with the objectives proposed in this doctoral dissertation. Thus, along the three main publications a diverse set of results is produced including: i) the introduction of a real-time collaborative programming environment called COLLECE-2.0, ii) a visual notation based on the metaphor of roads and traffic signs to represent concepts and programming problems, iii) a serious game oriented to introduce basic programming concepts to children, iv) a set of use cases that integrate these environments, and v) different evaluations with real users that seek to validate the usefulness of these environments.

A brief discussion of the results obtained from these scientific publications is introduced below.

1.5.1 COLLECE-2.0 platform

COLLECE-2.0 [39] is a collaborative learning platform aimed at solving group problems through working sessions and programming projects that challenge participants with the aim of improving their learning.

This platform initially emerged as an evolution of the COLLECE [4] system, a synchronous turn-based programming environment that allowed editing, compiling and executing programs implemented in the Java language between several participants remotely.

The difference between COLLECE-2.0 and its predecessor lies in the capacity of asynchronous editing of programming projects, support for multiple programming languages and the introduction of different *awareness* mechanisms, among others. The main characteristics of the platform are indicated below:

- **Modular**. The tool is easily installed in the Eclipse IDE as a plugin through its repository system.

- **Extensible**. An external API is offered publicly to be consumed by other Eclipse plugins.
- **Work sessions**. The system integrates a session management tool to create public or private sessions over the network for other users to connect and work simultaneously.
- **Problem-solving oriented**. The users collaborate toward a common solution for the given problem that has to be solved when connecting to a COLLECE-2.0 session.
- **Multi-user edition**. COLLECE-2.0 provides a low-latency, real-time collaborative editing system that allows for fluid editing, so that multiple users can edit the same files concurrently.
- **Repositories**. COLLECE-2.0 uses remote repositories to synchronize and persist the state of the different projects. Thus, users can make use of this mechanism to share projects with other users and leverage the features of the tool.
- **Awareness mechanisms**. The tool includes different awareness mechanisms which can be customized in the Eclipse work view: i) tele-pointers, to identify who is editing and where in the code they are doing it; ii) blocked regions, to enable users to block parts of the code from being edited by other session members; iii) blocked regions panel, to manage all the existing blocked regions in a file and their owners; iv) instant messaging with public support for all users in the session, or privately with a specific user; v) panel of connected users with identifying colors for each user; vi) statement of the problem that has to be solved.

At research level, COLLECE-2.0 allows to establish a common working environment in the context of a tool for the resolution of group programming problems with support to be extended by the use of *plug-ins*. Thanks to this extensibility, this platform has been proposed as a foundation to be used in other learning environments oriented to the visualization of programs through the ANGELA notation, which will be introduced in the following section.

The platform is thus established as a complete evaluation environment that can be used to study the impact that this type of environment has during the learning of programming through the analysis of different dimensions.

Relevant publications

The most relevant scientific publications related to COLLECE-2.0 are:

- S. Sánchez, M. Á. García, C. Lacave, A. I. Molina, C. González, D. Vallejo, and M. Á. Redondo. A modern approach to supporting program visualization: from a 2d notation to 3d representations using augmented reality [48]. This publication covers two well differentiated results: COLLECE-2.0 and the graphic notation ANGELA. With regard to this section, this publication introduced COLLECE-2.0 and the evolution it represented with respect to its first version.

- C. Lacave, M. A. García, A. I. Molina, S. Sánchez, M. A. Redondo, and M. Ortega. COLLECE-2.0: A real-time collaborative programming system on Eclipse [23]. This publication arises as a result of a pilot evaluation of the COLLECE-2.0 platform with 37 second and third year computer engineering students with the aim of assessing the ease of use and usefulness of the built-in awareness mechanisms, as well as their level of support by the solution.

1.5.2 ANGELA graphic notation

ANGELA (not**A**tio**N** of road si**G**ns to facilitat**E** the **L**earning of progr**A**mming) is a graphic notation based on the metaphor of roads and traffic signs that establishes an analogy between real world concepts and different aspects of programming, trying to reduce the complexity linked to the task of programming, specifically with the understanding of the programs, which should result in the decrease of the difficulty in learning. The representations can be visualized both in 2-D and 3-D, and even by an AR device that uses the space of the physical environment of the user to locate and align the visualization in real time. The construction of these visualizations (sets of graphic representations) is done automatically by implementing an Eclipse plug-in integrated in COLLECE-2.0, thus providing this environment with new program visualization capabilities.

The notation has been used in the fields of static visualization, dynamic visualization, and visualization of concurrent programming concepts. It has also been used in other serious game proposals, as will be seen in the next section.

In the field of static visualization, the notation is used to represent the structure and sentences of a program, so that it is possible to easily identify the elements that compose a program.

In the case of dynamic visualization, to the above is added a set of interactive elements that allow to simulate the execution of a program step

by step on these visualizations. In this case, the COLLECE-2.0 plugin that supports this type of visualizations allows to generate the dynamic visualization and to offer different mechanisms of interaction that allow to interact with it as if it was a software debugger. Thus, using this type of visualization, it is possible to advance the program step by step, the complete execution without waiting, the definition of breakpoints to pause the execution in a given sentence and the possibility of examining the state of the memory (e.g. value of variables and evaluation of expressions, among others) up to that point of execution.

Additionally, the notation can be used to visualize concepts of concurrent programming. In this sense, the COLLECE-2.0 plugin should be adapted to support this new need. Even so, and in a preliminary way, a set of graphic representations has been produced to visualize different concurrent programming concepts, such as semaphores and signaling, critical memory regions and execution threads. Using this set of graphical representations, it is possible to generate visualizations for more complex synchronization patterns such as the barrier synchronization pattern[5] [18].

Relevant publications

The most relevant scientific publications related to the ANGELA notation are:

- S. Sánchez, M. Á. García, C. Lacave, A. I. Molina, C. González, D. Vallejo, and M. Á. Redondo. A modern approach to supporting program visualization: from a 2d notation to 3d representations using augmented reality [48]. This publication covers two well differentiated results: COLLECE-2.0 and the graphic notation ANGELA. With regard to this section, this publication introduced the evolution of ANGELA notation from its initial 2-D version to its 3-D representation including an integration with COLLECE-2.0 to represent the visualizations using AR. In this sense, the published results are based on a set of evaluations with students that allow validating the dimensions of understanding and suitability of each of the graphic elements proposed to represent the different programming sentences and control structures using the notation. The analysis of each of these evaluations has motivated the improvements made to the notation and its evolution over time.

- S. Sánchez, C. Gómez, D. Vallejo, C. González, and M. Á. Redondo. An intelligent tutoring system to facilitate the learning of programming through the usage of dynamic graphic visualizations [49]. This publication is based on the work done by the previous one formalizing the notation and providing new improvements. The improvements are

[5]Video showing the visualization of the barrier synchronization pattern: `https://www.youtube.com/watch?v=66zaRSIjbPA`

based on extending the new version of the notation with support for dynamic visualization of programs through debugging in different visualization environments.

1.5.3 Serious game RoboTIC

RoboTIC is a serious game that arises with the aim of proposing new learning mechanisms aimed at improving children's interest in programming. The proposal is based on the use of the ANGELA notation as a basis for representing the execution of the game and immersive AR devices that enable the use of natural interaction mechanisms.

The game poses, at each level, that the player solves a challenge through interaction by gestures, indicating to the agent how to reach the objective of the scenario through a series of actions. As in the previous case, this scenario of AR is also built on a surface of the real world, from a set of blocks where the agent moves in a discreet way.

The actions that the agent can perform come indicated by the user in form of commands, which are executed by a vehicle in reduced size, which moves over the visualization at the same pace that the agent does over set of blocks.

Relevant publications

The most relevant scientific publications related to the serious game RoboTIC are:

- S. Sánchez, D. Vallejo, C. González, M. Á. Redondo, and J. J. Castro. Robotic: A serious game based on augmented reality for learning programming [51]. This publication proposes a serious game that facilitates the introduction of programming concepts in children. To this end, the proposal takes advantage of the research conducted in the area of program visualization to use an adapted version of the ANGELA notation on which the proposed serious game is based. The proposal has been evaluated with a small group of children (12) with the objective of analyzing the intrinsic motivation of the participants, their subjective opinion about programming and the qualitative perception of the proposal.

1.5.4 Learning environments

The proposals named so far can be used in different learning environments, as it is gathered along the publications mentioned above and introduced as specific use cases of the proposals:

- Use of the ANGELA notation in the classroom, where the teacher would load the visualization of a program onto the physical space of the classroom to explain specific concepts of program structure and operation to the students. These students would have the possibility to interact with the visualization thanks to their immersive devices that share the same visualization as the teacher.
- Active debugging environments through the combination of COLLECE-2.0 and the ANGELA notation, to facilitate the analysis of program execution visually.
- Game oriented learning environments using RoboTIC that introduce programming concepts to children.
- Teaching concurrent programming concepts in the classroom by creating a set of visualizations that make use of the ANGELA notation to complement the teacher's explanations.

2

Results

As mentioned above, the present doctoral dissertation is constituted as a compendium of three research articles. These articles are included in this chapter, together with their main metadata, as well as other research articles published in proceedings of international conferences and related to the main research topic.

Apart from this, other research articles are also cited, which are less directly related to the objectives of this doctoral dissertation, but which have emerged as a result of other work and collaborations. In this case, their metadata, the abstract and some of the most representative images are included. In addition, a brief review of them is also included, justifying their impact on the work done to complete this doctoral dissertation.

2.1 Articles published in indexed scientific journals included in the compendium

2.1.1 An Intelligent Tutoring System to Facilitate the Learning of Programming through the Usage of Dynamic Graphic Visualizations

- Title: An Intelligent Tutoring System to Facilitate the Learning of Programming through the Usage of Dynamic Graphic Visualizations [49]

- Authors: Santiago Sánchez, Cristian Gómez, David Vallejo, Carlos González, and Miguel Ángel Redondo

- Type: Journal

- Journal: Applied Sciences
- Publisher: Applied Sciences (Basel, Switzerland)
- E-ISSN: 2076-3417
- Year: 2020
- DOI: 10.3390/app10041518
- Category: Engineering, Multidisciplinary
- Impact Factor (2019): 2.474
- JCR Ranking: Q2
- Related to the current research topic: Yes

applied sciences

MDPI

Article

An Intelligent Tutoring System to Facilitate the Learning of Programming through the Usage of Dynamic Graphic Visualizations

Santiago Schez-Sobrino *, Cristian Gmez-Portes, David Vallejo, Carlos Glez-Morcillo and Miguel Á. Redondo

Department of Technologies and Information Systems, University of Castilla-La Mancha, Paseo de la Universidad 4, 13071 Ciudad Real, Spain; Cristian.Gomez@uclm.es (C.G.-P.); David.Vallejo@uclm.es (D.V.); Carlos.Gonzalez@uclm.es (C.G.-M.); Miguel.Redondo@uclm.es (M.Á.R.)

* Correspondence: Santiago.Sanchez@uclm.es; Tel.: +34-926-295-300

Received: 29 December 2019; Accepted: 19 February 2020; Published: 23 February 2020

check for updates

Featured Application: The proposal of this work focuses on offering an ITS that can be applied in programming learning environments (i.e., university degrees, schools, academies, etc.) with the objective of introducing programming transversally in different application areas and offering a complementary mechanism that facilitates its learning.

Abstract: The learning of programming is a field of research with relevant studies and publications for more than 25 years. Since its inception, it has been shown that its difficulty lies in the high level of abstraction required to understand certain programming concepts. However, this level can be reduced by using tools and graphic representations that motivate students and facilitate their understanding, associating real-world elements with specific programming concepts. Thus, this paper proposes the use of an intelligent tutoring system (ITS) that helps during the learning of programming by using a notation based on a metaphor of roads and traffic signs represented by 3D graphics in an augmented reality (AR) environment. These graphic visualizations can be generated automatically from the source code of the programs thanks to the modular and scalable design of the system. Students can use them by leveraging the available feedback system, and teachers can also use them in order to explain programming concepts during the classes. This work highlights the flexibility and extensibility of the proposal through its application in different use cases that we have selected as examples to show how the system could be exploited in a multitude of real learning scenarios.

Keywords: intelligent tutoring system (ITS); program visualization; algorithm visualization; programming learning; augmented reality; metaphors

1. Introduction

Artificial intelligence (AI) has a brilliant present and future. In fact, there are many areas of application today with solutions to problems that society demands and that are of great interest [1]. Among them, there are sophisticated systems focused on health, driving, robotics, or even video games. A particular case in which AI has considerable importance is education, with one of its highest priority objectives the elimination of the traditional educational model currently in place [2]. This approach focuses mainly on applying cutting-edge technology to develop intelligent learning systems that simulate the interaction between the teacher and the students or that are of great value in conducting knowledge with ease.

In this context, one of the disciplines with a relevant impact is programming, as it is considered necessary to learn due to its transcendence in professional and educational contexts. In economic terms, the number of jobs that require programming knowledge is increasing [3]. The European Commission estimates that over 90% of current vacancies require programming skills and that by 2020, there are expected to be approximately 825,000 vacancies that will need to be filled in Europe [4]. From the point of view of education, programming is proposed as a transversal tool that serves as a vehicle to promote computational thinking and to work on content from other subjects [5]. Hence, the aim is to tackle problems from childhood, reinforcing concepts and developing new mental models [6]. There is, therefore, a clear need to train new professionals to cover the current demand for programmers.

However, programming poses multiple difficulties for the students who are beginning to learn it. The most important ones are the usage of variables, the comprehension of control structures, the code modularization through functions, the handling of lists, or the correction of syntactic errors, among others [7,8]. Specifically, much of this content involves a degree of abstraction in the students that they still do not have because they do not understand the real effect that it could have to change the source code of a program on its execution or because they do not provide proper solutions to problems using a programming language.

There are numerous advantages of using intelligent tutoring systems (ITS) or assistants as support tools in the context of learning how to program, since they are directed towards the development of more effective personalized teaching and learning processes [9,10]. From the perspective of the role of the teachers, the development of this type of instrument involves powerful possibilities for them, among which the reduction in time for designing content and promoting appropriate knowledge. On the other hand, from the point of view of the students, learning takes place in an autonomous way, as these systems are designed with the purpose of complementing, and not replacing, the role of the teacher.

To this approach, we can add the use of graphic representations that relate programming concepts to others with which the students may be more familiar [11]. Thus, the conversion of more complex programs into others that are more accessible and less intimidating is possible. The use of 2D visualizations has a positive effect on the learning process, from an increase in the motivation [12] and participation of the students in the class [13] to the comprehension of programming concepts [14]. Alternatively, the use of 3D graphic representations provides multiple benefits to the user, from the use of three-dimensional spaces to take advantage of spatial memory capacities, in order to understand and help to remember the structure of the programs [15], to visualizing a bigger amount of program information [16], among other possibilities. Taking advantage of the use of this type of space, we can make use of augmented reality (AR) techniques so as to visualize programs and algorithms more naturally, leading to a more fulfilling experience for the student by stimulating the activation of their brain structures or contributing to a more active learning [17].

Unfortunately, the literature lacks tutors or assistants (ITS) who present intelligent behaviors while using a representation that reduces the abstraction or complexity that a task requires. Therefore, in order to fill this gap and to improve the understanding of learning to program, we established an intelligent system to help to program by means of autonomous dynamic visualizations that guide the student during the execution of a program. The tool converts the source code of a program into a less abstract and intimidating representation, which facilitates the tracking of its trace by leaning on a computer support that generates automatic visualizations. A general scheme of the system is shown in Figure 1.

The graphic representations generated are based on a metaphor of roads and traffic signs, which have been called ANGELA (i.e., "notAtioN of road siGns to facilitatE the Learning of prograMming"), a set of designs that establish a relationship of correspondence between programming concepts and road traffic elements. Thus, we try to define a mental model at a higher level with which to reduce the abstraction required by programming.

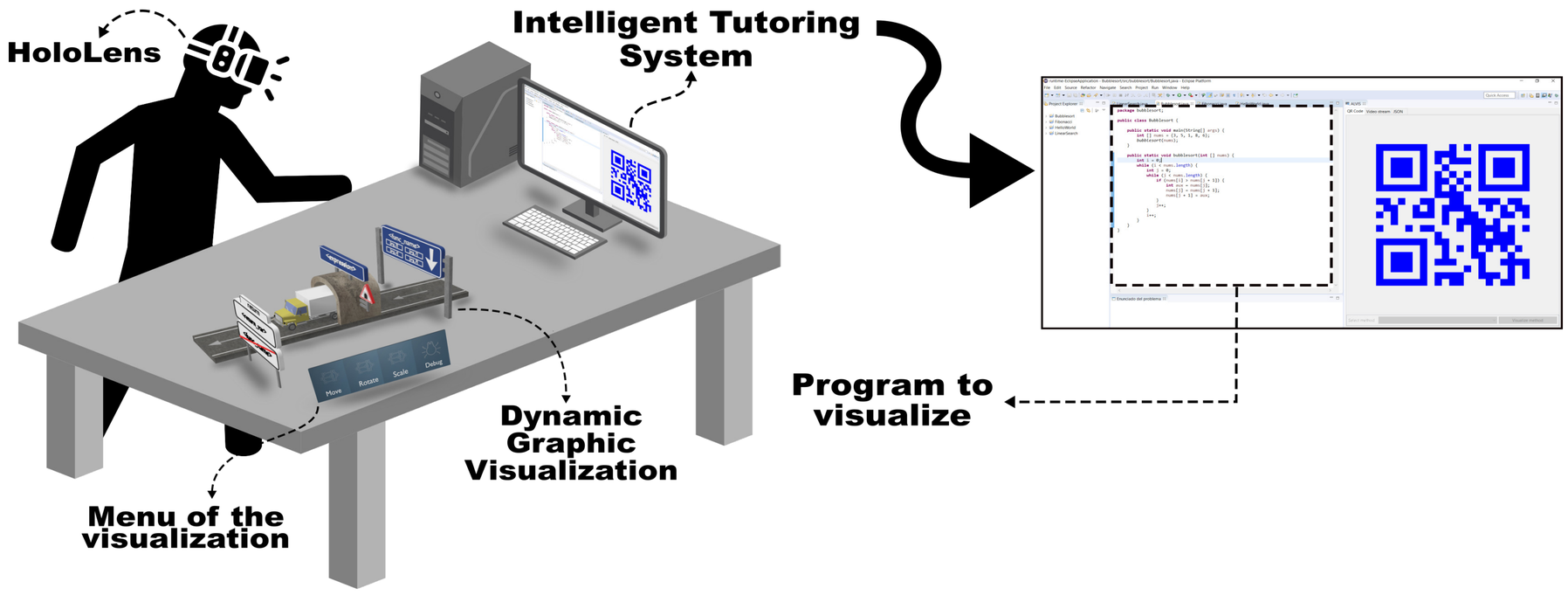


Figure 1. General overview of the proposed intelligent tutoring system (ITS).

The rest of the article is structured as follows: Section 2 presents some intelligent assistants (ITS) in the context of learning to program; Section 3 focuses on the proposal of this work, presents the ITS and the related integrated notation, and explains the automatic generation of dynamic visualizations; Section 4 describes a set of use cases that attempt to demonstrate the extensibility and flexibility of the proposal; finally, Section 5 details the conclusions and future lines of research.

2. Related Work

The existing literature presents tools to address the problem of learning to program. In this section, we review some of them focusing on both the context of intelligent assistants or tutors (ITS) and software visualization systems.

ITS is a system designed to replicate the work of teachers automatically. The goal of these systems is to provide individualized learning such as the one that a student might have by attending private classes. As described in [18], these systems are characterized by providing personalized feedback to each student, usually without requiring the intervention of a human teacher. Bearing this in mind, there are different works in the context of learning to program that are of interest.

The work presented in [19] describes an intelligent system to teach a part of the Java programming language, offering the students feedback of their solutions. Similarly, the work in [20] presented a system designed to learn how to program in Prolog language, guiding the student through an exercise. In the context of teaching web programming, the work in [21] showed a system for learning PHP, whose feature was to provide personalized feedback to the students according to their solutions, whatever they may be. Another approach was the one presented in [22], which showed a system that helped to solve custom programming exercises based on flowcharts by making decisions.

Over the years, several authors have attempted to give evidence of the benefits that ITS provides to the students. Several works conducted a literature review where they collected the results after evaluating an ITS whose conclusions confirmed the effectiveness of using this type of tool in the classroom [10,23]. However, there are studies that showed a skeptical position about the effectiveness of providing feedback to the students during the learning process. For example, in [24], it was found that the quality of feedback could lead to negative emotions such as boredom or frustration. Similarly, the work presented in [25] raised a debate about the timing of providing feedback, as this could lead to a better or worse understanding of what has been studied.

With respect to software visualization systems, there are many of them with different purposes: some focused on showing control structures and programming concepts in a user-friendly way (static visualization), and others focused on showing the trace of a running program through animations (dynamic visualization). In this last case, different systems stand out because of the graphic representations that they use.

The work in [26] described a system to represent graphically the concurrent behavior of programs using a metaphor based on cities. Along these lines, the work in [26] showed an application that used

techniques based on gamification to present puzzles in which some elements of the programming languages were treated in the form of growing plants, such as loops and iterations. Likewise, the work in [27] described a system that used avatars and 3D animations to represent programming concepts, for example exchanging messages between objects, among others. Furthermore, although more abstract, the work in [28] presented a system that helped to understand concepts related to concurrent programming through the use of characters and 3D boxes.

The literature also evaluated the effectiveness of using program visualizations and algorithms in the context of learning to program. A variety of papers presented results of tests and experiments in favor and against. Among the publications supporting this, it is noteworthy to mention the findings in [29,30], where a number of tests were held with the students who were able to demonstrate a better comprehension of the elements of the programming languages and the proposed algorithms. In contrast, the results presented in [31,32] offered a more controversial view, highlighting the use of visualizations, rather than its quality, and especially the fact that it is difficult for the teacher to create them and to keep the students engaged during classes. Besides the selection of suitable metaphors, the tools that support both the visualization and the interaction are just as important. In [33], several experiments were performed with students to discover the difficulties they had in trying to understand the problems related to recursive programming and concluded that when they used tools that helped them to visualize the behavior of the program, their ability to complete the exercises improved as it became more dynamic. Furthermore, in [34], there was a review of other related tools that use visualizations to help with the teaching of programming with successful results.

Furthermore, the effectivity of the visualizations can be enhanced by using AR techniques, as described in [35], which concluded with certain benefits that could help the process of learning to program, including improvements in learning, an increase in motivation, and easiness in interacting with one's own visualizations and collaborating with other learners; for example, the one presented in [36], in which paper based markers were used to help to teach the concepts of computer graphics programming using AR and the OpenGL 3D graphic specification; and the one presented in [37], in which the same system was assessed with the students who achieved favorable results in encouraging them to learn and enhance their motivation.

The differentiating feature of this work is the integration of an intelligent behavior into a software visualization system, which takes special care to provide feedback to the student. This work also presents another singularity: the use of AR as a visualization technology, which can be leveraged to generate learning environments on a real environment, while improving the realism of the analogy with the physical elements that it represents. Thus, we try, on the one hand, to enable the autonomous learning of the students and, on the other hand, to capture and maintain their attention and motivation.

3. Proposed ITS and Notation of Road and Traffic Signs

The system proposed in this paper is intended to facilitate the introduction of programming to university students who are just beginning to program and who lack sufficient knowledge to solve problems through some programming language. Mainly, this situation is linked to the abstraction required to understand the programming concepts that are taught, to the lack of knowledge of the language, or to the difficulty to design and build algorithms [7]. Thus, the proposed ITS integrates the use of metaphors that allow us to reduce this level of abstraction by including analogies between concepts from the real world and aspects of programming, trying to reduce the complexity linked to the task of programming. This analogy establishes a notation with roads and traffic signs that we have called ANGELA, which represents the structure and sentences that compose a program (static visualization), going through the representation by means of a vehicle that goes through its structure, and executing these sentences (dynamic visualization). This metaphor is intended to be motivating and easy to understand, as it includes familiar elements from the daily lives of the students. Thus, a direct analogy is established between the graphic representations in the metaphor and the sentences defined in the Java programming language, whose choice is due not only to the popularity in the professional

field, but also in the educational one [38]. The statements defined allowed us to represent graphically the function definition, the condition statement, the loop statement, the expression evaluation, and the function return. In addition, a representation was added to this set for the breakpoints and another one to simulate the execution thread of a program in order to include dynamic visualization mechanisms in the system. It is interesting to note that this set did not distinguish between different types of loops (e.g., while, for, or do-while), as well as with types of conditions (e.g., if-then-else or switch-case), simplifying the construction of the visualizations to a reduced set of graphic elements that were easily understood by the students. Figure 2 shows the set of elements that are part of the notation:

- Function definition (Figure 2a): It provides information related to the name of the function and its input arguments. This is the first graphic representation with which a full visualization must begin. The arrow of the traffic sign includes the reading direction of the visualization.
- Conditional statement (Figure 2b): It provides information related to the condition. This condition will then be used to set the new execution flow that will be followed by the program; if the condition is met, the program will execute its left side, otherwise its right side.
- Loop statement (Figure 2c): It provides information about the condition to be met to enter the loop. If the condition is fulfilled, the execution flow will be driven to the right road of the graphic representation, running all the sentences on such side. Otherwise, the execution flow will be driven through the left road. In both cases, the arrows on the road show the direction to be followed. Using a roundabout, we make an analogy with the concept of repetition.
- Function return (Figure 2d): It provides information related to the function that is finished including its name and variables to be returned.
- Evaluation of expressions (Figure 2e): It provides information related to arbitrary expressions such as assignments, function calls, etc. We use a tunnel to illustrate the evaluation of the expression when the vehicle goes through it.
- Breakpoint (Figure 2f): It provides a mechanism to stop the program execution as if it is a debugger. In this case, and with the aim of being integrated with the rest of the metaphor, we use a traffic cone.
- Thread (Figure 2g): It provides information about the current execution point of the program, allowing the user to explore the program through a functionality similar to that of a debugger. The graphic representation used, a vehicle, would go through the graphic representations step-by-step executing the related sentences. Multiple vehicles going through the visualization would represent different threads executing the same function.

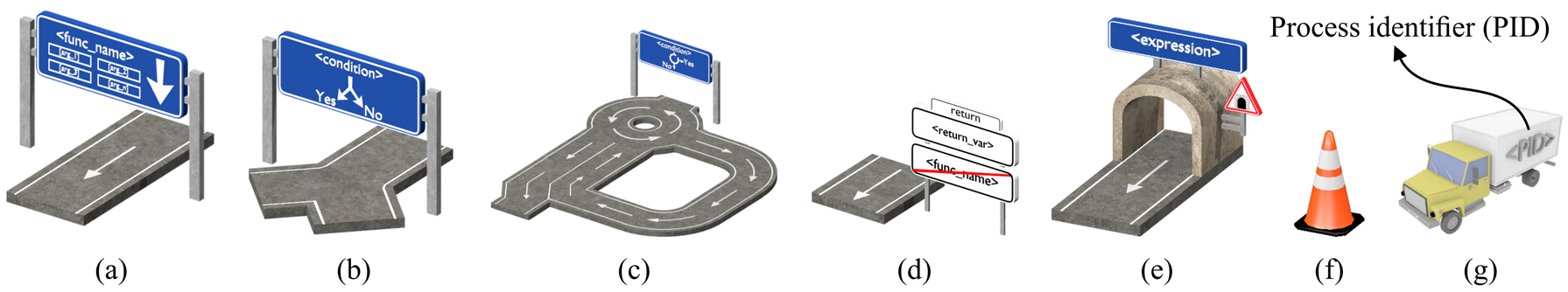


Figure 2. Graphic representations of notAtioN of road siGns to facilitatE the Learning of prograMming (ANGELA) converted to an orthogonally projected 3D space. (**a**) Function definition. (**b**) Conditional statement. (**c**) Loop statement. (**d**) Function return. (**e**) Evaluation of expressions. (**f**) Breakpoint. (**g**) Thread.

The decisions about the design in the graphic representations were inspired by the "Vienna Convention on Road Signs and Signal" (1968) [39] whose proposals were adopted internationally by most countries. This presented a direct advantage to those students who were obtaining a driving license, since they were already familiar with such designs.

These visualizations could be displayed through an AR device. Among the state-of-the-art devices, Microsoft HoloLens excelled thanks to the immersion and interaction capabilities that it

offered. This way, students could benefit from using the ITS through a natural interface more intuitive for them.

The system facilitated the construction of the visualizations by generating them automatically from the source code of the programs. The visualizations generated were then sent to the AR device, which acted as a viewer that received the necessary information about the visualization from the ITS through the network. For this, from the source code of the program, a simplified hierarchy was generated in the JavaScript Object Notation (JSON (data interchange format using text to transmit data consisting of attribute-value pair [40])) format that could be easily processed by the AR device and that contained all the information necessary to construct the visualization. To do this, the ITS analyzed the abstract syntax tree (AST) generated from the source code and identified each of the language elements related to the proposed graphic representations (see Figure 2). The hierarchy generated in JSON format kept each of these associations in a tree-like form, along with the connectors needed to construct the visualization in a linked way. Figure 3 shows an example of the simplified sub-hierarchies generated for some of the proposed graphic representations. The final visualization would comprise the complete hierarchy from these sub-hierarchies.

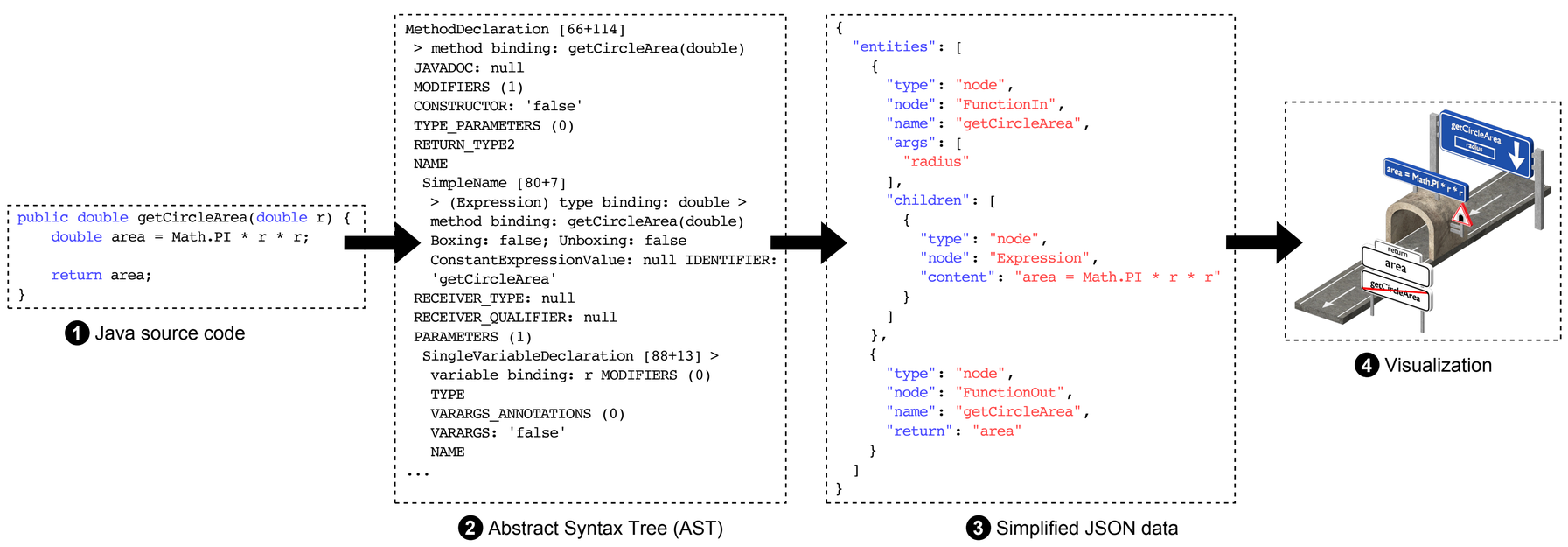


Figure 3. Generation of the JSON structure with the relevant information used to build the visualization from the source code of the program.

For the dynamic visualization mechanisms, the ITS was responsible for intercepting calls to the chosen debugger and sent this information to the visualization device in order to animate the movement of the vehicle through the graphic representations. On the side of the device, the interaction made by the user was sent to the ITS to manipulate the debugger status (i.e., advance the program execution, pause it, etc.).

Currently, the ITS implementation is being conducted through a plug-in implemented for COLLECE-2.0 [41,42], a collaborative distributed system for learning to program based on the Eclipse development environment. This plug-in utilizes the COLLECE-2.0 infrastructure, performing the tasks mentioned above, to generate the JSON from the AST of the source code of the program. The debugger is used by the ITS through the API provided by the JDT Debug plug-in (https://projects.eclipse.org/projects/eclipse.jdt.debug) from Eclipse.

Figure 4 shows a complete example of the dynamic visualization generated using the proposed ITS through the AR device.

The movement of the vehicle along the visualization was done by constructing a directed cyclic graph. This graph was generated from the graphic representations added to the display, which defined the set of nodes that the vehicle must follow. Figure 5 shows an example of the generation of such a graph for the visualization generated in Figure 4.

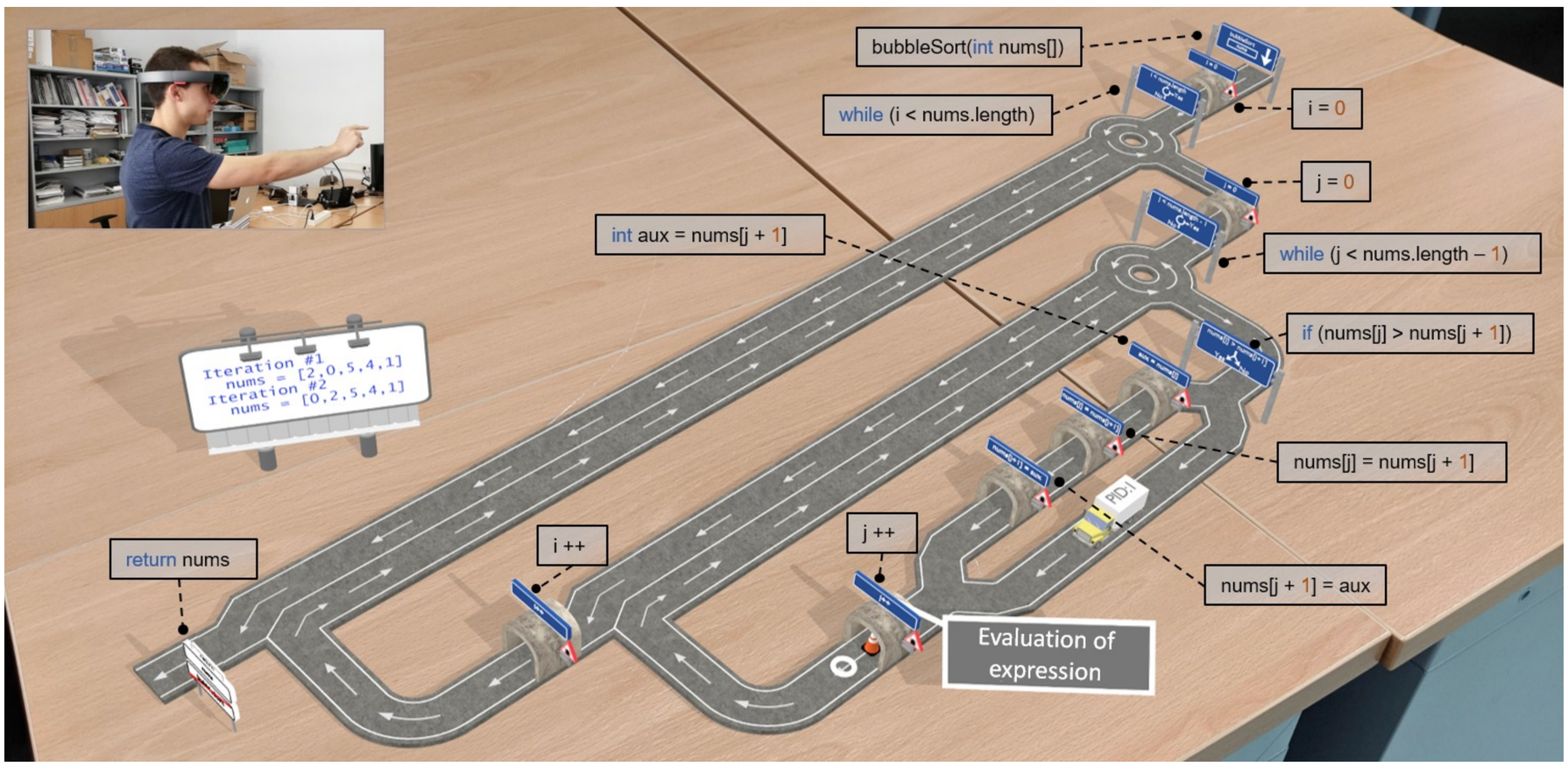
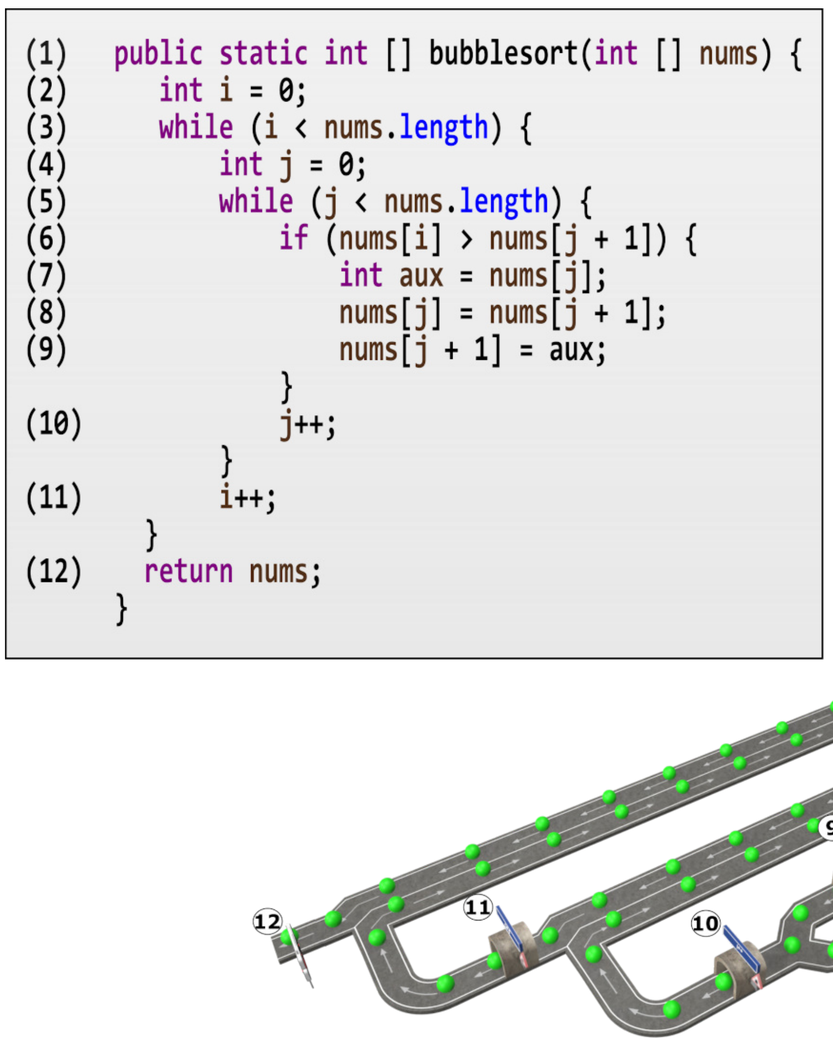


Figure 4. Dynamic visualization of the "Bubble sort" algorithm as seen through the AR device. At the bottom right, the execution thread of the program (vehicle) is shown moving towards the breakpoint (traffic cone) and an information panel when the cursor is superimposed on a graphic representation. At the same time, on the left side, a billboard is shown, which prints the status of the algorithm at a particular point of the execution.



```
(1)   public static int [] bubblesort(int [] nums) {
(2)      int i = 0;
(3)      while (i < nums.length) {
(4)          int j = 0;
(5)          while (j < nums.length) {
(6)              if (nums[i] > nums[j + 1]) {
(7)                  int aux = nums[j];
(8)                  nums[j] = nums[j + 1];
(9)                  nums[j + 1] = aux;
                 }
(10)             j++;
             }
(11)         i++;
         }
(12)     return nums;
      }
```


Figure 5. Example of the generation of a directed cyclic graph for the "Bubble sort" algorithm. The green spheres represent the sequence of nodes that the vehicle must follow.

The feedback provided to the user was used to guide and help them to understand the program that they were running. Thus, it was possible to define personalized feedback as comments in the source code, which were stored in JSON and shown to the user in the form of text bubbles that would appear when the vehicle passed through the graphic representation associated with the line of code where the comment was inserted. When defining these comments, certain control labels could be introduced to manipulate the execution of the program, as seen in Listing 1. In that case, the execution of the dynamic display would be stopped when the first comment was reached, due to the control label defined in the last line of the comment. This label system was flexible enough to define other

properties that allowed the teacher to direct the student's learning by using the comments included in the programming language itself.

Listing 1. Example of JAVA source code that includes a control label for the ITS.

```
// First, we add all the elements in the array using a reduce
// expression available in the Stream API introduced in Java 8:
//! its:execution = "pause"
List<Integer> integers = Arrays.asList(1, 2, 3, 4, 5);
Integer sum = integers.stream().reduce(0, (a, b) -> a + b);

// Second, ...
```

In addition to feedback based on comments, the ITS also incorporated a mechanism based on software metrics to provide additional information to the students. The selected metric was called "cyclomatic complexity", proposed by McCabe [43], and used to measure quantitatively the complexity of a program. To do this, we could calculate the cyclomatic complexity (CC) of a program with a single entry point and a single exit point in a simplified way according to [44], as $CC = NB + 1$, where NB is the number of branches in a program.

In this context, the system calculated this cyclomatic complexity automatically so that the method was represented, counting the total number of occurrences of the following keywords: **if**, **for**, **while**, **case**, **return**, **&&**, || and **?** (for the conditional ternary operator). Once this value was calculated, the result was shown on the display by changing the color to red for those graphic representations that increased the cyclomatic complexity of the method over the recommended value: 10. This value was selected from that proposed by McCabe in his work, identified as a recommended limit and interpreted as the program being "simple and easy to understand".

Figure 6 shows an example of a fragment of a visualization colored in red as the cyclomatic complexity increased over the recommended value, assuming that it belonged to a larger method. In this way, the students could reconsider their solution and modify it to reduce its complexity.

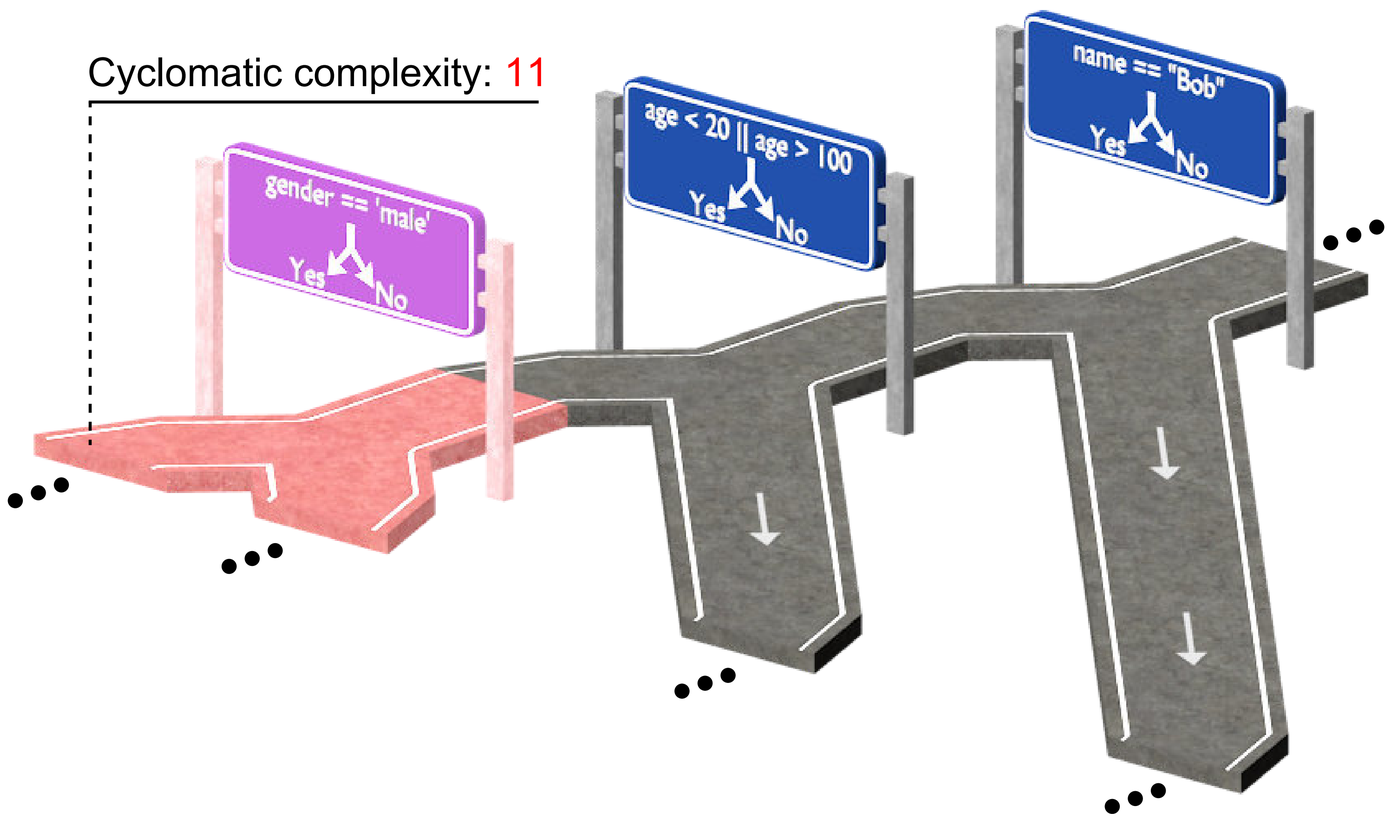


Figure 6. Cyclomatic complexity shown as a red shading, which highlights the graphic representation related to the statement that increases the complexity over the recommended limit.

4. Resulting Use Cases

The objective of the proposed ITS was to be used in learning to program environments to facilitate learning by providing aids and guides that allowed the student to understand different concepts such as program structure, control statements, their dynamic aspect by tracing the program execution, etc.

Moreover, the system could be extended to facilitate the learning of more specific programming concepts, such as those arising in concurrent programming contexts, and even in less advanced ones aimed at children who are trying to learn to program through video games. Thus, this section highlights a set of scenarios where the proposed system was used, showing its flexibility through the dynamic visualization of programs, child-oriented game environments, the use of the system as a use case in the classroom, and the understanding of concurrent programming concepts.

4.1. Dynamic Program Visualization

The students who were learning to program hardly used the debugging tools offered by current programming environments because their use still required a high level of abstraction, among other reasons. These tools helped to understand the execution of a program or to detect and correct errors at runtime.

In this context, the proposed ITS was used to build an interactive AR debugging environment where the teachers could teach programming concepts or demonstrate the use of a debugger in a real situation. Such an environment was built on a physical surface of the real world, from the 3D graphic representations presented in Figure 2.

Through a menu associated with the visualization, the user was in charge of starting the program debugging. Through this process, the Eclipse debugger was launched, which executed the actions performed in the visualization by sending orders through network sockets to the plug-in that acted as a server. Once the debugger was launched, the vehicle appeared, placing itself over the starting point of the visualization, which was initially the function definition statement.

The actions that the user could perform were the classic ones that a debugger integrates, which were executed through gesture interaction mechanisms supported by the visualization device. Among the most relevant actions, it is worth mentioning the possibility of checking the value of the variables when the user selected a sentence through which the vehicle passed, executing the program step-by-step through discrete vehicle movements or even adding breakpoints that enabled its movement without pausing until the statement that contained such an element. On the other hand, the user also had the possibility of checking the output of his/her program through a console in the shape of a billboard, which was placed next to the representation that printed the states through which it passed. Additionally, the comment system provided explanations to the user about what happened during the execution of the program, in order to facilitate the understanding of the source code.

This process ended when the vehicle reached the return sentence, disappearing from the display and stopping the process in both the device and the debugger in the Eclipse environment.

4.2. Agent-Based Game Environments

The application of the proposed ITS was interesting for programming agent-based models in game environments, in order to motivate 8 to 15-year-olds to take interest in computers and learn basic programming concepts. Students of these ages could be familiar with the concept by playing with toys reflecting actual objects, including cars, carpets simulating the streets of a city, and giant puzzles that allowed them to build structures based on roads.

The proposed system could be extended to support a didactic environment of programming based on agents, where the player communicates with the agent through a pre-established set of commands that allows them to manipulate the state of the agent.

In terms of learning, the levels of the game represented different challenges that the player must solve. These challenges were posed with a difficulty that introduced programming concepts to the player incrementally. Thus, the first levels of the game would introduce programming concepts related to the execution of expressions and conditions, to finally move on to the use of loops and functions.

The stages of the game presented a challenge to overcome by means of different actions that told the main character how to reach the goal of the level. Such stages were automatically generated over the real world by means of applying AR techniques. The character would execute the actions

sequentially, so that the player specified them as a list of commands. Thus, the player could replicate the movement of a horse in chess by specifying two orders to advance, one to turn right and another to advance forward. If the character reached the goal of the level after executing the sequence of actions, then the stage would be completed. Otherwise, the player would have to provide an alternative sequence. To check that the current action was being executed, a small character ran through them at the same time as the main character.

Figure 7 shows a snapshot of one of the levels included in the game.

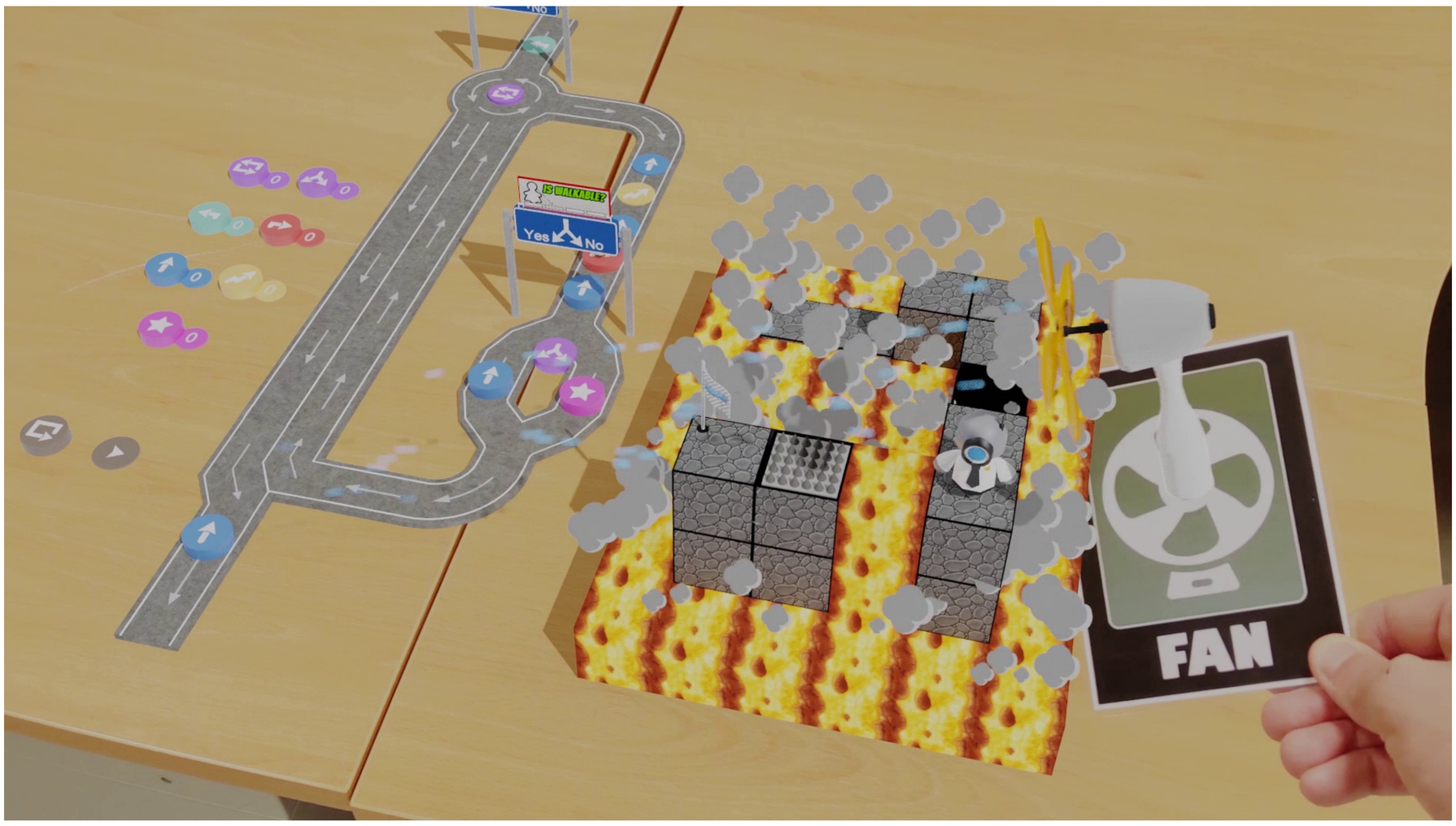


Figure 7. A snapshot of the RoboTIC game running the level used during the evaluation. A video playing this full level can be found at https://youtu.be/T1I2mrX7BUw.

4.3. Using the ITS in the Classroom

The use of ITS in the classroom was related to the use and integration in the COLLECE-2.0 environment to allow the teachers and the students to build programs collaboratively and to visualize them dynamically. This collaborative feature was achieved through COLLECE-2.0 sessions, whose actions were replicated on all visualization devices through network sockets. For this scenario, we considered that all the people involved in the learning process had this device to visualize and interact with the representation located on a real-world surface.

Mainly, the teacher was the one who decided the program to be displayed, positioning the representation on a suitable place in the classroom. In this way, the teacher shared the visualization so that the students could both interact with the representation and move freely through it.

Thus, in this use case, the ITS played a secondary role, and it was the teacher who guided the students during the visualization, stopping its execution to ask questions of the students about what was happening in the program. The students could also interact with the visualization in order to solve doubts that may arise during the lesson.

As an example and to place the description, we considered the "Bubble sort" algorithm as shown in Figure 4. In this visualization, a billboard that printed the program output was placed, a break point on an expression evaluation, and a vehicle that went through the graphic representations executing sentences.

In this context, the teacher could start an explanation by running the algorithm step-by-step during its first iterations, emphasizing the most relevant points of the execution.

Thus, the students could understand the behavior of the algorithm, using a lower level of abstraction and handling representations that were already known. Afterwards, several questions

could be asked to check the degree of understanding obtained after the initial iterations; for example, what the state of the list at a given time was, what the value of a variable in any given iteration was, or which branch of the inner conditional sentence the vehicle would pass through and why. Likewise, these questions could also be asked directly using the elements of the graphic representation.

Similarly, the students could also pose questions to the teacher about aspects that were not understood. In this case, the student could stop the execution and ask why a variable was updated to a certain value or add a ghost vehicle that served to reproduce the behavior that the main truck performed.

4.4. Comprehension of Concurrent Programming Concepts

A more advanced use case was the one oriented toward facilitating the learning of concurrent programming concepts. In this sense, the ANGELA notation integrated in the ITS was ideal for visualizing the execution flow of the programs, specifically through the vehicle that went through the graphic representations as a thread of execution.

Concurrent and real-time programming entailed some challenges in students who were used to working in programs with a single execution flow. This was due to the new concepts that had to be introduced to the student, such as processes, synchronization and communication mechanisms, etc. The proposed ITS could leverage the included notation to help visualize these concepts.

The identification of the critical sections in a program, i.e., memory variables that are shared between different processes, would be noteworthy among the concepts that the notation could represent. Such a concept could be represented by painting yellow diagonal stripes on the roads and placing vehicles that would represent the different running threads with their PID on their sides. In the case of the proposal, trucks were used to represent such threads.

In order to synchronize all the trucks (i.e., threads), we could use traffic lights to stop/start the movement of the trucks, that is to stop/start the running thread respectively to avoid them entering the critical section at the same time. To control the state of traffic lights, pressure plates were placed on the road to simulate the action of the wait/signal on the traffic light when the truck drove on it. Such behavior can be seen in Figure 8, where a thread with $PID = 3$ is running through the critical section. At the same time, another thread with $PID = 4$ is waiting at the traffic light before the critical section until the other thread drives on the pressure plate to signal the traffic light.

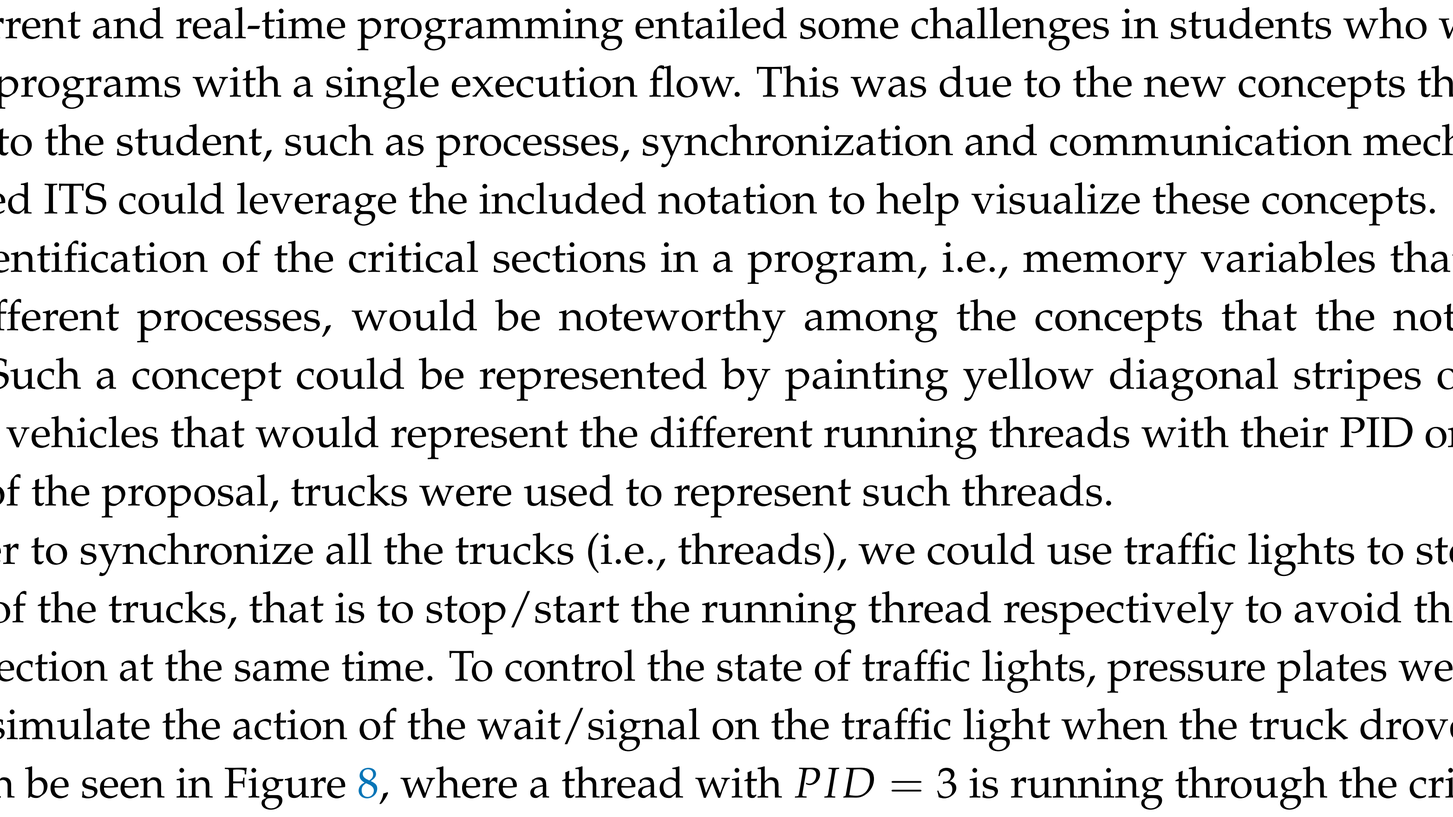


Figure 8. The ITS in action using the ANGELA notation showing a use case of concurrent programming. In this context, a thread (a) tries to access the critical section of a program (c), currently accessed by another thread and protected by a traffic light with a barrier (b) controlled by a pressure plate for signaling the semaphore (d). A video playing the full sequence can be found at https://youtu.be/66zaRSIjbPA.

5. Conclusions

In this work, we presented an ITS aimed to make learning to program easier through a notation based on the roads and traffic signs metaphor (ANGELA) through augmented reality. The system was presented in terms of the features and implementation in order to show the different characteristics of the proposal. Furthermore, through COLLECE-2.0, the system was integrated in a complete collaborative learning environment for programming.

The use cases raised exposed the usefulness, scalability, and flexibility of the system in multiple use cases, which contributed to validating the system in different areas such as interactive debuggers, its collaborative use in the classroom, the learning of concurrent programming concepts, and its integration in game environments oriented toward the learning of programming foundations in children.

As future lines of work, we highlight the need to work on two fundamental axes: (1) the in-depth study of the benefits that use cases bring to the learning of programming and (2) the analysis of their exploitation in AR environments on mobile devices with current technologies (e.g., ARCore, ARKit, Vuforia, etc.), which would allow the exploitation of the 3D space in an economic way in terms of the costs of the devices, thus giving the population with reduced resources access to the system.

Author Contributions: Conceptualization, S.S.-S. and C.G.-P.; funding acquisition, D.V., C.G.-M., and M.Á.R.; investigation, S.S.-S., C.G.-P., and D.V.; methodology, D.V. and C.G.-M.; project administration, M.Á.R.; software, S.S.-S. and C.G.-P.; supervision, D.V.; validation, C.G.-M. and M.Á.R.; visualization, C.G.-P.; writing—original draft, S.S.-S. and C.G.-P.; writing—review and editing, D.V. and C.G.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Economy, Industry and Competitiveness, and the European Grant Number "TIN2015-66731-C2-2-R".

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach*; Pearson Education Limited: Kuala Lumpur, Malaysia, 2016.
2. Mu, P. Research on artificial intelligence education and its value orientation. In Proceedings of the 1st International Education Technology and Research Conference (IETRC 2019), Tianjin, China, 14–15 September 2019; Francis Academic Press: London, UK, 2019; pp. 771–775.
3. Burning Glass Technologies. *Beyond Point and Click: The Expanding Demand for Coding Skills*; Burning Glass Technologies: Boston, MA, USA, 2016.
4. Europe Commission. Coding—The 21st Century Skill. 2018. Available online: https://ec.europa.eu/digital-single-market/en/coding-21st-century-skill (accessed on 24 December 2019).
5. Wing, J.M. Computational thinking and thinking about computing. *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2008**, *366*, 3717–3725. [CrossRef] [PubMed]
6. Llorens Largo, F.; García-Peñalvo, F.J.; Molero Prieto, X.; Vendrell Vidal, E. La enseñanza de la informática, la programación y el pensamiento computacional en los estudios preuniversitarios. *Educ. Knowl. Soc.* **2017**, *18*, 7–17. [CrossRef]
7. Bosse, Y.; Gerosa, M.A. Why is programming so difficult to learn? *ACM SIGSOFT Softw. Eng. Notes* **2017**, *41*, 1–6. [CrossRef]
8. Figueiredo, J.; García-Peñalvo, F.J. Building Skills in Introductory Programming. In Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality, Salamanca, Spain, 24–26 October 2018; ACM: New York, NY, USA, 2018, pp. 46–50.
9. Koti, M.S.; Kumta, S.D. Role of Intelligent Tutoring System In Enhancing The Quality of Education. *Int. J. Adv. Stud. Sci. Rese.* **2018**, *3*, 330–334.
10. Kulik, J.A.; Fletcher, J. Effectiveness of intelligent tutoring systems: A meta-analytic review. *Rev. Educ. Res.* **2016**, *86*, 42–78. [CrossRef]

11. Hidalgo-Céspedes, J.; Marín-Raventós, G.; Lara-Villagrán, V. Learning principles in program visualizations: A systematic literature review. In Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE), Erie, PA, USA, 12–15 October 2016; pp. 1–9.
12. Velázquez-Iturbide, J.Á.; Hernán-Losada, I.; Paredes-Velasco, M. Evaluating the effect of program visualization on student motivation. *IEEE Trans. Educ.* **2017**, *60*, 238–245. [CrossRef]
13. Urquiza-Fuentes, J.; Velázquez-Iturbide, J.Á. Toward the effective use of educational program animations: The roles of student's engagement and topic complexity. *Comput. Educ.* **2013**, *67*, 178–192. [CrossRef]
14. Urquiza-Fuentes, J.; Vel'zquez-Iturbide, J.Á. A long-term evaluation of educational animations of functional programs. In Proceedings of the 2012 IEEE 12th International Conference on Advanced Learning Technologies, Rome, Italy, 4–6 July 2012; pp. 26–30.
15. Burgess, N.; Maguire, E.A.; O'Keefe, J. The human hippocampus and spatial and episodic memory. *Neuron* **2002**, *35*, 625–641. [CrossRef]
16. Teyseyre, A.R.; Campo, M.R. An overview of 3D software visualization. *IEEE Trans. Vis. Comput. Graph.* **2008**, *15*, 87–105. [CrossRef]
17. Cabero Almenara, J.; Barroso, J. *The Educational Possibilities of Augmented Reality*; University of Alicante: Alicante, Spain, 2016.
18. Psotka, J.; Massey, L.D.; Mutter, S.A. *Intelligent Tutoring Systems: Lessons Learned*; Psychology Press: Hove, UK, 1988.
19. Holland, J.; Mitrovic, A.; Martin, B. *J-LATTE: A Constraint-Based Tutor for Java*; University of Canterbury: Christchurch, New Zealand, 2009.
20. Hong, J. Guided programming and automated error analysis in an intelligent Prolog tutor. *Int. J. Hum. Comput. Stud.* **2004**, *61*, 505–534. [CrossRef]
21. Weragama, D.; Reye, J. The PHP intelligent tutoring system. In *Proceedings of the International Conference on Artificial Intelligence in Education*; Springer: Berlin, Germany, 2013; pp. 583–586.
22. Hooshyar, D.; Ahmad, R.B.; Yousefi, M.; Fathi, M.; Horng, S.J.; Lim, H. SITS: A solution-based intelligent tutoring system for students' acquisition of problem-solving skills in computer programming. *Innov. Educ. Teach. Int.* **2018**, *55*, 325–335. [CrossRef]
23. Steenbergen-Hu, S.; Cooper, H. A meta-analysis of the effectiveness of intelligent tutoring systems on college students' academic learning. *J. Educ. Psychol.* **2014**, *106*, 331. [CrossRef]
24. Robison, J.; McQuiggan, S.; Lester, J. Evaluating the consequences of affective feedback in intelligent tutoring systems. In Proceedings of the 2009 3rd International Conference on Affective Computing and Intelligent Interaction and Workshops, Amsterdam, The Netherlands, 10–12 September 2009; pp. 1–6.
25. Kehrer, P.; Kelly, K.; Heffernan, N. Does immediate feedback while doing homework improve learning? In Proceedings of the Twenty-Sixth International FLAIRS Conference, St. Pete Beach, FL, USA, 22–24 May 2013.
26. Waller, J.; Wulf, C.; Fittkau, F.; Döhring, P.; Hasselbring, W. Synchrovis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency. In Proceedings of the 2013 First IEEE Working Conference on Software Visualization (VISSOFT), Eindhoven, The Netherlands, 27–28 September 2013; pp. 1–4.
27. Jimenez-Diaz, G.; Gonzalez-Calero, P.A.; Gomez-Albarran, M. Role-play virtual worlds for teaching object-oriented design: The ViRPlay development experience. *Softw. Pract. Exp.* **2012**, *42*, 235–253. [CrossRef]
28. Mathur, A.S.; Ozkan, B.K.; Majumdar, R. iDeA: An immersive debugger for actors. In Proceedings of the 17th ACM SIGPLAN International Workshop on Erlang, St. Louis, MO, USA, 23–29 September 2018; ACM: New York, NY, USA, 2018; pp. 1–12.
29. Lazaridis, V.; Samaras, N.; Sifaleras, A. An empirical study on factors influencing the effectiveness of algorithm visualization. *Comput. Appl. Eng. Educ.* **2013**, *21*, 410–420. [CrossRef]
30. Velazquez-Iturbide, J.A.; Debdi, O.; Esteban-Sanchez, N.; Pizarro, C. GreedEx: A visualization tool for experimentation and discovery learning of greedy algorithms. *IEEE Trans. Learn. Technol.* **2013**, *6*, 130–143. [CrossRef]
31. Hundhausen, C.D.; Douglas, S.A.; Stasko, J.T. A meta-study of algorithm visualization effectiveness. *J. Vis. Lang. Comput.* **2002**, *13*, 259–290. [CrossRef]

32. Naps, T.L.; Rößling, G.; Almstrum, V.; Dann, W.; Fleischer, R.; Hundhausen, C.; Korhonen, A.; Malmi, L.; McNally, M.; Rodger, S.; et al. Exploring the role of visualization and engagement in computer science education. In Proceedings of the ACM Sigcse Bulletin, Cincinnati, KY, USA, 27 February–3 March 2002; ACM: New York, NY, USA, 2002; Volume 35, pp. 131–152.
33. Lacave, C.; Molina, A.I.; Giralt, J. Identificando algunas causas del fracaso en el aprendizaje de la recursividad: Análisis experimental en las asignaturas de programación. In *Jornadas de Enseñanza Universitaria de la Informática (19es: 2013: Castelló de la Plana)*; Universitat Jaume I. Escola Superior de Tecnologia i Ciències Experimentals: Castelló de la Plana, Spain, 2013.
34. Sánchez, S.; García, M.; Lacave, C.; Molina, A.I.; González, C.; Vallejo, D.; Redondo, M. Applying mixed reality techniques for the visualization of programs and algorithms in a programming learning environment. In Proceedings of the 10th International Conference on Mobile, Hybrid, and On-line Learning, Rome, Italy, 25–29 March 2018; pp. 84–89.
35. Bacca, J.; Baldiris, S.; Fabregat, R.; Graf, S.; others. *Augmented Reality Trends in Education: A Systematic Review of Research and Applications*; International Forum of Educational Technology & Society: Denton, TX, USA, 2014.
36. Teng, C.H.; Chen, J.Y. An augmented reality environment for learning opengl programming. In Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, Fukuoka, Japan, 4–7 September 2012; pp. 996–1001.
37. Teng, C.H.; Chen, J.Y.; Chen, Z.H. Impact of augmented reality on programming language learning: Efficiency and perception. *J. Educ. Comput. Res.* **2018**, *56*, 254–271. [CrossRef]
38. Vujošević-Janičić, M.; Tošić, D. The role of programming paradigms in the first programming courses. *Teach. Math.* **2008**, *11*, 63–83.
39. UNECE. Convention on Road Signs and Signals. *U. N. Treaty Ser.* **1968**, *1091*, 3.
40. Bray, T. The JavaScript Object Notation (JSON) Data Interchange Format. 2017. Available online: https://tools.ietf.org/html/rfc8259 (accessed 2 February 2020).
41. Sánchez, S.; Redondo, M.A.; Vallejo, D.; González, C.; Bravo, C. COLLECE 2.0: A distributed real-time collaborative programming environment for the Eclipse platform. In Proceedings of the 11th International Conference on Interfaces and Human Computer Interaction (IADIS), Lisbon, Portugal, 21–23 July 2017; pp. 136–142.
42. Lacave, C.; García, M.Á.; Molina, A.I.; Sánchez, S.; Redondo, M.Á.; Ortega, M. COLLECE-2.0: Un sistema de programación colaborativa en tiempo real sobre Eclipse. In Proceedings of the 21st International Symposium on Computers in Education, Jerez de la Frontera, Spain, 19–21 September 2019.
43. McCabe, T.J. A complexity measure. *IEEE Trans. Softw. Eng.* **1976**, *SE-2*, 308–320. doi:10.1109/TSE.1976.233837. [CrossRef]
44. Hansen, W.J. Measurement of program complexity by the pair: (Cyclomatic Number, Operator Count). *SIGPLAN Not.* **1978**, *13*, 29–33. doi:10.1145/954373.954375. [CrossRef]

2.1.2 RoboTIC: A serious game based on augmented reality for learning programming

- Title: RoboTIC: A serious game based on augmented reality for learning programming [51]
- Authors: Santiago Sánchez, David Vallejo, Carlos González, Miguel Ángel Redondo, and José Jesús Castro
- Type: Journal
- Journal: Multimedia Tools and Applications
- Publisher: Springer Nature
- E-ISSN: 1573-7721
- Year: 2020
- DOI: 10.1007/s11042-020-09202-z
- Category: Computer Science, Software Engineering
- Impact Factor (2019): 2.313
- JCR Ranking: Q2
- Related to the current research topic: Yes

Multimedia Tools and Applications
https://doi.org/10.1007/s11042-020-09202-z

RoboTIC: A serious game based on augmented reality for learning programming

Santiago Schez-Sobrino[1] · David Vallejo[1] · Carlos Glez-Morcillo[1] · Miguel Á. Redondo[1] · José Jesús Castro-Schez[1]

Received: 3 September 2019 / Revised: 26 May 2020 / Accepted: 8 June 2020 /
Published online: 02 July 2020

Abstract

Coding skills are becoming more and more important in today's world, especially within the context of the fourth industrial revolution. They also help practice other 21 century skills such as computational thinking, problem solving and teamwork. Unfortunately, learning how to program is tough and can be also frustrating for beginner students. In this work we introduce RoboTIC, a serious game based on gamification and Augmented Reality that facilitates the learning of programming to students in lower levels of the education system by using a novel set of visual metaphors derived from a notation of roads and traffic signs. The architecture that supports RoboTIC has been designed to allow the integration of multimedia components when new programming concepts and techniques must be addressed and to add game levels that enable students to learn incrementally. Experiments have been conducted in a youth center with children who do not have coding skills at all to demonstrate the feasibility of the proposal. The results show promising conclusions in terms of children's motivation and interest in programming.

Keywords Learning programming · Serious games · Visual programming · Augmented reality

✉ Santiago Schez-Sobrino
santiago.sanchez@uclm.es

David Vallejo
david.vallejo@uclm.es

Carlos Glez-Morcillo
carlos.gonzalez@uclm.es

Miguel Á. Redondo
miguel.redondo@uclm.es

José Jesús Castro-Schez
josejesus.castro@uclm.es

[1] University of Castilla-La Mancha, Paseo de la Universidad 4, Ciudad Real, 13071, Spain

1 Introduction

The learning of programming is one of the most important competences that can be acquired today. This is also considered to be critical in the area of the so-called fourth industrial revolution [39], in which the impact of the technologies such as artificial intelligence or robotics are changing the way we live, our relationships, and how we work together. Such is the case, that there are studies that predict that a significant part of the work currently being undertaken in this field will be totally automated [7], drastically modifying the economic model and generating, inherently, challenges and new business possibilities [2].

At the educational level, there are already ambitious programs based on including skills related to learning programming in the public curricula or, at least, after school programs. Countries such as Finland or South Korea, which have a very good reputation in terms of basic education and achieve excellent scores on the PISA tests, are representative examples in this respect. On a more general level, there are also initiatives such as the *Hour of Code* [29], aimed at introducing programming to people between the ages of 4 and 10, typically through the construction of interactive applications such as video games and robot programming, among others.

In stages prior to professional development, during primary and secondary education (K-12), it has been demonstrated that programming provides certain benefits over problem solving through so-called computational thinking, which implies the use of programming concepts through the analysis and recognition of repetitive patterns, the design of algorithms that allow solving problems step by step, the use of abstractions that simplify those problems and the decomposition of those problems into other simpler problems easier to be solved [50].

Unfortunately, programming is a complex task and therefore the teaching/learning process represents a significant challenge in the context of increasing students' chances of success in the new digital world. Thus, it is essential to motivate and stimulate the student by a qualified teacher who has the necessary mechanisms to adequately guide this process through approaches that go beyond the traditional ones that materialize in face-to-face classes. In fact, these approaches tend not to take full advantage of the cognitive abilities of the individual, which have been shown to be optimized to process information in a multimodal way through the combination of experiences involving the use of voice, gestures, sounds, and images, among others [30]. In this context, the use of active learning environments as traversal platform, based on a learn anywhere, anytime approach can contribute to increase the flexibility of the students when practicing and improving their skills as a programmer.

One of these techniques is to use gamification tools that include elements of video games in less related contexts [20]. Thus, these tools can be used to enhance the learning experience of programming by increasing the motivation of students and their participation in activities. This can be done in a general way, applying analogies to the evaluation process by replacing traditional terms with more gamified ones (e.g. current course by game, subject credits by points, learning units by levels, etc.) [13, 26], and more specifically, by creating video games that integrate programming concepts into their gameplay [17, 40].

In the field of video games, these concepts can be encompassed in the so-called serious games [8]. Thus, by means of specific games, learning mechanisms integrated in the gameplay can be included where the player has to solve programming challenges in order to advance in the game (e.g. [51]), and others where the player interacts with the game world by means of commands that represent sentences in a programming language. An example

of the latter would be LOGO [1] and its derivatives (e.g. [34]), which emerge as tools for teaching logic and programming concepts to the player.

Thus, both gamification and serious games allow to reduce the level of abstraction at the time of understanding programming concepts, through its relationship with elements in the context of video games. This relationship aims to facilitate the understanding of concepts explained through other familiar to students through the use of metaphors. In other words, using the concept of metaphor to make use of techniques that reduce the level of abstraction required during learning by creating visual metaphors that serve to understand specific concepts of programming [19]. These visual metaphors can be used in software visualization environments to represent the structure of programs, visualize the execution of an algorithm, or visually program applications [9].

The visual metaphors can be transferred to a three-dimensional space to benefit from different advantages, such as being able to show a greater amount of information due to the new dimension [38], use a realistic learning environment by using representations close to the real world [44] and cause a more effective use of the student's spatial memory in order to recognize and remember certain behaviors found in programs through the human subconscious [3, 28].

This three-dimensional space can be deployed in an Augmented Reality (AR) environment that allows interaction with visual elements in a more natural way, resulting in a more enriching experience for the student, given the potential benefits of using this approach in an educational context [4, 10]. Such AR environments can be leveraged to improve the learning gains in contrast to more traditional learning models [21], reduce the level of abstraction required by a virtual environment by including real-world elements [25] and improve the collaborative problem solving [10, 27], among others.

In this context, the present work introduces RoboTIC, a serious game designed to facilitate the learning of programming concepts in students of elementary educational levels (from 8 to 15 years), in order to develop their computational thinking through the ideas of gamification, serious games, visual metaphors, and 3-D representation and interaction. The main objective of the proposal is to motivate the learning and self-learning, to improve the understanding of fundamental programming concepts and to facilitate the comprehension of problem solving techniques. RoboTIC incorporates elements of the real world through a set of metaphors based on roads and traffic signs visualized in a three-dimensional environment of AR and grouped in different levels that the student must solve. The proposed architecture has been designed to add, in a scalable way, multimedia resources that facilitate the integration of new programming elements and new levels so that the students themselves can approach them in the context of an incremental complexity.

RoboTIC has been evaluated through an analysis of the serious game with a group of children, using an experiment that consisted of several phases and made it possible to evaluate in terms of learning basic programming concepts such as instructions, conditional sentences and loops.

The main contributions of this article are as follows:

- Firstly, we introduce RoboTIC, a serious game for learning programming concepts through a block based programming environment, supported by a scalable architecture that allows to integrate new levels in an incremental way.
- Secondly, an evaluation of RoboTIC is conducted by children that are faced with the challenge of completing a level where basic programming concepts are addressed.
- Thirdly, RoboTIC has been developed considering leveraging 3-D graphics and using AR gameplay mechanisms actively in the game levels.

– Fourthly, natural interaction mechanisms are introduced to play RoboTIC, as well as other physical elements to affect the virtual world.

The rest of the paper is organized as follows. In Section 2, some similar tools and environments based on learning programming concepts are described, as well as introducing the approach of this work. Section 3 focuses on the tool proposed, its architecture and its main features. In Section 4, the research questions raised and an evaluation of the tool with actual children are presented. Finally, Section 5 draws some final conclusions, discusses the experimental results and suggests possible lines of future work.

2 Related work

Within the scope of serious games intended to train in programming competitions there is a wide variety of alternatives proposed by several authors. The proposal for this work is based on LOGO [1], although the existing literature also includes other related works. This is the case of PlayLOGO 3D [36], an implementation of LOGO on a 3D environment that allows turn-based battles between players using movement commands to try to collide with the other player's character. Other examples would be Cube Game [37] and Lightbot [18], where you have to solve block labyrinths by guiding the main character to a specific position using movement commands. The main contribution of these works lies in introducing the operation of a computer through predefined commands, making players see that a computer can only understand a limited set of instructions. The reader is invited to go to [46], which includes a systematic review of these and other similar serious games in the context of learning programming.

Other more advanced alternatives venture into the field of visual programming [6] using tools such as Scratch [31] or Blockly [15], which expand the possibilities of the previous tools by allowing to define the logic of more complex programs. These tools would be oriented to introduce concepts of programming languages, such as the use of variables, functions, loops, conditional sentences, mathematical operations, etc.

In line with the above but in relation to the metaphors used during the learning of programming, in [47] it is proposed a visual programming system that uses real-world metaphors to build programs, such as telephones to call functions, boxes for variables, and arrows next to people to represent entry and exit commands, among others. Also, one would find the set of metaphors presented in [41] to visually represent the execution of programs. These metaphors represent the possible roles that a variable can take, for example, if it is a variable that will increase in a loop, a series of footprints will be used in the ground with the past and future values that the variable will take, while if it is an array, it will be made by means of a set of tombs with the values of the array sculpted on them.

In the field of 3-D graphics there are also other works that aim to facilitate the learning of programming through different metaphors. This is the case of [32], where the authors used three-dimensional animations involving pipes and cubes to represent data assignments between variables, as well as other concepts such as planes to represent scopes of functions, among others. In a more abstract way, in [22], avatars and 3-D animations are used to represent programming concepts, such as message passing 3 between objects (avatars). Also, in a more advanced way using Virtual Reality techniques, in [33] it is proposed a system that facilitates the understanding of concurrent programming concepts through actors and 3-D boxes.

Some other works intend to improve the computational thinking and the process of learning to program by including different contents based on AR. In [35] a tool combining immersive environments and 2-D interaction is proposed to foster interest in computational thinking through dancing. In [16] an AR book is proposed to explain electronic circuits and programming using Scratch for Arduino. In [11] was presented a programming toy that consisted of physical markers used to build a virtual map on which players could program a character's movement to reach a target through an Android application. In the same line, in [23], a similar tool is proposed but the character's movement is performed through setting physical cards on the board. Finally, in [45] a setup consisting of a computer, a camera and physical cards with QR codes was used to facilitate the learning of 3-D programming concepts like translating an object through the 3-D space, change the lights in the scene, etc.

The tool proposed in this work aims to improve the current situation by using AR as a necessary gameplay mechanism to complete the levels of the game and keep the players motivated. In the list of previous works, AR was used by means of physical markers to slightly affect some gameplay mechanics (e.g. visualizing information). In the case of the proposal of this work, AR is intended to be used (i) passively so that the player uses the perspective of the level to discover the best strategy to solve it and (ii) actively to increase the motivation of the player to understand programming concepts such as conditional sentences through physical cards. These AR mechanisms are intended to be leveraged using more natural interaction ways that reduce the existing limitations of physical markers. Moreover, the visual metaphor based on roads and traffic signs that is included in RoboTIC is actively used in the game to reinforce the explained programming concepts by representing the execution flow, loops and conditional sentences. This visual metaphor, called ANGELA, has been proposed in previous works [42, 43] where it was successfully evaluated as a notation to facilitate the learning of programming.

3 RoboTIC approach to learn computational thinking competence

3.1 General overview

RoboTIC is an educational serious game oriented to the learning of elementary programming concepts such as, for example, execution flows, conditionals and loops, among others, and thus improving the overall computational thinking. The application is mainly aimed at children between the ages of 8 and 15 who want to start programming through video-games.

It includes different multimedia game elements (graphic content, audio and multimodal interaction) that facilitate the comprehension of different programming concepts. On the one hand, there is the execution flow of the program, which is defined by a sequence of graphic elements based on the metaphor of roads and traffic signs, derived from the ANGELA notation. This notation allows to compose the structure of a program and the associated logic through the use of roads and traffic signs, respectively. In the case of RoboTIC, an abstraction of the metaphor has been used to simplify and facilitate the understanding of the game to the target audience. Thus, the loops are represented by a roundabout and a traffic sign indicating the number of times the body of the loop will be repeated. In the case of conditional sentences, the use of a bifurcation is proposed where the possible conditions are predefined beforehand by means of a series of conditional cards that represent the possible events that may lead to one or the other path of the bifurcation.

The video-game represents the virtual world applying AR technologies that allow the player to interact with that world through the use of natural user interfaces (NUI). These

means of interaction allow to influence the virtual world through the use of gestural and voice interfaces, or through the alteration of elements of the physical world. In such way, AR in RoboTIC is used to keep players motivated towards the levels they have to solve. In the same way, it is used as a game mechanic through the use of AR stickers and the perspective itself, forcing the player to explore the level from different angles to get a correct solution of the level.

Each level in RoboTIC presents a challenge that the player must solve by instructing the main character (robot) how to proceed to reach the goal of the stage. This stage is built of blocks that share the same size, so that the movement of the robot through the level is done discreetly. The actions that the robot can perform for the level are indicated by the player in the form of commands that the robot will perform sequentially. Thus, the player could communicate to the robot two commands to advance, one to turn to the right and another one to advance, to achieve a movement similar to the one made by a horse in chess. In case the goal is in the final position reached by the robot, the player would complete the level; otherwise, the sequence of orders would have to be corrected with a new alternative. These commands are executed by a small replica of the robot (mini-robot), which moves over the sequence of instructions at the same time as the robot executes them.

The video-game is compiled for the Microsoft HoloLens AR device and is available for download at: www.esi.uclm.es/www/santiago.sanchez/robotic/RoboTic_1.2.4.2.appxbundle. Fig. 1 shows a snapshot of RoboTIC running one of the available levels.

To support the functionality provided by RoboTIC, the system architecture is defined by different layers that communicate with each other, maintaining a division of responsibilities based on what each layer brings to the user. This division of responsibilities facilitates its expansion and improvement depending on the domain to be affected:

– **Perception layer**: is in charge of obtaining information from the user and the surrounding environment for further processing. Specifically, the AR device is responsible for analyzing the real world, the movement of the user, the actions it performs, etc. to allow interaction with the system. This layer provides the user with the necessary interaction mechanisms to influence the game world.

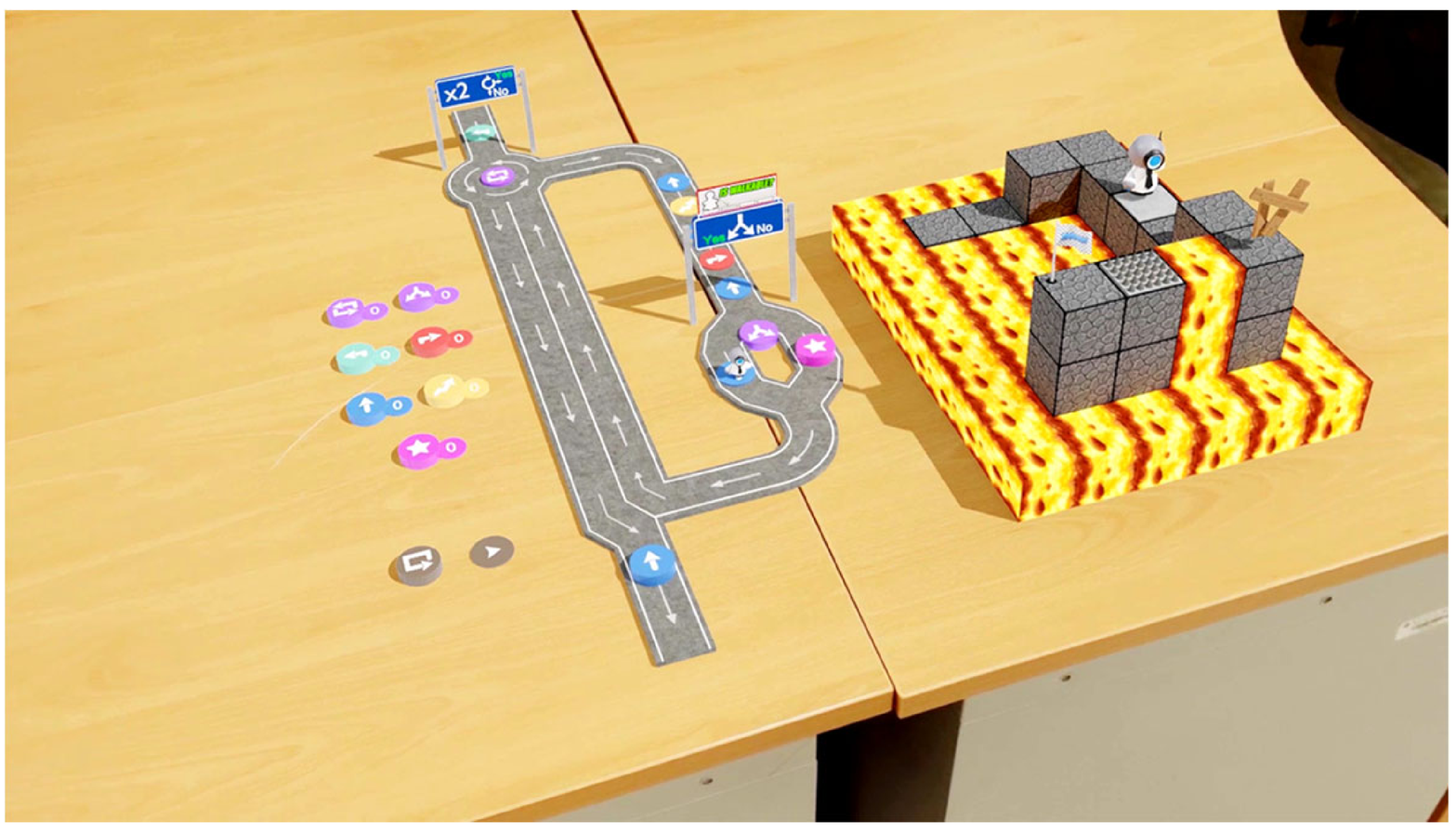


Fig. 1 A snapshot of the RoboTIC game running the level used during the evaluation. A video playing this full level can be found at https://youtu.be/T1I2mrX7BUw

- **Cognitive layer**: this layer processes the behavior defined by the user and assigned to the robot of the virtual world with which it interacts, as well as the influence of other elements of the physical world on the virtual one and the load of levels. This layer provides the user with the challenges that must be solved to overcome the levels of play, as well as the validation of the rules defined.
- **Representation layer**: this layer is dedicated to the construction of the virtual world from metaphors of the real world that facilitate the assimilation of programming concepts. This layer provides the user with the visual feedback and awareness mechanisms necessary to maintain their motivation and interest during the game experience.

Fig. 2 shows a diagram of those layers with the main components that define them, as well as the elements with which the user interacts in the game.

3.2 Perception layer

The domain of the perception layer supposes the users' entry point to RoboTIC, so it is necessary that it offers simple interaction mechanisms that the user knows how to take advantage of to maintain their interest at all times. For this reason, and given the capabilities it offers over other alternatives, the Microsoft HoloLens AR device has been selected as the execution medium for the video-game and the Unity video-game engine for its development.

The device makes it possible to analyze the environment and detect real-world elements such as the floor, ceiling, walls and tables, among others. In this way, the perception layer is able to identify any horizontal surface and use it to load the level of the game on it. These horizontal surfaces are typically the floor or the tables, although the seats of the chairs are also identified as such. The perception layer will only load the game levels onto the real-world elements identified as tables or on the floor.

The game levels are adapted to the size of the surface, never exceeding one square meter in size so that the level can be viewed completely through the field of view of the AR device at an appropriate distance. These surfaces are divided in half into two distinct parts, being the first half (i) the part of the game area that contains the sequence of instructions along with the mini-robot and the second half (ii) the part that shows the level of the game to be solved and the robot.

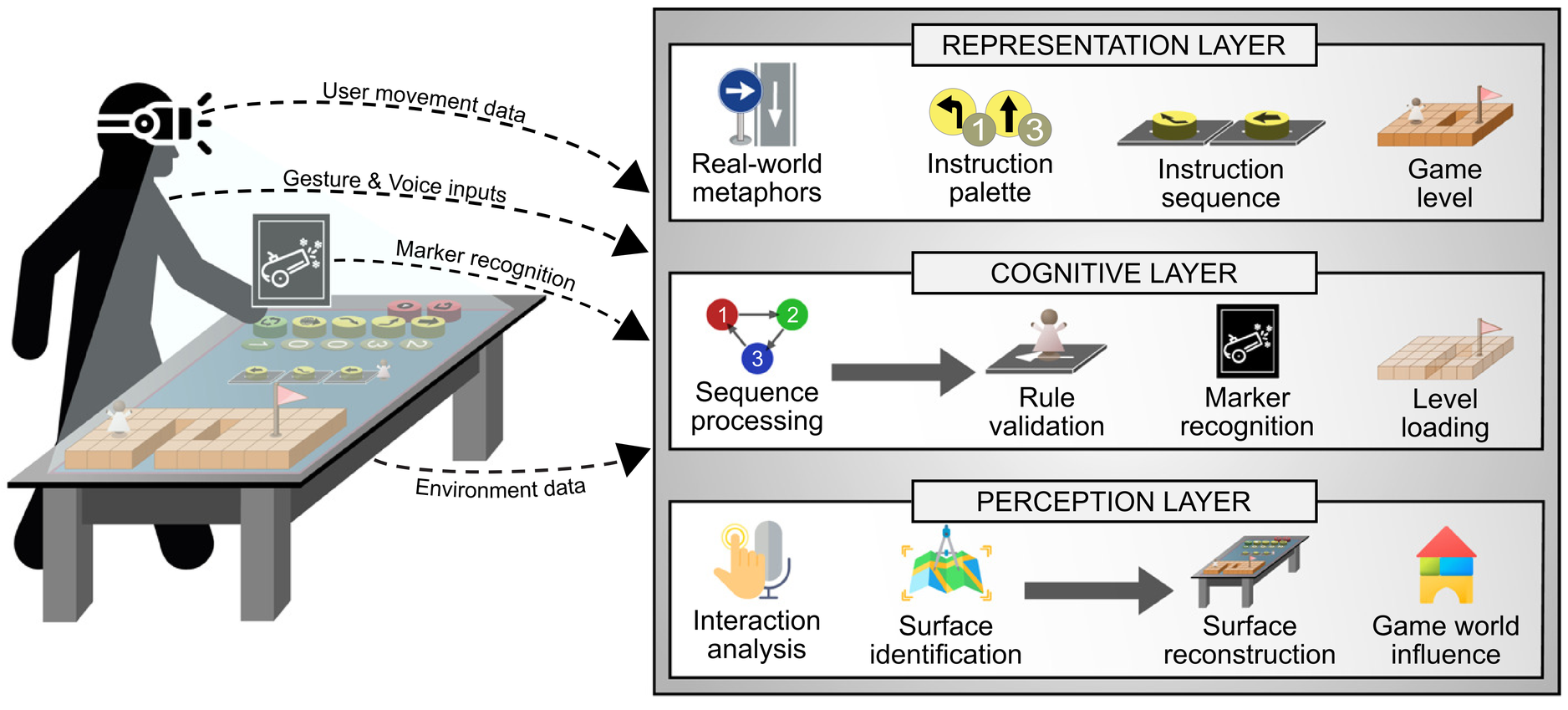


Fig. 2 General overview of the layers and elements that comprise the system

At the same time, this layer also provides the player with different interaction mechanisms to influence the game. The main method would be based on gestures recognized by the AR device and that allow the user to select the instructions that the robot will execute on the level, besides allowing the interaction with the different elements of the interface. Another method of interaction is provided implicitly by the device and the sensors it includes for positioning and orientation, allowing the user to move through the physical world to observe the level of the game from all its angles in order to identify the appropriate sequence of instructions that solves the level. Finally, interaction mechanisms are also integrated through recognition of physical marks that allow special actions to be performed on the level, such as freezing a lava path or destroying an obstacle for the robot to advance.

The perception layer shares with the cognitive layer the information related to the surface where the game level is executed, as well as the information and actions triggered by the interaction that the player makes with it.

3.3 Cognitive layer

The cognitive layer is where the processing of the information obtained from the perception layer is performed, in such a way that the robot executes the sequence of instructions defined by the user.

The instructions that can be executed by the robot are divided into control instructions, condition instructions and execution instructions. The former and the second correspond to the definition of the logical structure of execution by means of loops, conditional sentences and the conditions themselves, while the latter include those instructions that indicate to the robot the actions to be performed. The simile used is that the execution instructions tell the robot what to do, while the control and condition instructions define how to do it.

This layer is also in charge of the logic necessary to load the game levels. The game initially has ten levels, which present a progressive difficulty to the player. During the first levels the player is introduced with the execution instructions, and then give way to the introduction of the control instructions and conditions. Finally, the player must use the physical marks to overcome the last levels.

The levels are stored as files in JSON format that contain all the necessary information to be loaded during execution and replicate the full level. This information includes: (i) the size of the level indicated in maximum number of blocks along each axis of coordinates, (ii) the location where the robot will appear, (iii) the location where the target or goal to be reached by the player will be found, (iv) the number of instructions of each type that can be used in the level, (v) the items available on the level and (vi) the location and type of each of the blocks that compose the level represented by numerical identifiers. An example of a level and its internal representation in the specified data file is shown in Fig. 3. Specifically, the key `map` of the level data file defines the blocks that form the level through a two-dimensional list that indicates the type of block that exists in each position, according to the Equation 1

$$B_{(X,Y,Z)} = B_{i,j} \mid i = X + w \times Y,\ j = Z \tag{1}$$

where B is the type of block that occupies the position (X, Y, Z) of the level, i is the index of the list corresponding to the coordinates (X, Y), j is the index of the list that indicates the height of the level on the Z axis and w is the width of the level along the X axis. Thus, the block $B_{(1,2,1)} = B_{7,1}$ would correspond to block type 0, i.e. the stone block on which the level goal is located in the figure.

Fig. 3 A simple 3x3 level with two layers (left) along with its defining JSON data (right)

This loading of levels facilitates the addition of new levels, requested at the beginning to a web service that is in charge of serving the available list of levels. Thus, users can load their own levels in this web service so that they are available to the rest of the players, in such a way that a selection with the desired levels can be configured from the game.

3.4 Representation layer

The representation layer is oriented to the rendering of the different virtual elements that shape the game interface and the levels, both the part where the sequence of instructions is defined and the part where the level is built.

On the one hand, the game interface is defined by various menus and feedback messages, which provide the user with information about each state of the game.

During the resolution of the levels, the player is introduced mainly with the graphic elements that conform the level, the robot and the mini-robot. In the part of the level where the sequence of instructions is defined, the mini-robot would be located on a fragment of road. This road will increase in size as new instructions are added to the sequence and finally the mini-robot will move through it, executing the instructions that it finds in its path for the robot to move.

To the left of the sequence of instructions is the palette of instructions, each of which is represented by means of circular buttons. Here, the player can select those actions that he/she wants to add to the sequence, and once he/she has finished, execute the sequence to try to make the robot reach the target. The user has total freedom to remove from the sequence those instructions that he/she does not consider useful or replace them with others, as long as he/she respects the limitation on the number of instructions of each type available for the level. In special cases where a control instruction is added from the palette, the player's focus will shift to the sequence of instructions to complete the parts required by that control instruction.

The control instructions regarding conditional sentences are represented by a bifurcation on the road, assuming the left lane as the sequence to be executed if the condition is met, while the right lane would contain those instructions that will only be executed if the condition is not met. The condition to be evaluated in this type of control instructions can be selected by the player from a list of cards with predefined conditions.

In the case of control instructions concerning loops, their visual representation would be made by means of several sections of road arranged in the form of a roundabout, thus

reflecting the concept of repetition. These instructions are limited only to the repetition of a number of times of the body of the loop (right part). The number of repetitions can be increased by interacting directly on the signal.

4 Experimental results

In order to evaluate the proposal of this work, an experiment has been conducted with children where they tested the game in a real scenario of AR with the Microsoft HoloLens visualization device and in the context of a level that addressed programming concepts related to instructions, conditional sentences and loops. This section describes the research questions raised, the performance of the experiment, the usage scenario, the participants, the results obtained and the possible limitations faced. Discussion of the results and their relation to the research questions formulated are left to the reader in the Section 5.

4.1 Participants

This experiment involved 12 children (9 boys and 3 girls) from the youth center of Torralba in Calatrava (Ciudad Real, Spain). The average age of the participants was 12 years old. They were recruited from a campaign based on the city council's social networks and promotional posters on the walls of the youth center. They did not receive any incentive to participate in the experiment. None of them knew the game or had tried it before. None of the organizers of the experiment knew the participants beforehand.

Due to the above, the authors consider that the results obtained from the experiment are not biased by the profile of the participants or the way in which they were recruited.

4.2 Method

To validate the usefulness of the proposal, the following research question arises, which we can be divided into more specific ones:

- **RQ**: Does the use of programming learning tools based on AR improve the children's motivation and interest in programming?
    - **RQ1**: Does understanding and knowing the game and how it works intrinsically increase children's interest in programming?
    - **RQ2**: Does using AR interfaces improve children's motivation for programming?
    - **RQ3**: What is children's subjective perception of the ease of use, usefulness and intention of use of the game?

In order to answer these questions, an experiment has been proposed following the experimental process of Wohlin et al. [48] and formulating the experimental objective using the template provided in the Goal-Question-Metric (GQM) [5] as shown below:

*To **analyze** the RoboTIC serious game for learning programming **with the purpose of** evaluating the user experience and to know the subjective opinion **with regard to** ease of use, usefulness, intention, interest and motivation of this game, **from the point of view** of children attending the school **within the context of** an scenario where there is no background in programming knowledge nor AR.*

Therefore, the experiment aims to demonstrate the usefulness of the tool for acquiring knowledge to solve problems using computational thinking, the ease of use, and its motivational component. For this purpose, a learning activity has been designed for children with the profile mentioned above, and with reduced groups of participants to avoid distractions.

Fig. 4 graphically illustrates the experimental design followed in this learning activity, describing the elements involved in each phase of the evaluation.

The experiment consisted of 2 phases; a first phase of preparation and another phase of intervention that included pre-test, development and post-test.

During the preparation phase the participants were divided into 3 groups of four children each given the reduced age of the participants, in order to isolate them in a room during the experiment to reduce possible distractions and thus achieve a working group more focused in the experiment. In addition, a tutorial was given on the use of the game and the AR device, explaining the possibilities of interaction, the functioning of the game and the objective they had to reach to solve the level. Fig. 5 shows a photo taken during this phase with the 4 members of the first group.

In the intervention phase, each working group performed three tasks of the experiment: i) pre-test, ii) development and iii) post-test. The total duration of the experiment for each working group was between 20 and 30 minutes, considering that each participant had to perform the development task individually. For both the pre-test and the post-test, the participants completed surveys in which several questions were asked in order to evaluate certain variables. In both tests, questions that were not open-ended had to be answered in a 5-point Likert scale. The answer options were replaced by *emojis* to make them easier to be understood by the participants [24] (1: *strongly disagree* being the angriest emoji to 5: *strongly agree* being the happiest emoji):

- *pre-test*: the participant's demographic information was requested, i.e. sex, age, current education level and desired future education level. In addition, some general questions were asked in the context of programming, in order to know their personal experience and opinion about video games, computers, and more specifically, about programming. In the Table 1 are included the questions of the questionnaire for this pre-test.
- *post-test*: some general programming questions were repeated again (G5A, G8A-G11A) in order to determine whether they had changed their minds in any of their answers after the development phase. This test mainly included questions aimed at evaluating the participants' subjective perception of the game, inspired by the Technology Acceptance Model (TAM) [12] and analyzing the variables Perceived usefulness (PU), Perceived ease-of-use (PEOU) and Intention of use (ITU) of the model, as well as others aimed at learning about the participants' intrinsic motivation. Finally, some open-ended questions were included where participants could indicate what they liked most about the experience and what they liked least. In the Table 2 are included the questions of the questionnaire for this post-test.

During the development phase, participants had to solve a RoboTIC level where various elements of the game were used, such as AR stickers, instructions, use of loops and conditions (listed in Fig. 4). Thus, the level introduced a programming problem that had to be solved by a combination of the above elements (see video referenced in Section 3). The instructions allowed to define the sequence of commands to be executed at the level, while the loops and conditions allowed to define the execution flow.

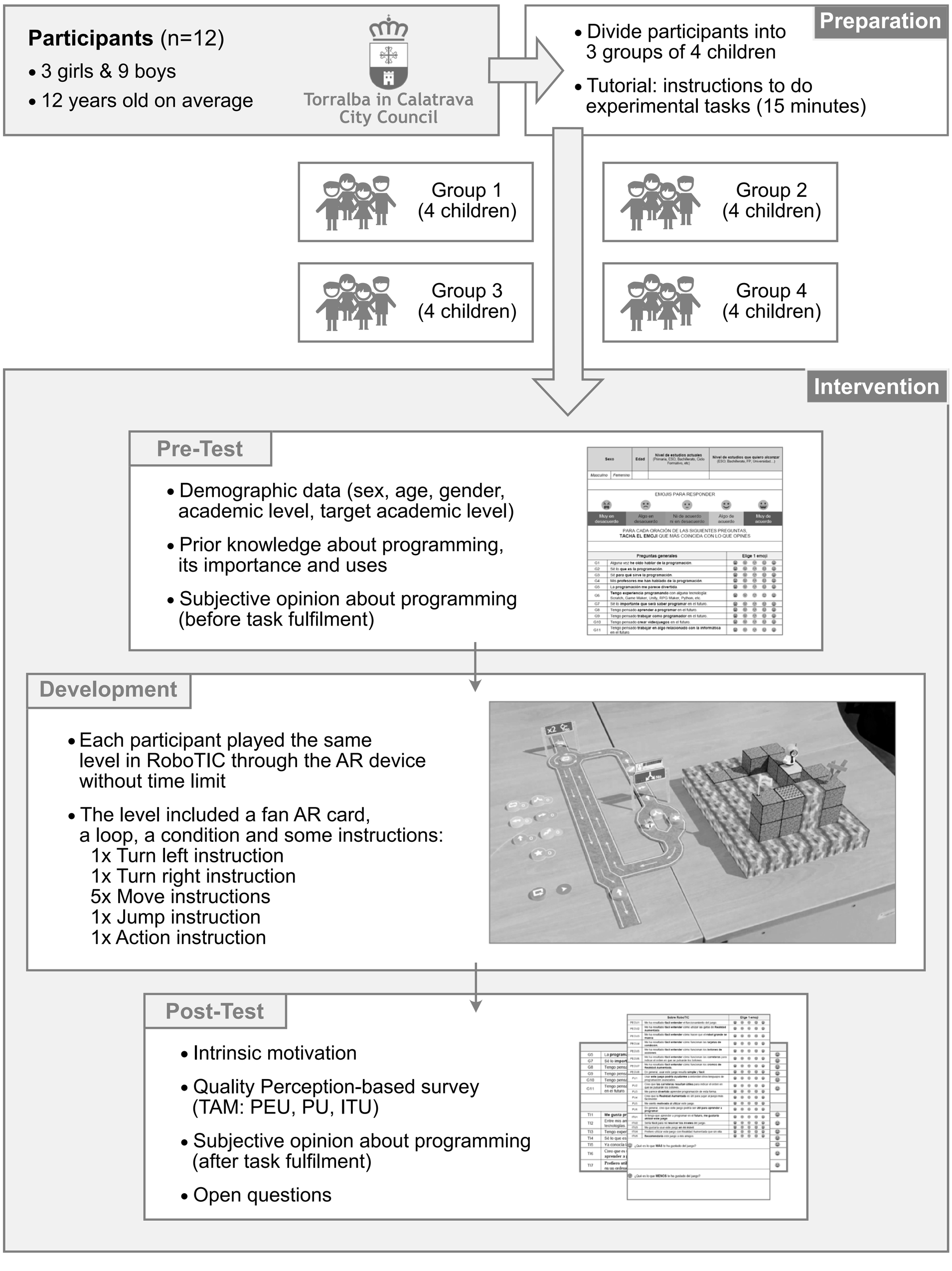


Fig. 4 Experimental design

Each participant had to put on the AR device and start solving the level individually. The Microsoft HoloLens device allows streaming video via the web in order to see what the person wearing the device is seeing at that moment. This was used to see what the participants were doing to solve the level and how they interacted with the game.

Fig. 5 Photo taken during the preparation phase of the experiment

4.3 Results

First, the data obtained in the pre-test were analyzed considering the answers whose value on the Likert scale was greater or equal to 3, obtaining percentages of participants from the grouping of similar questions. This analysis concluded that 92% of the participants wanted to pursue higher education, only 12.5% had heard about programming, knew what it is or what it is used for (G1-G4 items) and none of them had previously programmed (G6 item), didn't want to dedicate to programming in the future (G8-G9 items), didn't know its importance (G7 item) or didn't think it was fun (G5 item), although 33.33% wanted to dedicate to creating video games or to computing in the future (G10-G11 items).

After analyzing again the same questions in the post-test part after the development phase and having been able to test a level of RoboTIC, it was found that now 66.67% of the

Table 1 Items in the survey for the pre-test

Item	Item statement
G1	I've ever heard of programming.
G2	I know what is programming.
G3	I know what programming is used for.
G4	My teachers have told me about programming.
G5	I find programming fun.
G6	I have experience programming with some technology: Scratch, Game Maker, Unity, RPG Maker, Python, etc.
G7	I know how important it will be to know how to program in the future.
G8	I intend to learn to program in the future.
G9	I intend to work as a programmer in the future.
G10	I intend to make video games in the future.
G11	I intend to work on something related to computer science in the future.

Table 2 Items in the survey for the *post-test*

Item	Item statement
G5A	I find programming fun.
G8A	I intend to learn to program in the future.
G9A	I intend to work as a programmer in the future.
G10A	I intend to make video games in the future.
G11A	I intend to work on something related to computer science in the future.
AR1	I have understood what is Augmented Reality.
AR2	I already knew the Microsoft HoloLens or other Augmented Reality device.
AR3	I think it's useful and fun to use Augmented Reality to learn how to program.
AR4	I prefer to use tools that use gestures and Augmented Reality rather than keyboard and mouse to learn how to program.
PEOU1	I found it easy to understand how the game works.
PEOU2	I found it easy to understand how to use the Augmented Reality device.
PEOU3	I found it easy to understand how to make the robot to move.
PEOU4	I found it easy to understand how the condition cards work.
PEOU5	I found it easy to understand how the action buttons work..
PEOU6	I have found it easy to understand how roads work to indicate the order in which buttons will be pressed.
PEOU7	I found it easy to understand how the Augmented Reality stickers work.
PEOU8	In general, using this game is simple and easy.
PU1	Using this game could help me understand other advanced programming languages.
PU2	I think roads are useful to indicate the order in which the buttons will be pressed.
PU3	I think it's fun to learn programming this way.
PU4	I think augmented aeality is useful for playing the game more easily.
PU5	I feel motivated to use this game.
PU6	Overall, I think this game could be useful for learning how to program.
ITU1	If I have to learn to program in the future, I would like to use this game.
ITU2	It would be easy for me to solve the levels of the game.
ITU3	I would like to use this game on my smartphone.
ITU4	I'd rather use this game with Augmented Reality than without it.
ITU5	I would recommend this game to my friends.

participants thought programming was fun (item G5) and 62.5% plan to learn to program in the future, dedicate to computing or work creating video games (item G8-G11).

The conclusions drawn from the above variables are justified by applying the McNemar's test for two related samples [14], with the aim of demonstrating the difference between these

variables before (G5, G8-G11) and after (G5A, G8A-G11A) the development part during the intervention phase. These variables have been transformed into dichotomous variables where the values of the Likert scale between 1 and 2 would represent a disagree response from the participants, and the values between 3 and 5 an agree response. Thus, the test would give the results shown in the Table 3.

As can be observed, there are significant differences for the variables G5, G8, G9 and G11 ($p < 0.05$), thus confirming the impact that the development part has had on the participants. In the case of variable G10 there would be no significant differences ($p = 0.508$; $p > 0.05$), from which it can be assumed that the participants will not necessarily want to commit themselves to the creation of video games in the future.

Regarding the intrinsic motivation of the participants when using Augmented Reality tools, in the Table 4 the values for averages, standard deviations and medians of the related post-test items (ARx) are shown. The results obtained for items AR3 and AR4 show a mean score on the Likert scale above 4 indicating that participants were more predisposed to use augmented reality technologies to learn programming, even though they had never used an AR device before (item AR2). The AR1 item is directly related to the explanation given to the participants of what AR is previously in the preparation phase, demonstrating with a score on the Likert scale higher than 4 that they had understood the explanation.

In relation to the participants' subjective perception of ease of use (PEOU), utility (PU) and intention of use (ITU), the results obtained for the means, standard deviations and medians of the items of the post-test involved are shown in the Table 5, as well as the overall mean of each variable category. The results obtained show scores higher than 4 on the Likert scale for all cases except for the item PEOU4, whose standard deviation (0.79) is also far from that of the rest of the items. This may indicate that RoboTIC condition cards are one of the most difficult elements of the game to understand. On the contrary, the item PEOU3 presents a score of 5 with a standard deviation of 0.0, indicating a consensus among the participants on the understanding of the functioning of the movement of the main character from the instruction buttons. In particular, we would like to highlight the item PEOU6 related to the metaphor of roads and traffic signs, which allows us to validate the simplification of the notation ANGELA as suitable for the context of serious games.

The validity of such results can be ensured by a Wald-Wolfowitz runs test [49] that confirms that the responses obtained from the participants are randomly enough, i.e. the sample values are mutually independent. The results after running such test is shown in the last columns of the same table, where we can conclude that the majority of the values of the variables are randomly distributed ($p < 0.05$), with the exception of the PU5 and ITU3 variables and the PEOU1, PEOU3 and PEOU4 variables, not being possible to perform the

Table 3 McNemar test results for variables G5, G8, G9, G10 and G11 before and after the development phase

Items related	Sig.	Interpretation
G5 & G5A	0.008	Significant differences
G8 & G8A	0.016	Significant differences
G9 & G9A	0.008	Significant differences
G10 & G10A	0.508	Non-significant differences
G11 & G11A	0.008	Significant differences

Table 4 Mean and median values for the intrinsic motivation regarding the AR items. Standard deviations are shown in parentheses

Item	Mean	Median
AR1	4.33 (0.49)	4
AR2	1.08 (0.29)	1
AR3	4.58 (0.67)	5
AR4	4.42 (0.51)	4

runs test for these last three variables due to the reduced diversity of values obtained from the participants.

Finally, the open-ended questions were answered in a variety of ways, focusing mainly on the fact that the participants liked the graphic aspect of the game, namely the instruction buttons, the roads and the movement of the mini-robot through them.

4.4 Findings and suggestions

After conducting the learning activity, we can extract some valuable experience that could be transferred to a real class environment. Using RoboTIC in such environment would imply the design of a series of game levels of incremental complexity aimed to explain some of the programming concepts in the way they are usually structured:

Table 5 Mean and median values for the TAM framework variables; perceived ease-of-use (PEOU), perceived usefulness (PU) and intention of use (ITU), along with the runs test result and its interpretation. Standard deviations are shown in parentheses

Item	Mean	Median	Mean (global)	Sig.	Interpretation
PEOU1	4.17 (0.58)	4.0	4.35 (0.40)	-	Unable to compute
PEOU2	4.50 (0.52)	4.5		0.762	Randomly distributed values
PEOU3	5.00 (0.00)	5.0		-	Unable to compute
PEOU4	3.58 (0.79)	3.0		-	Unable to compute
PEOU5	4.50 (0.52)	4.5		0.364	Randomly distributed values
PEOU6	4.33 (0.49)	4.0		0.908	Randomly distributed values
PEOU7	4.50 (0.52)	4.5		0.130	Randomly distributed values
PEOU8	4.25 (0.45)	4.0		1.000	Randomly distributed values
PU1	4.50 (0.52)	4.5	4.49 (0.19)	0.762	Randomly distributed values
PU2	4.25 (0.45)	4.0		1.000	Randomly distributed values
PU3	4.67 (0.49)	5.0		0.051	Randomly distributed values
PU4	4.33 (0.49)	4.0		1.000	Randomly distributed values
PU5	4.75 (0.45)	5.0		0.012	Non-randomly distributed values
PU6	4.42 (0.51)	4.0		0.145	Randomly distributed values
ITU1	4.58 (0.51)	5.0	4.67 (0.13)	0.405	Randomly distributed values
ITU2	4.67 (0.49)	5.0		0.206	Randomly distributed values
ITU3	4.75 (0.45)	5.0		0.012	Non-randomly distributed values
ITU4	4.50 (0.52)	4.5		0.130	Randomly distributed values
ITU5	4.83 (0.39)	5.0		0.843	Randomly distributed values

1. Sentences: players would define the set of instruction buttons that would serve to complete a simple game level, for example, by moving the robot just two tiles forward. These instructions are executed sequentially, resembling how programs are executed line by line. This is the same as programmers would write their programs using code.
2. Combination of sentences: in more complex levels, the players would need to use a combination of all the available instruction buttons to complete the level. This would imply that the players plan and evaluate the level beforehand to achieve the best combination of instructions to solve the game level.
3. Conditional sentences: next concept to understand would be conditional sentences. These ones would allow students to complete the game levels depending on random events happening in the game and that require specific flow control.
4. Conditional sentences (nested): as before, more complex in-game events would require more complex conditional sentences. In these levels, the user is introduced in how to nest conditional sentences to solve more advanced problems (game levels).
5. Conditional sentences (AR cards): another way to explain flow control through conditional sentences is by using AR cards that can be used to trigger some behavior in the game levels. Therefore, this mechanism can be used to motivate the students learning this concept.
6. Loops: after introducing conditions, students can use the concept of loops to accomplish repetitive tasks. This concept is the same one used in programs made by programmers.
7. Loops (nested): as before, more complex tasks would require nesting a loop within another one.
8. Functions: finally, the last game levels would involve using the function concept to execute instruction sequences made for other levels but which can be reused for the new level. In this way, students can easily understand the reuse and modularization of programs.

Then, the teacher would present the concept in class and then would provide the game to the students so that they can play the related game level to better understand the explained concept. If the students complete the level of play, it would mean that they have understood the related concept successfully.

4.5 Study limitations

Despite the results obtained, there are some limitations and risks for the external validity of the study. In terms of the nature of the participants, given the small size of the sample treated, it has not been possible to conduct an exhaustive hypothesis validation, leading to a descriptive statistical analysis. Regarding the experimental task performed, although the game level that the participants have tested is complete enough to include all the elements of the game and to deal with all the available programming concepts, experimental tasks that respect the expected learning flow are considered, so that the participants learn programming concepts incrementally.

5 Conclusions and future work

In this work we have presented RoboTIC, a serious game oriented to learning programming and improve the computational thinking for children who have no coding skills. The different components that are integrated into the RoboTIC architecture have been introduced

within the context of the learning possibilities it provides thanks to the use of AR and the game mechanics designed to motivate students. This layered architecture is scalable enough to add new levels and multimedia resources that make possible to address new programming concepts and techniques. Such scalability is achieved through the usage of different data files that allow to create new game levels and load them without having to rebuild the tool. In case of requiring more advanced resources, the architecture facilitates adding new game elements, such as new AR cards or game level blocks.

The analysis of the results obtained after the evaluation of RoboTIC concludes answering the research questions posed at the beginning of this research work. Thus, with regard to the children's motivation and interest in programming (RQ1) it is concluded that the students have felt more motivated after playing RoboTIC and have been more receptive to learning programming. Regarding AR, the study reveals that its use as an immersive technology improves the children's motivation towards programming, making learning more attractive (RQ2). Finally, the analysis of the subjective perception towards ease of use, usefulness and intention to use RoboTIC shows that the serious game proposed in this work is easy to understand and use (RQ3). The affirmative answers to the raised research questions imply, therefore, returning to the initial research question: *Does the use of programming learning tools based on augmented reality improve the children's motivation and interest in programming?*, which can be answered affirmatively considering, however, the limitations previously stated in Section 4.5.

The evaluation of RoboTIC and the execution of the experiment have allowed us to extract some valuable experience that could be reused in real class environment. Thus, we have proposed a possible learning path on how to use RoboTIC in the classroom by firstly explaining the programming concept and then leaving the students play the related game level to check if they have really understood such concept.

As lines of future work, the inclusion of new programming concepts are considered to be included in RoboTIC, such as the use of functions to introduce structured programming or recursion as a more clean way to reduce program complexity when coding. Migrating RoboTIC to another hardware platform with a lower cost than the Microsoft HoloLens will be also addressed so that the tool becomes more accessible. Thus, it is considered a simplified version that works on smartphones using some technology such as ARCore (Android) or ARKit (iOS), or using physical marks that are employed for placing the game levels using AR; in this case, we would have to assess new NUIs that allow for easy use of the system. In the same way, we also intend to introduce RoboTIC in education centers that are specialized in teaching coding skills and robotics. This will allow us to conduct a more exhaustive study of the current version of RoboTIC, involving a greater number of students. Finally, the integration of another gamification techniques is proposed to keep players motivated, such as, for example, medals and achievements depending on the student's performance when solving a level.

Acknowledgements This work 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. The authors would like to thank Pablo Gutiérrez Caravantes for coordinating the experiments carried out in the youth center of Torralba in Calatrava (Ciudad Real, Spain) and the undergraduate students that participated in the project *Telefónica Talentum* for the development of the first software prototype.

Compliance with Ethical Standards

Conflict of interests The authors declare that they have no conflict of interest.

References

1. Abelson H, Goodman N, Rudolph L (1974) LOGO manual
2. Bloem J, Van Doorn M, Duivestein S, Excoffier D, Maas R, Van Ommeren E (2014) The fourth industrial revolution. Things Tighten 8:1–40
3. Burgess N, Maguire EA, O'Keefe J (2002) The human hippocampus and spatial and episodic memory. Neuron 35(4):625–641
4. Bacca J, Baldiris S, Fabregat R, Graf S (2014) Augmented reality trends in education:, A systematic review of research and applications. Educational Technology and Society 17(4):133–149
5. Basili VR, Caldiera G, Rombach HD (1994) The goal question metric approach. Encyclopedia of Software Engineering 2:528–532
6. BENTRAD S, Meslati D (2011) Visual programming and program visualization – towards an ideal visual software engineering system –. ACEEE International Journal on Information Technology 1:56–62
7. Colombo AW, Karnouskos S, Kaynak O, Shi Y, Yin S (2017) Industrial cyberphysical systems: a backbone of the fourth industrial revolution. IEEE Ind Electron Mag 11(1):6–16
8. Connolly TM, Boyle EA, MacArthur E, Hainey T, Boyle JM (2012) A systematic literature review of empirical evidence on computer games and serious games. Computers & education 59(2):661–686
9. Diehl S (2007) Software visualization: visualizing the structure, behaviour, and evolution of software. Springer Science & Business Media
10. Dunleavy M, Dede C (2014). In: Spector JM, Merrill MD, Elen J, Bishop MJ (eds) Augmented reality teaching and learning, 4th eds. New York, Springer, pp 735–745
11. da Silva Esteves AM, Santana ALM, Lyra R (2019) Use of augmented reality for computational thinking stimulation through virtual
12. Davis FD (1993) User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. International journal of man-machine studies 38(3):475–487
13. Elshiekh R, Butgerit L (2017) Using gamification to teach students programming concepts. Open Access Library Journal 4(08):1
14. Eliasziw M, Donner A (1991) Application of the mcnemar test to non-independent matched pair data. Statistics in medicine 10(12):1981–1991
15. Fraser N (2015) Ten things we've learned from Blockly. In: 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond), IEEE, pp 49–50
16. Figueiredo M, Cifredo-Chacón MÁ, Gonçalves V (2016) Learning programming and electronics with augmented reality. In: International Conference on Universal Access in Human-Computer Interaction, Springer, pp 57–64
17. Gallego-Durán FJ, Villagrá-Arnedo CJ, Llorens Largo F, Molina-Carmona R (2017) Plman: A game-based learning activity for teaching logic thinking and programming. International Journal of Engineering Education
18. Gouws LA, Bradshaw K, Wentworth P (2013) Computational thinking in educational activities. In: Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13, ACM Press, New York, New York, USA, pp 10
19. Hidalgo-Céspedes J, Marín-Raventós G, Lara-villagrán V (2016) Learning principles in program visualizations: a systematic literature review. In: Proceedings of the 46th Annual Frontiers in Education (FIE) Conference. IEEE, pp 1–9
20. Ibanez MB, Di-Serio A, Delgado-Kloos C (2014) Gamification for engaging computer science students in learning activities: a case study. IEEE Transactions on learning technologies 7(3):291–301
21. Jee HK, Lim S, Youn J, Lee J (2014) An augmented reality-based authoring tool for e-learning applications. Multimedia Tools and Applications 68(2):225–235
22. Jimenez-Diaz G, Gonzalez-Calero PA, Gomez-Albarran M (2012) Role-play virtual worlds for teaching object-oriented design: the viRPlay development experience. Software:, Practice and Experience 42(2):235–253
23. Krpan D, Mladenović S, Ujević B (2018) Tangible programming with augmented reality. In: 12th International Technology, Education and Development Conference
24. Kaye LK, Malone SA, Wall HJ (2017) Emojis: insights, affordances, and possibilities for psychological science. Trends Cogn Sci 21(2):66–68
25. Kim TJ, Huh JH, Kim JM (2018) Bi-directional education contents using vr equipments and augmented reality. Multimedia Tools and Applications 77(22):30089–30104
26. Kumar B, Khurana P (2012) Gamification in education-learn computer programming with fun. International Journal of Computers and Distributed Systems 2(1):46–53

27. Kaufmann H, Schmalstieg D (2002) Mathematics and geometry education with collaborative augmented reality. In: ACM SIGGRAPH 2002 conference abstracts and applications, pp 37–41
28. Knight C, Munro M (2000) Virtual but visible software. In: Proceedings of the IEEE International Conference on Information Visualisation, pp 198–205 https://doi.org/10.1109/IV.2000.859756
29. Majumdar A (2018) The hour of code: an initiative to break the barriers of coding. XRDS 24(3):12–13. https://doi.org/10.1145/3186711
30. Myers BA (1990) Taxonomies of visual programming and program visualization. Journal of Visual Languages & Computing 1(1):97–123
31. Maloney J, Resnick M, Rusk N, Silverman B, Eastmond E (2010) The scratch programming language and environment. ACM Trans Comput Educ 10(4):1–15
32. Milne I, Rowe G (2004) Ogre: Three-dimensional program visualization for novice programmers. Educ Inf Technol 9(3):219–237
33. Mathur AS, Ozkan BK, Majumdar R (2018) Idea: An Immersive Debugger for Actors. In: Proceedings of the 17th ACM SIGPLAN International Workshop on Erlang, St. Louis, MO USA, ACM, pp 1–12
34. Paliokas I, Arapidis C, Mpimpitsos M (2011) PlayLOGO 3D: A 3D Interactive Video Game for Early Programming Education: Let LOGO Be a Game. In: 2011 Third International Conference on Games and Virtual Worlds for Serious Applications, Athens, 2011, pp. 24–31. https://doi.org/10.1109/VS-GAMES.2011.10
35. Parmar D, Isaac J, Babu SV, D'Souza N, Leonard AE, Jörg S, Gundersen K, Daily SB (2016) Programming moves: Design and evaluation of applying embodied interaction in virtual environments to enhance computational thinking in middle school students. In: 2016 IEEE Virtual Reality (VR). IEEE, pp 131–140
36. Paliokas I, Arapidis C, Mpimpitsos M (2011) PlayLOGO 3D: A 3D Interactive Video Game for Early Programming Education: Let LOGO Be a Game. In: 2011 Third International Conference on Games and Virtual Worlds for Serious Applications,Athens, 2011, pp 24–31 https://doi.org/doi:10.1109/VS-GAMES.2011.10
37. Piteira M, Haddad SR (2011) Innovate in your program computer class. In: Proceedings of the 2011 Workshop on Open Source and Design of Communication - OSDOC '11, ACM Press, New York. New York, USA, pp 49
38. Robertson GG, Card SK, Mackinlay JD (1993) Information visualization using 3d interactive animation. Commun ACM 36(4):57–71
39. Schwab K (2017) The fourth industrial revolution. Currency
40. Sarkar SP, Sarker B, Hossain SA (2016) Cross platform interactive programming learning environment for kids with edutainment and gamification. In: 19Th international conference on computer and information technology, ICCIT, IEEE, pp 218–222
41. Sajaniemi J, Kuittinen M (2003) Program animation based on the roles of variables. In: Proceedings of the 2003 ACM symposium on Software visualization, San Diego, California, USA, ACM, pp 7–ff
42. Schez-Sobrino S, García MÁ, Gómez C, Vallejo D, Lacave C, Glez-Morcillo C, Molina AI, Albusac JA, Redondo MÁ (2019) ANGELA: A novel approach of graphic notation based on the metaphor of road signs to facilitate the learning of programming. In: Proceedings of the 7th International Conference on Technological Ecosystems for Enhancing Multiculturality
43. Schez-Sobrino S, Gmez-Portes C, Vallejo D, Glez-Morcillo C, Redondo MÁ (2020) An intelligent tutoring system to facilitate the learning of programming through the usage of dynamic graphic visualizations. Appl Sci 10(4):1518
44. Teyseyre AR, Campo MR (2009) An overview of 3d software visualization. IEEE transactions on visualization and computer graphics 15(1):87–105
45. Teng CH, Chen JY, Chen ZH (2018) Impact of augmented reality on programming language learning: Efficiency and perception. J Educ Comput Res 56(2):254–271
46. Vahldick A, Mendes AJ, Marcelino MJ (2014) A review of games designed to improve introductory computer programming competencies. In: 2014 IEEE Frontiers in education conference (FIE) Proceedings. IEEE, pp 1–7
47. Vasilopoulos IV, van Schaik P (2018) Koios: design, development, and evaluation of an educational visual tool for greek novice programmers. J Educ Comput Res, 0(0)
48. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experiment Process. pp 31–39
49. Wald A, Wolfowitz J (1940) On a test whether two samples are from the same population. The Annals of Mathematical Statistics 11(2):147–162

50. Wing JM (2006) Computational thinking. Commun ACM 49(3):33–35
51. White R, Tian F, Smith P (2016). In: Code lab: a game that teaches high level programming languages. In: Proceedings of the 30th International BCS Human Computer Interaction Conference: Fusion!, Poole. BCS Learning & Development Ltd., United Kingdom, pp 1–8, https://doi.org/10.14236/ewic/HCI2016.76

2.1.3 A modern approach to supporting program visualization: from a 2D notation to 3D representations using augmented reality

- Title: A modern approach to supporting program visualization: from a 2D notation to 3D representations using augmented reality [48]
- Authors: Santiago Sánchez, María de los Angeles García, Carmen Lacave, Ana Isabel Molina, Carlos González, David Vallejo, and Miguel Ángel Redondo
- Type: Journal
- Journal: Multimedia Tools and Applications
- Publisher: Springer Nature
- E-ISSN: 1573-7721
- Year: 2020
- DOI: 10.1007/s11042-020-09611-0
- Category: Computer Science, Software Engineering
- Impact Factor (2019): 2.313
- JCR Ranking: Q2
- Related to the current research topic: Yes

Multimedia Tools and Applications
https://doi.org/10.1007/s11042-020-09611-0

A modern approach to supporting program visualization: from a 2D notation to 3D representations using augmented reality

Santiago Schez-Sobrino[1] · María Á. García[1] · Carmen Lacave[1] · Ana I. Molina[1] · Carlos Glez-Morcillo[1] · David Vallejo[1] · Miguel Á. Redondo[1]

Received: 4 June 2019 / Revised: 30 June 2020 / Accepted: 12 August 2020 /
Published online: 03 September 2020

Abstract

The visualization of programs and algorithms has been demonstrated to be essential when learning to program. Nevertheless, existing graphic representations require a high level of abstraction that most beginner programmers cannot understand. Current state-of-the-art approaches provide promising alternatives, but a significant part leaves the advantages of graphic representation in the background. These advantages include abstracting the source code by means of symbols that make them easier to understand without previous training. This work introduces the evolution of a 2-D graphic notation to a 3-D environment, which represents an improvement to a complete platform for collaborative programming learning through problem solving, named COLLECE-2.0. This improvement provides the platform with capabilities to visualize programs through augmented reality by using a new set of graphic representations, which are based on roads and traffic signs in the context of programming learning. These visual models have been evaluated by Computer Science students to know whether the proposed notation is intuitive and useful. The obtained results show that the proposed notation is suitable for representing programming concepts and easy to understand. We also present a series of improvements, integrated as a new subsystem in the aforementioned platform, which allows the automatic construction of 3-D visualizations on an augmented reality environment. These visualizations use the proposed notation and leverage the scalability and architecture of COLLECE-2.0.

Keywords Computer-Supported Collaborative Learning (CSCL) · Program visualization · Programming learning · Three-dimensional displays · Augmented reality · Eclipse

✉ Santiago Schez-Sobrino
santiago.sanchez@uclm.es

[1] Department of Information Technologies and Systems, University of Castilla-La Mancha (Spain), Paseo de la Universidad 4, 13071, Ciudad Real, Spain

1 Introduction

Programming is a field which is constantly advancing in parallel with current times and society. Learning how to code must be addressed effectively, given the number of jobs that are now and will be emerging in the future. This requires programming skills [14].

According to the employment portal GlassDoor [17], the latest 2018 reports reveal that 9 of the top 25 jobs in the U.S. required programming skills. By 2026, an estimated 58% of jobs in Science, Engineering, Mathematics, and Information Technology (STEM) will require programming skills [56]. However, only 10% of graduates in these areas do so in computer science [41]. There is, therefore, a need to increase the number of graduates in disciplines involving programming skills to meet the current demand.

The learning of programming presents multiple difficulties for students who start programming. Among the main ones, the use of variables, the use of control structures, the correction of syntax errors, the modularization of code through functions, or the operation of lists or arrays stand out [5]. In this context, most of them are semantic-related. These difficulties are a result of a number of reasons, such as the students' lack of capacity to deal with abstractions, the fact that they do not manage to understand the real effect that changes in code generates, or their inability to express solutions by means of a programming language, among others. This learning process can be improved by using abstractions that facilitate the understanding of these programming concepts. Thus, the use of graphic representations aimed at visualizing the structure and behavior of programs and algorithms arises.

Various proposals have been made to facilitate the learning process and reduce the level of abstraction necessary to understand programs and algorithms [43], fulfilling the cognitive objectives of level 2 of the taxonomy proposed by Bloom [4]. On the one hand, tools that strengthen the student participation through a Computer Supported Collaborative Learning or CSCL [29] approach can be used, such as COLLECE [7], Cole-Programming [24], and COALA [23]. On the other hand, techniques for visualizing programs and algorithms which facilitate the understanding of concepts related to programming can be employed, such as Greedex [63] and VisBack [31].

Regarding this second approach, the visualization of programs and algorithms helps to understand programming concepts thanks to the provided level of abstraction. The difference between the two concepts is noteworthy [44]. The visualization of programs allows to graphically represent source code, identifying data structures and other elements, while the visualization of algorithms tries to graphically represent the behavior of programs in a more abstract way, often requiring human intervention to create such visualizations. These terms belong to a family of concepts named software visualization, which, in turn, is part of the subset named information visualization [13].

There is empirical evidence to state that this type of visualization positively affects the student's motivation [21, 60]. This provides certain benefits, such as a better understanding of loops and other control structures [32], and ease of understanding how programs, in specific domains as recursive programming, work [30]. The participation in class gets also improved [58], as well as the understanding of concepts related to programming [57], even when only graphic animation is shown without the teacher providing any explanation about it [62]. As a consequence, experiments with students who have used mechanisms for visualizing programs and algorithms have been conducted, and their grades were higher than those obtained by students using more traditional learning approaches [10, 51].

Furthermore, visualizations based on 3-D graphic representations provide multiple benefits over traditional 2-D, since the third dimension allows more information to be displayed

[45], provides greater realism by using representations which are closer to the real world [54], and produces a more effective use of the user's spatial memory in order to improve the recall and recognition of certain behaviors found in programs through the human subconscious [8, 27]. By means of this three-dimensional space, the use of augmented reality techniques introduces benefits when visualizing programs and algorithms in a more natural way. This results in a richer experience for the student, given the potential advantages of using this approach in educational contexts [1, 15].

In addition, the process of understanding programming concepts can be facilitated by establishing relationships between the concepts to be explained and others already known by the students, through the use of visual metaphors [20].

Given the introduced context, this work presents an improvement to COLLECE-2.0 [49, 50], one of the tools that we have developed to facilitate programming learning in a collaborative way with multiple students, in real-time, and remotely. This improvement includes a visualization module that allows us to exploit new paradigms of interaction through 3-D graphic displays, taking advantage of the use of augmented reality techniques. The first version of the system, named COLLECE, did not include visualization or real-time collaboration mechanisms, but was formally evaluated with Computer Science (CS) students in a real environment, obtaining satisfactory results that made the use of the tool viable in a programming learning environment [6, 36]. We aimed at extrapolating these results to the new version proposed in this manuscript, as it mainly offers a series of improvements on the shortcomings detected in version 1.0.

The integrated visualization module relies on a 3-D metaphor, which is also proposed and allows first-year CS students to understand programming concepts in a programming learning environment. In the case of Spain and other EU countries, programming is not included in the elementary and high school curricula, so these students are not assumed to have any prior programming knowledge when they begin their university studies. The proposed 3-D metaphor is easily extensible to support other education levels, such as those involving young students interested in programming through more entertaining experiences like games. Also, the metaphor is intended to work within the visualization module as a tool to complement the teaching strategies applied to the teaching of programming in its early stages.

In previous developments, we already proposed other two-dimensional representations based on trees to express backtracking algorithms [31] and *barrels* for the case of the knapsack problem [63]. Now, we propose a metaphor based on roads and traffic signs, which facilitates the visualization of programs by using commonly known real-world analogies. This metaphor has been selected after a study with CS students and teachers, and we have obtained positive results through its evaluation with two groups of CS students. Finally, the metaphor has been integrated into the COLLECE-2.0 visualization module considering the improvements proposed after its evaluation.

The main contributions of this article are as follows:

- An improvement over COLLECE-2.0, a collaborative platform for problem-based learning of programming, is introduced, which is related to the visualization of programs using a notation based on the metaphor of roads and traffic signs.
- The proposed metaphor is intended to complement the learning process by facilitating the learning of programming. Such metaphor has been evaluated with groups of CS students, along with the new improvements made after the analysis of the obtained results.

– The new notation makes use of a 3-D environment through an augmented reality-based visualization, implemented in COLLECE-2.0 as a new feature that leverages its architecture to automatically generate the graphical visualization from source code.
– Natural interaction mechanisms are introduced to manipulate the visualization, together with the support for the visualization of the software metrics related to the graphically represented programs. Plus, the solution is scalable enough to i) add new graphic representations, ii) change them to make future improvements, and iii) replace the visualization device.

The rest of the paper is organized as follows. Section 2, describes some similar graphic representations based on different metaphors; also, we introduce the previous work on which this paper is based. Section 3 focuses on objectives to be defined when proposing a new notation and on the notation itself. Section 4 describes the evaluation of the proposed notation made with groups of CS students. Section 5 presents the improvements made to the visualization according to the evaluation, along with a new 3-D visualization approach based on the previous results and on the existing limitations of 2-D visualizations. Section 6 describes the implementation details to achieve the visualization system integrated in COLLECE-2.0 according to the notation presented. Section 7 discusses the results obtained and the limitations of the evaluation, the metaphor and the tool. Finally, Section 8 draws some final conclusions and suggests possible lines of future work.

2 Related work and background

2.1 Visualizations and metaphors

The taxonomy defined by Myers [38] distinguishes between *visual programming*, where programs are created using graphic representation techniques, and *program visualization*, where programs are represented by graphic elements to detail some aspects of either the program or its execution. In both cases, different visual metaphors are contemplated in an attempt to abstract the user from the representation of certain programming concepts.

In the field of visual programming, we find the Nassi-Shneidermann programming diagrams of [40] used to represent programs in a structured way similarly to flowcharts. Bischoff et al. [2] proposes a set of icons organized in the form of flow diagrams to facilitate programming of industrial machines. In this line, and in a more graphic way, Vasilopoulos and Van Schaik [59] propose a visual programming system that uses real-world metaphors to build programs, such as telephones to call functions, boxes for variables, and arrows next to people to represent entry and exit commands, among others.

In the case of program visualization, a classification is established according to the information represented and its nature, which may be *static* or *dynamic* depending on whether or not the visualization provides information at program runtime, respectively. According to this classification, there are several tools that provide this support through different metaphors used to represent the visualization of programs and algorithms graphically. Regarding the dynamic visualization, we find the work of Rowe and Thorburn [46], which proposes a tool that uses a two-dimensional grid to represent the state of the program memory, which changes throughout its execution. In a similar way, Gajraj et al. [16] present a system that shows visual explanations about the sentences of the program. These explanations are based on the representation of basic concepts, such as assignments of data to variables, through relationships between containers. More abstractly, we find the

set of metaphors presented in [47] to represent the execution of programs visually. These metaphors represent the possible roles that a variable can take, for instance, provided that it is a variable that will increase in a loop, a series of footprints will be used in the ground with the past and future values that the variable will take, however if it is an array, it will be made by means of a set of tombs with the values of the array sculpted on them. A more extended review for program visualization and visual programming can be found in several works like the surveys by [9] and [11].

In the field of 3-D graphics, there are also other works that aim to facilitate programming learning through visualizations. This is the case of Milne and Rowe [35], who use three-dimensional animations involving pipes and cubes to represent data assignments between variables, as well as other concepts such as planes to represent scopes of functions, among others. In a more abstract way, Jimenez-Diaz et al. [22] use avatars and 3-D animations in order to represent programming concepts, such as message passing between objects (avatars). Also, in a more advanced way using Virtual Reality techniques, Mathur et al. [33] propose a system that facilitates the understanding of concurrent programming concepts through actors and 3-D boxes.

The effectiveness of this type of visualizations, when learning to program, results in very different opinions. Among the favorable ones, we highlight the results obtained in [12, 25, 32], where different experiences were conducted with students who improved their understanding of the algorithms proposed and other elements of the programming languages. On the contrary, the results shown in [21, 39] offer a less positive view of how the visualization is used, rather than focusing on the quality of the visualization, the teacher's difficulty in creating them and the efforts needed to keep the students interested during class.

On the other hand, the effectiveness of visualizations can be improved using augmented reality techniques, as presented in [1], where some benefits that can contribute to the process of learning programming are concluded, such as learning gains, motivation, interaction and collaboration. An example of this is shown in [52], where printed markers are used to teach concepts of graphic programming with OpenGL using augmented reality, and in [53], where the system was evaluated with students, obtaining favorable results when the students were involved in learning tasks, resulting in a more motivating experience for them.

Thus, in this work we propose a metaphor that is attractive enough to maintain the motivation and interest of the students during the classes, as well as easy to understand and simple to be generated due to the automatic generation offered by the system.

2.2 COLLECE-2.0

As mentioned above, this paper proposes an extension of COLLECE-2.0[1] to provide support for the visualization of programs. COLLECE-2.0 is a platform based on the Eclipse IDE, which provides a set of features that facilitate programming learning through the participation and remote collaboration of the users to solve a proposed problem. These features allow the users to be aware of the environment that surrounds them within the system, at any time, and cooperate with one another to achieve a functional solution to the addressed problem, as well as to learn from what they see their peers do.

With this approach, COLLECE-2.0 incorporates the following features, summarized in Fig. 1 and superimposed on a screenshot of the tool:

[1]The software is available for download from http://blog.uclm.es/grupochico/proyecto-iapro/collece-2-0/

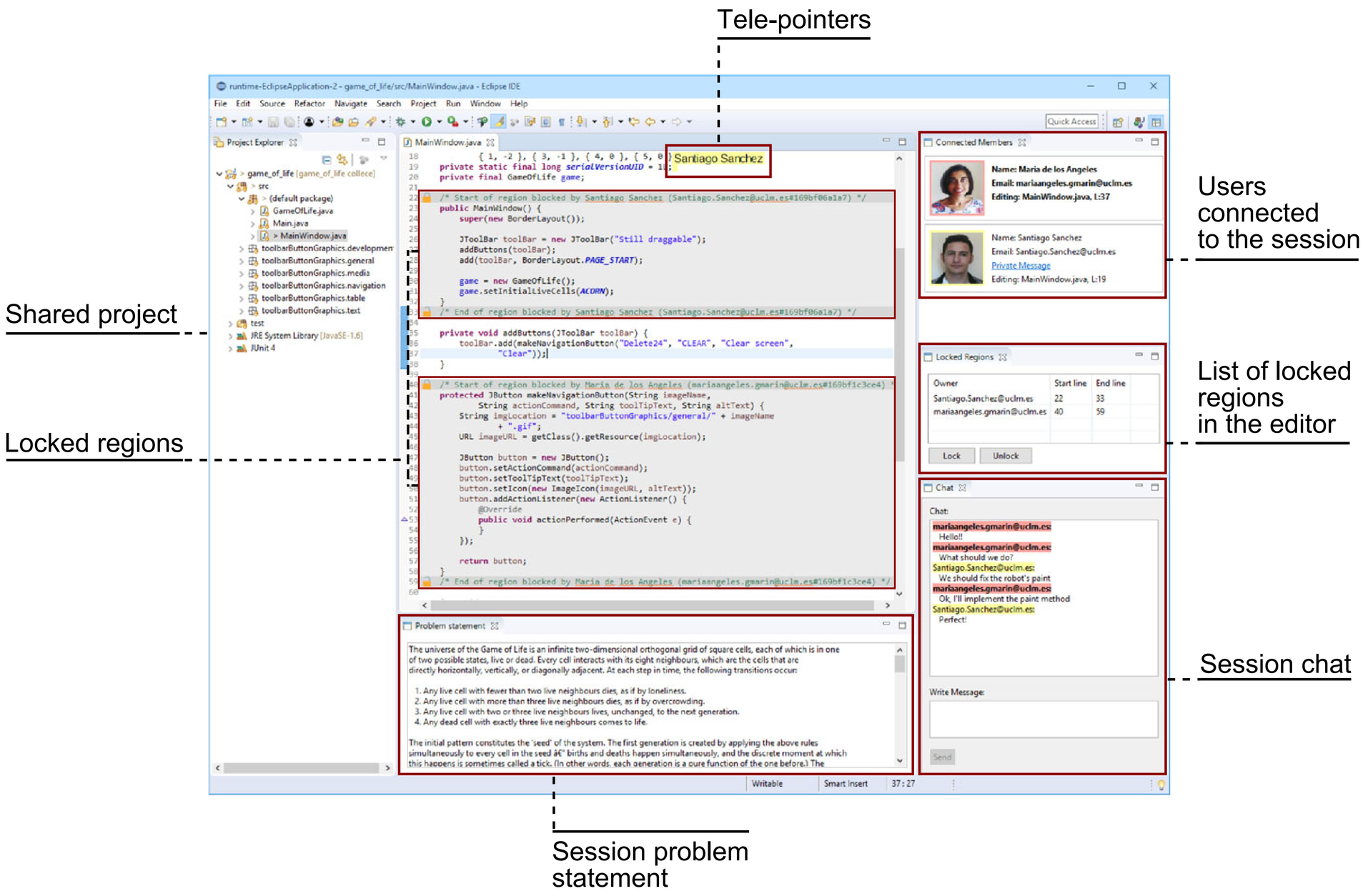


Fig. 1 A snapshot of COLLECE-2.0 with two members connected to a work session. Superimposed on the image, the main features provided by the platform are shown

- **Modularity**. The tool is easily installed as a plugin through the Eclipse repository system. Thanks to this, the tool provides robust extensibility through other plugins.
- **Working sessions**. The way to organize the work with the tool is based on sessions where multiple users connect and work simultaneously. These sessions are directly associated with a programming problem to be solved and can be created by any user.
- **Multi-user edition**. COLLECE-2.0 provides a low-latency, real-time collaborative editing system that allows for fluid editing, so that multiple users can edit the same files concurrently.
- **Use of tele-pointers**. Users can know who is editing which section of the file at any time, using tele-pointers that color the code regions associated with a particular user in the editor.
- **Blocking of regions**. COLLECE-2.0 prevents other users from modifying sections of code within a file, thus preventing conflicts and improving the coordination between the users. Blocking regions allows this by shading lines of code that are blocked, which means that they belong to a user.
- **Communication**. The tool provides an instant messaging mechanism which identifies the users who are participating in the session by using a color and their names. This communication can be public (to all the members of the session) or private (to a specific user).
- **Shared projects**. COLLECE-2.0 makes intensive use of repository systems to persistently maintain the state of the code-related projects associated with the sessions. The users benefit directly from this by being able to share any remote repository (for example, from a provider such as Bitbucket or GitHub) in a work session with other users.

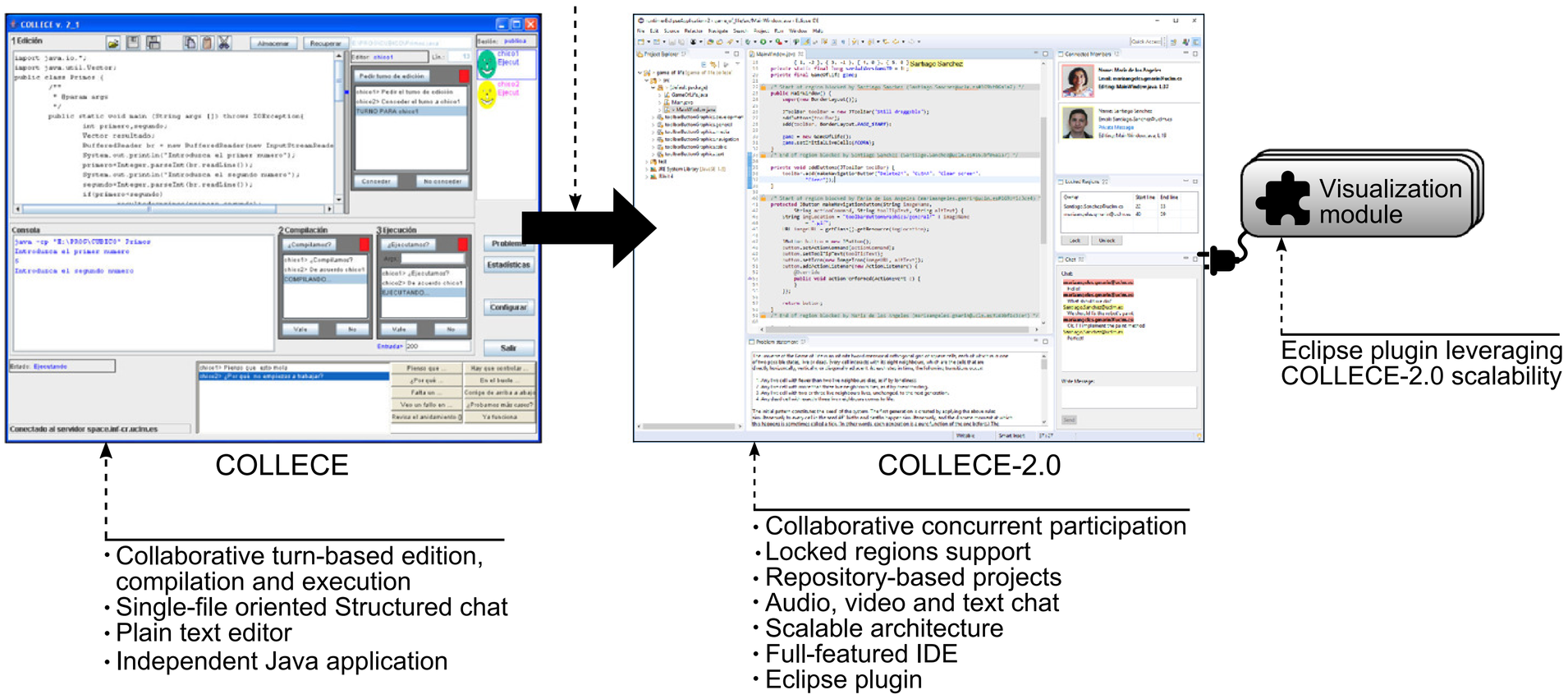


Fig. 2 Evolution from COLLECE to COLLECE-2.0 along with the major changes, prior to the evaluation and improvements

This tool is based on a previous version, called COLLECE [7], which allowed collaborative editing of source code in Java. This collaboration was done by requesting different types of turns to edit, compile and execute the shared file, as well as the possibility to communicate with other users through a structured chat.

Figure 2 shows an image of COLLECE-2.0 from screenshots that represent the evolution of the tool. This figure starts from COLLECE towards COLLECE-2.0 and the improvement presented in this work. Likewise, it indicates the evaluation conducted of the tool to motivate the transition to the new version.

3 Road-based notation

3.1 Objectives

When a graphic representation to visualize programs and algorithms is proposed, the objectives to be pursued must be considered, which are directly related to facilitate the learning of programming according to the difficulties and challenges that the students find during the learning process [5]. Hence, in this work we have established three fundamental objectives which the graphic representations must fulfill: i) easy for the student to understand, ii) easy for the teacher to use during lectures and iii) motivating for the students. It should be noted that the proposed metaphor is conceived as a tool to complement the teaching strategies applied to the teaching of programming in its early stages.

Apart from the main objectives, it is mandatory to clarify that the objective audience of the proposed metaphor is formed by those students who have never had contact with programming, that is, those students who start programming in their first year of Computer Science. This audience is also made up of students from other university degrees related to engineering and other technical fields that require some knowledge of programming. In both cases, these students do not usually have programming knowledge because in Spain

and other EU countries programming is not included in the elementary and high school curricula.

Section 3 describes the proposed representation and the conducted quasi-experiment in order for the students to evaluate this notation, considering the previous objectives as well as the improvements integrated after the analysis of the results obtained. The ultimate goal is to provide some initial results that enable extended evaluations in the future.

3.2 Initial proposal

The graphic representations designed in this research reflect a direct association between the code statements used in the programming language and the representations themselves. It should be noted that the use of the metaphor is understood within programming learning environments where complex programs are not proposed, but rather programs that represent concepts more specific to be constructed with a reduced number of code sentences. Thus, the visualizations built with the proposed graphic representations are simple enough for the metaphor not to present scalability problems in this type of environment. Other environments, oriented to professional use, are out of the scope of this work.

The notation is initially aimed at providing a static representation of the program and is intended to be language-agnostic, although the selection of sentences has been made from Java programming language as it is one of the most widely used in education and industry [61].

When identifying the minimum set of instructions necessary to represent a program in Java, we obtain five types of instructions that can be graphically represented: i) condition statements, ii) loop sentences, iii) function definition, iv) function returns and v) evaluation of expressions. There are many other language statements that have been omitted from this set, such as the use of break and continue, which are not included in the graphic representations in that they are unconditional jumps. This is because learning programming at the university is done following the paradigms of structured programming and modular programming. Also, a similar statement would be the case-based selection statement, or *switch*; in the introductory programming courses, the main aim is to teach programming independently of the programming language, so that the multiple selection statement is usually presented as a generalization of the conditional statement in which the break is not required. On the other hand, and in order to abstain the user from the details of language implementation, no distinction is made between the different types of loops. Thus, the metaphor aims to abstract the student from the syntax of the programming language by focusing mainly on the meaning of the instructions, i.e., semantics.

To decide on the graphic representation associated with each of the five sentences, we conducted a study among teachers and students in which they were asked for suggestions of representations simple to understand, easy to use, and motivating [48]. The students presented 13 proposals, while the teachers provided a total of 7. Most of the received proposals were related to using roads and traffic signs to represent programming concepts, so the final work was based on such proposals to produce a road-metaphor-based graphic representation assembled as a flowchart to reach the initial objectives. In this way, the metaphor aims to visually show the static structure of the program or algorithm to be represented by a sequence of roads and traffic signs.

Figure 3 shows the set of chosen graphic representations, each of them associated with the corresponding language statements previously identified:

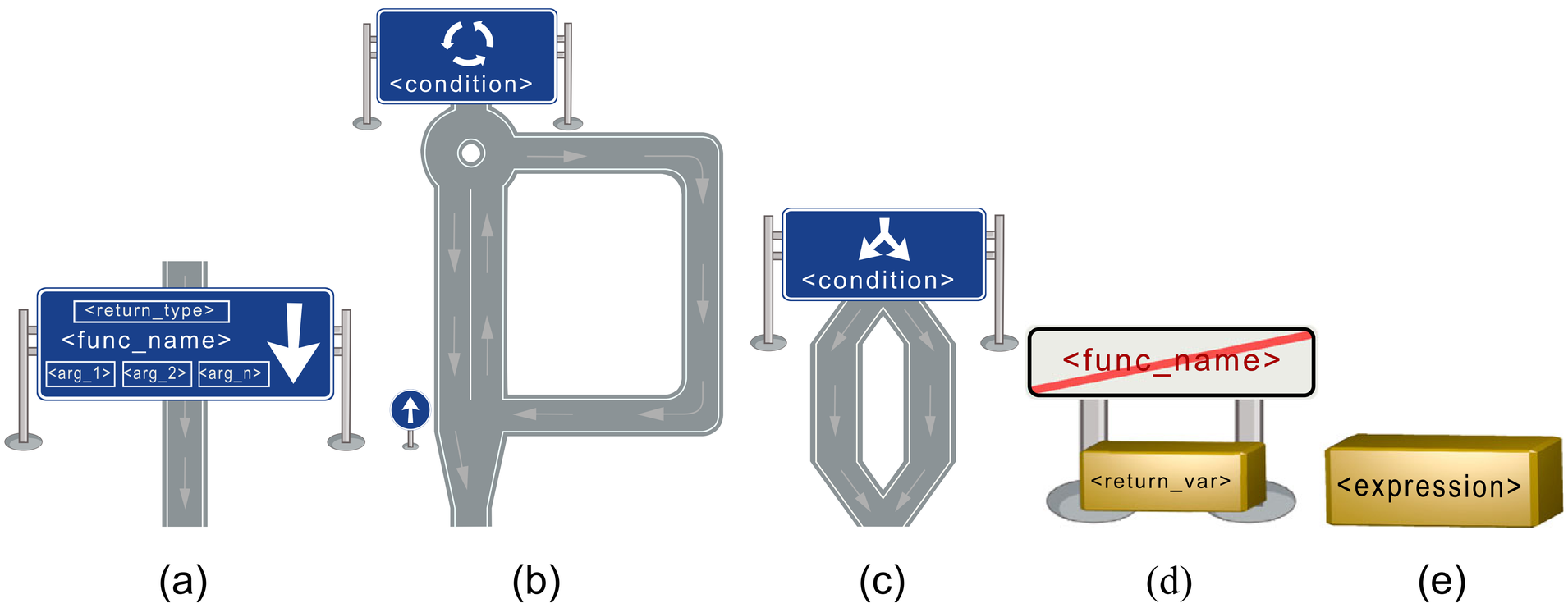


Fig. 3 Set of initial graphic representations: **a** function definition, **b** loop sentence, **c** condition statement, **d** function return, and **e** expression evaluation

- Function definition. Represented in Fig. 3a by a traffic sign and a portion of the road. In this representation, the traffic sign provides information about the function name, its arguments and the type of information that it returns. Every function visualization should start with this graphic representation.
- Loop sentences. Represented in Fig. 3b by a traffic sign, a roundabout and several sections of road. As in condition statements, a traffic sign is used to show the condition that the loop body decides or not to execute. The roundabout represents the key concept of the metaphor; a vehicle can travel indefinitely around the roundabout until it decides to leave at one of its exits. In this case, the eastern exit and the southern entrance of the roundabout would represent the execution of the loop, i.e., loop body, while the southern exit would mean abandoning it. This metaphor allows to stress the importance of defining the loop repetition condition correctly so as to avoid infinite loops, which would be reflected in the fact that the cars would remain in the roundabout indefinitely.
- Condition statements. Represented in Fig. 3c by a fork in two roads next to a traffic sign. The road on the left represents the case where the condition of the sentence is met, while the road on the right would represent the opposite case. The traffic sign is used to show the condition under evaluation to the user. This metaphor reflects the execution of only one of the existing alternatives perfectly.
- Function return. Represented in Fig. 3d by a traffic sign with a box. The traffic sign shows the name of the function which is being abandoned, while the box represents the variable that is being returned.
- Evaluation of expressions. Represented in Fig. 3e by a box. The information shown in the box is the literal transcription of the expression that is being evaluated, such as an assignment or function call. This metaphor is intended to illustrate the concept of the variable as a container of values, which can be given by any correct expression.

Although a left-to-right/top-to-bottom interpretation of the visualization is assumed. All representations include the arrows drawn that also indicate the direction to be followed by the user when they interpret the visualization. A reduced contrast between the arrows and the roads is proposed in order to avoid highlighting the former, so that the user is not distracted by any secondary elements.

With regard to the design of the graphic representations, we have made reference to the road signs proposed in “Vienna Convention on Road Signs and Signals” in 1968 [55],

adopted internationally by most countries. This allows the users to have a basic idea of what the graphic representations may mean, as they are already familiar with the notation.

4 Assessment of the notation proposed

4.1 Objective

The main objective of the evaluation of the notation is to obtain some initial results that allow us to know whether the proposed notation is suitable to represent the semantic meaning of the programs and whether it is really understandable. In other words, the main goal is to evaluate the dimensions of suitability and understanding.

The results obtained will motivate the subsequent lines of work that will be addressed to improve or adapt the notation.

4.2 Method and participants

To evaluate the understanding and suitability of the notation proposed, a blind quasi-experiment was conducted with two groups of students from the Bachelor's Degree in Computer Science: one was a beginner, consisting of students who were beginning to learn programming and were introduced with some programming concepts like expressions, loops, condition statements and functions (n = 63), and the other was more experienced, with knowledge of algorithms, data structures, object-oriented programming and recursion (n = 89). To simplify, the first group will be called from now on the beginner group and the second one will be referred to as the intermediate group. Both groups had no knowledge of programming before enrolling in the CS university degree, since programming does not exist in the elementary and high school curricula in Spain.

Each of the students in the groups had to accomplish two tasks so as to obtain both objective and subjective results to be evaluated later, with a time limit of 50 minutes to attain both tasks.

In these two tasks, the students were given a visualization in the proposed notation which they had to transcribe into pseudocode, into any programming language, or into natural language. These tasks were adapted to the level of knowledge of the evaluation groups: the programs provided to the beginner group were to obtain the number of prime numbers from a list, and the number of perfect numbers from a list; on the other hand, the intermediate group was provided with the representation of the bubble sort algorithm [28], and then, the representation based on the classic problem of the knapsack [26]. Thus, although the tasks posed very specific programming exercises, they should not present any problem for the participants as they are similar to those given during the classes (according to their programming skills).

The objective of making a transcript of the visualization was to check whether the participants understood the semantic meaning of the graphic representations. Thus, the analysis of the results of the transcriptions implied evaluating whether the participants had understood the graphic representations, obviating the fact that the syntax used in the transcriptions was correct.

During the completion of the first task, the students were provided with no assistance that would help them identify the meaning of the graphic representations that composed the visualization. For the second task, a legend with the meaning of each of the graphic representations was provided, indicating the associated programming concept.

```
public static double [] knapsack(double capacity,
                                 double [] weights) {
    double [] result = new double[weights.length];
    double rest = capacity;

    for (int i = 0; i < weights.length; i++) {
        result[i] = 0.0;
    }

    for (int i = 0; i < weights.length; i++) {
        if (weights[i] < rest) {
            result[i] = 1.0;
            rest -= weights[i];
        }
        else {
            result[i] = rest / weights[i];
            rest -= rest / weights[i];
        }
    }

    return result;
}
```

Listing 1 Code implementation of the Knapsack-problem in Java

As a way of example, Listing 1 shows the Java transcription of the graphic representation in Fig. 4 as a solution for that task.

Upon the completion of the exercise, the students in both groups scored each of the graphic representations on a Likert scale of 5 points (1: hard to understand, 5: easy to understand), depending on how easy to understand and how appropriate it was. By "easy to understand" we mean whether the graphic representations can be understood without the need for any kinds of legend or previous training. By "appropriate" we mean whether the proposed notation really represents the concept from which it is intended to abstract.

Finally, the students were given the opportunity to provide some comments with the aim of making future improvements to the prepared notation. The whole procedure used to evaluate the notation proposed is illustrated in Fig. 5.

4.3 Results

The results obtained are shown in Table 1, grouping the graphic representations by dimension evaluated on the basis of the calculated mean, mode, and standard deviation.

On the basis of these data, average values higher than 3 can be observed in all the graphic representations, thus indicating a positive assessment of these, although there is a wider disparity of opinions in the beginner group, which may be related to the nature of the evaluation group itself and its programming skills.

In this table, the values for averages greater than 4 are highlighted in order to show that the participants of the intermediate group generally rated the graphic representations better than the members of the beginner group. When we assess the dimension for understanding expressions, there is a difference of 0.64 between the two groups. This occurs similarly with other graphic representations, for instance, in the magnitude for the suitability

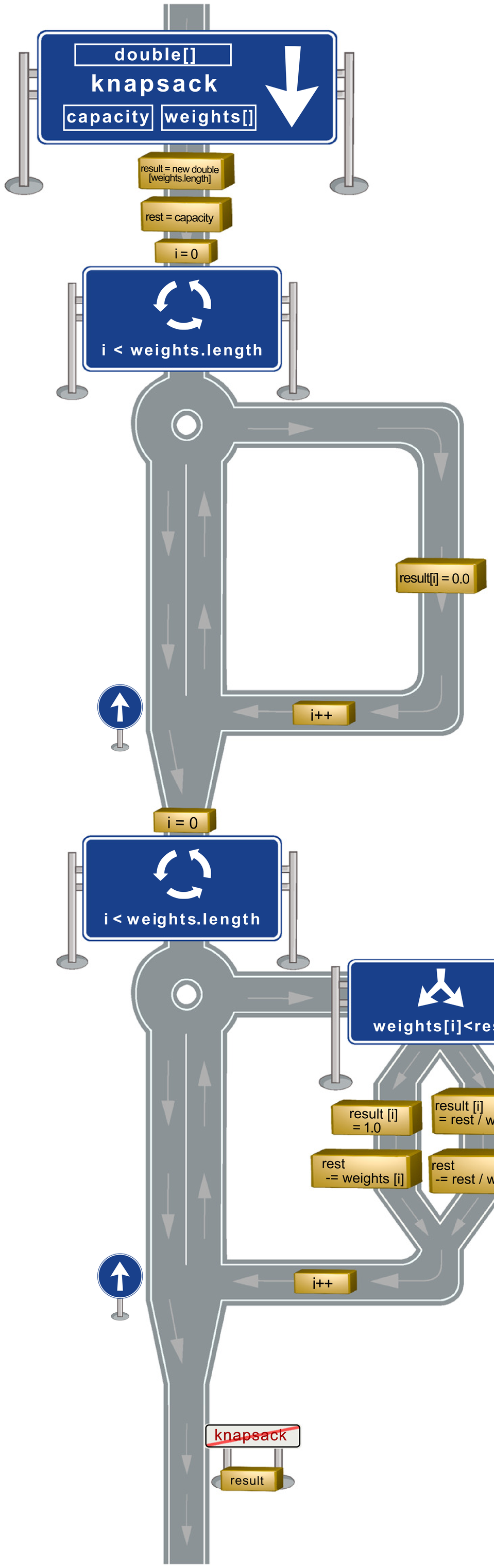


Fig. 4 Road-based graphic representation of the implementation of the Knapsack-problem

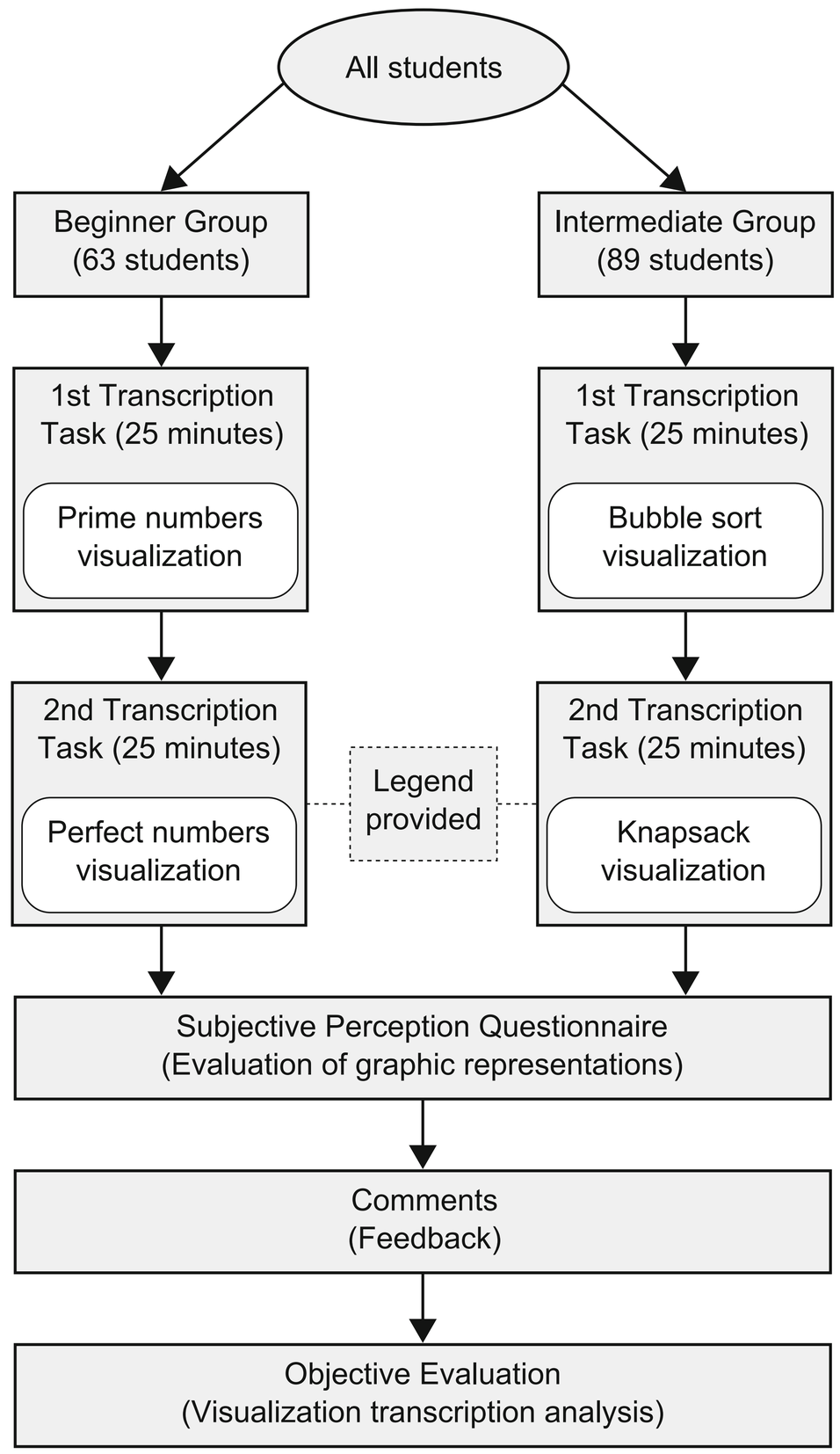


Fig. 5 The phases of the experimental evaluation conducted

of functions (0.55) and the return function statements (0.41). Therefore, it can be observed that the perception of the participants of the beginner group on both dimensions differs significantly with respect to the perception that the intermediate group has on the different graphic representations. This result is probably due to the optimism of the students who start programming, as opposed to those who already have some experience.

Figures 6 and 7 also show the scores of both groups as a percentage, comparing them for each graphic representation in the context of the dimensions assessed. By relating the two evaluation groups in both charts, it can be seen that the loop and function definition sentences are the ones that obtained the lowest maximum scores, with only an average of 35% for both dimensions. This is noted in the subsequent comments provided by the students regarding visualization, where the metaphor of the roundabout to represent loops was referred to.

Figures 8 and 9 show the results obtained after analyzing the visualization transcriptions provided by the students for the first and second tasks, respectively. In the first figure, the "Overall identification" columns mean whether the association made by the students between the graphic representation and the related concept is straightforward. By relating these columns with the results shown in Table 1, it can be seen how the perception that the students had about the understanding of the graphic representations is consistent with the actual solutions that they provided. The other columns in the chart show specific elements which belong to the graphic representations and how easy they were to understand. Comparing both figures, it can be seen that while solving the second task, the results obtained

Table 1 Statistical values relating the "Understanding" and "Suitability" dimensions to the scores provided by both beginner and intermediate groups for the 2-D evaluation (1: hard, 5: easy)

	Understanding					Suitability				
	Function	Condition	Loop	Expression	Return	Function	Condition	Loop	Expression	Return
Beginner										
Mean	3.45	**4.11**	3.83	3.96	3.85	3.45	**4.04**	3.69	3.89	3.86
Mode	4.00	5.00	4.00	5.00	5.00	4.00	5.00	4.00	5.00	5.00
Standard deviation (σ)	1.01	0.95	1.04	1.12	1.05	1.03	0.95	1.08	1.09	1.11
Interm.										
Mean	3.57	**4.14**	3.97	**4.60**	**4.30**	**4.00**	**4 21**	3.98	**4.51**	**4.27**
Mode	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
Standard deviation (σ)	0.94	0.90	1.08	0.63	0.82	0.99	0.97	1.05	0.69	0.84

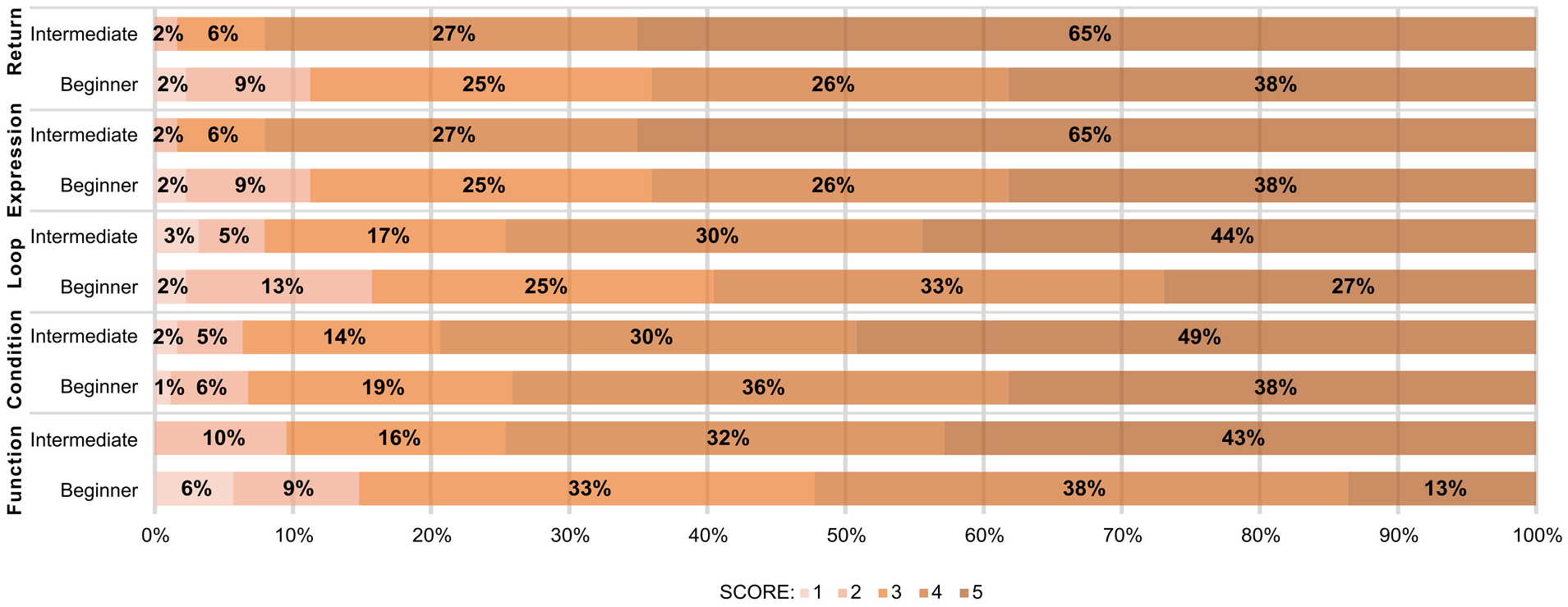


Fig. 6 "Suitability" scores for every graphic representation per evaluation group for the 2-D notation

were better than those obtained from the first task. This is due to the fact that the legend provided to the students detailed how to interpret the graphic representations. Even so, the differences between the results of both evaluation groups are still visible.

The most common errors detected when correcting the transcriptions involved redefining variables passed as arguments to the function, reading the standard input to assign values to variables that were actually already initialized, and misinterpreting the visualization causing the elements to be transcribed in the wrong order.

5 Improvements

5.1 Changes made to the initial notation

From the analysis of the results and the comments made by the groups of students, it can be seen that, in general, the understanding and suitability dimensions of the initial notation are positively assessed by the students.

Nevertheless, in response to the comments made by the evaluation groups, we made an effort to improve the understanding and suitability of the representations, resulting in those illustrated in Fig.10:

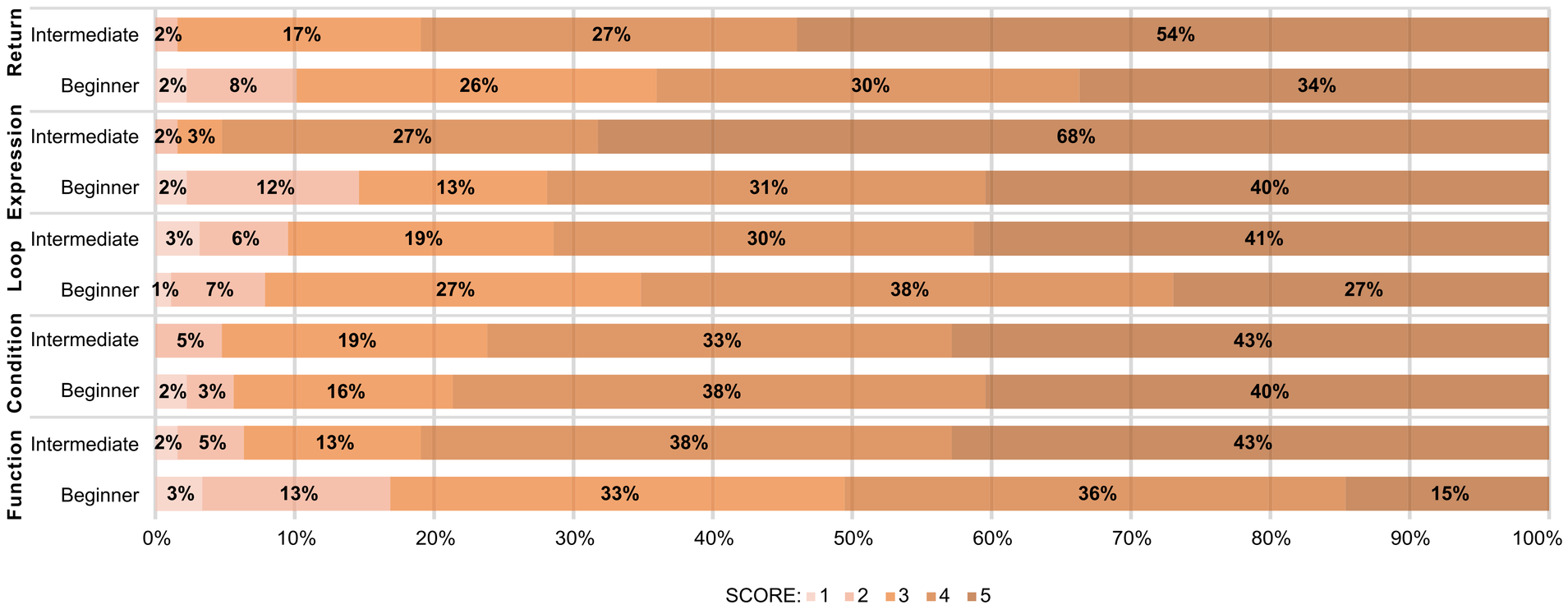


Fig. 7 "Understanding" scores for every graphic representation per evaluation group for the 2-D notation

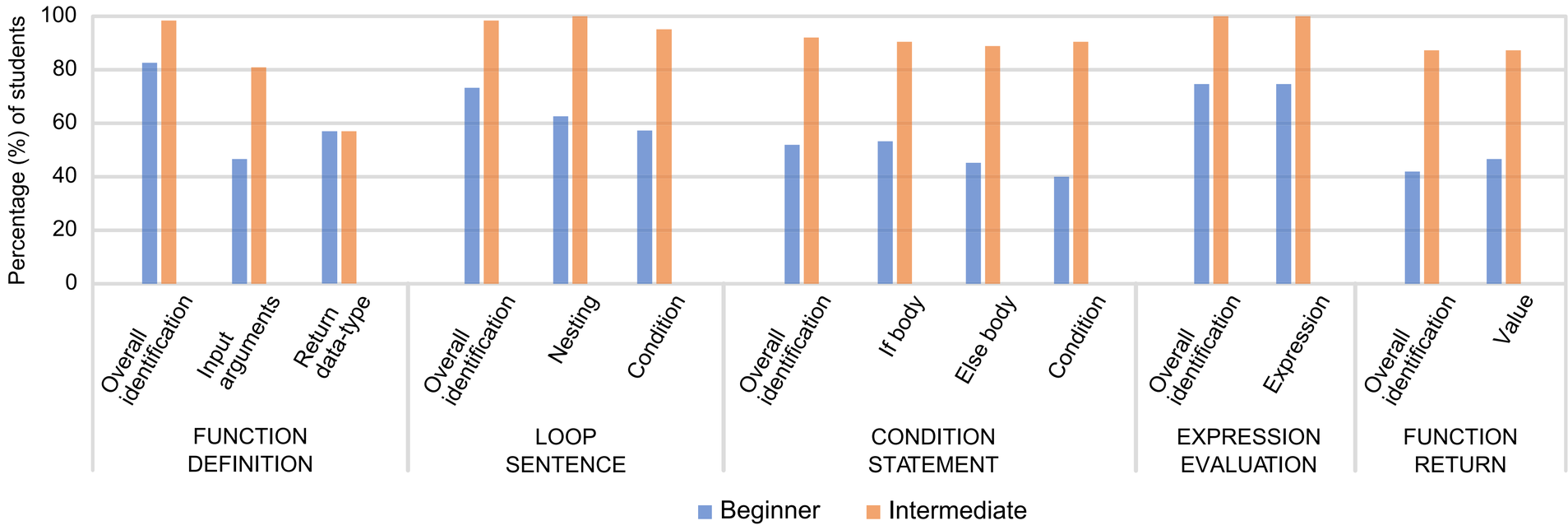


Fig. 8 Evaluation results of the solutions provided by the students when solving the first task. The columns represent the percentage of students who demonstrated an understanding of the criteria defined for each graphic representation

- Function definition (Fig. 10a). We decided to omit the data type returned by the function. In general, and in order to keep the visualization as simple but understandable as possible, we preferred to ignore the data types, as in non-typed programming languages. This also occurs in any other graphic representation where the data types of the variables are referred to.
- Loop sentences (Fig. 10b). The traffic sign has been modified to include arrows and text tags which indicate the direction of interpretation in case the condition of the loop is met or not. For roads, arrows have been included in the roundabout to indicate the direction of interpretation. The mandatory northbound direction sign has also been removed when the loop is iterated again and replaced by a continuous line prohibiting passage over the road.
- Condition statements (Fig. 10c). As with the loop sentences, text labels have been included in the traffic sign to indicate the meaning of the interpretation.
- Function return (Fig. 10d). In order to avoid confusion, the function return value is composed of another traffic sign, which is on top of the end-of-function sign and placed under another sign with the word “return”.

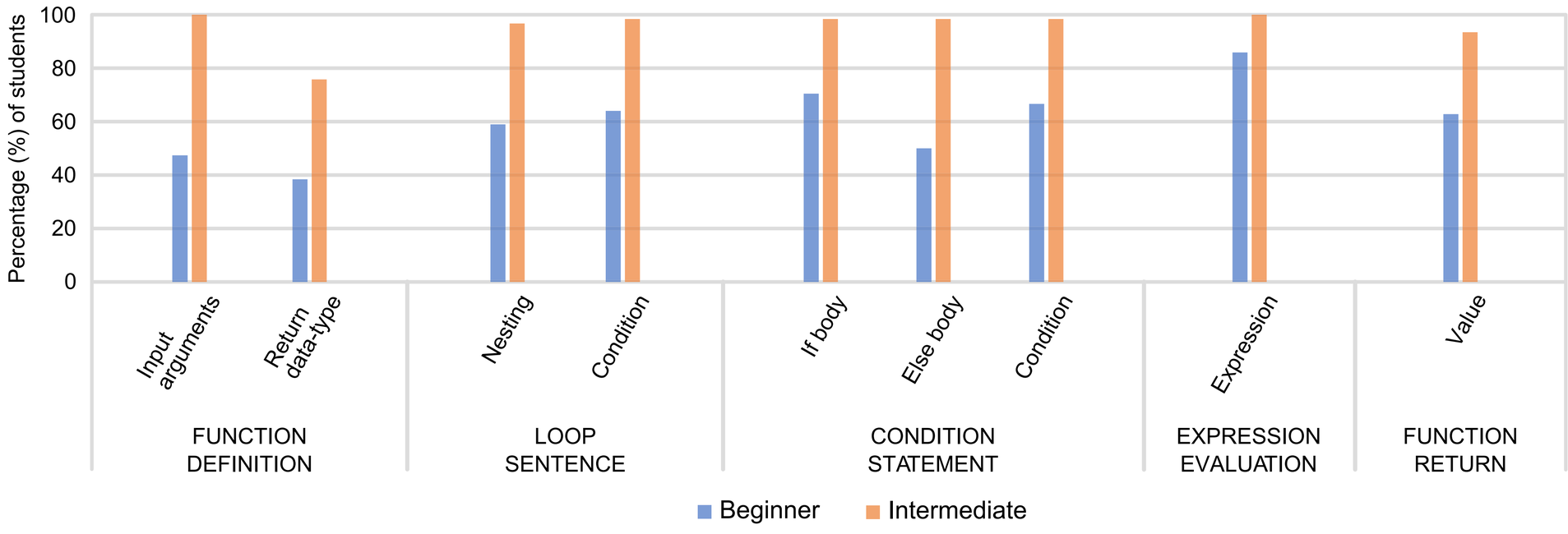
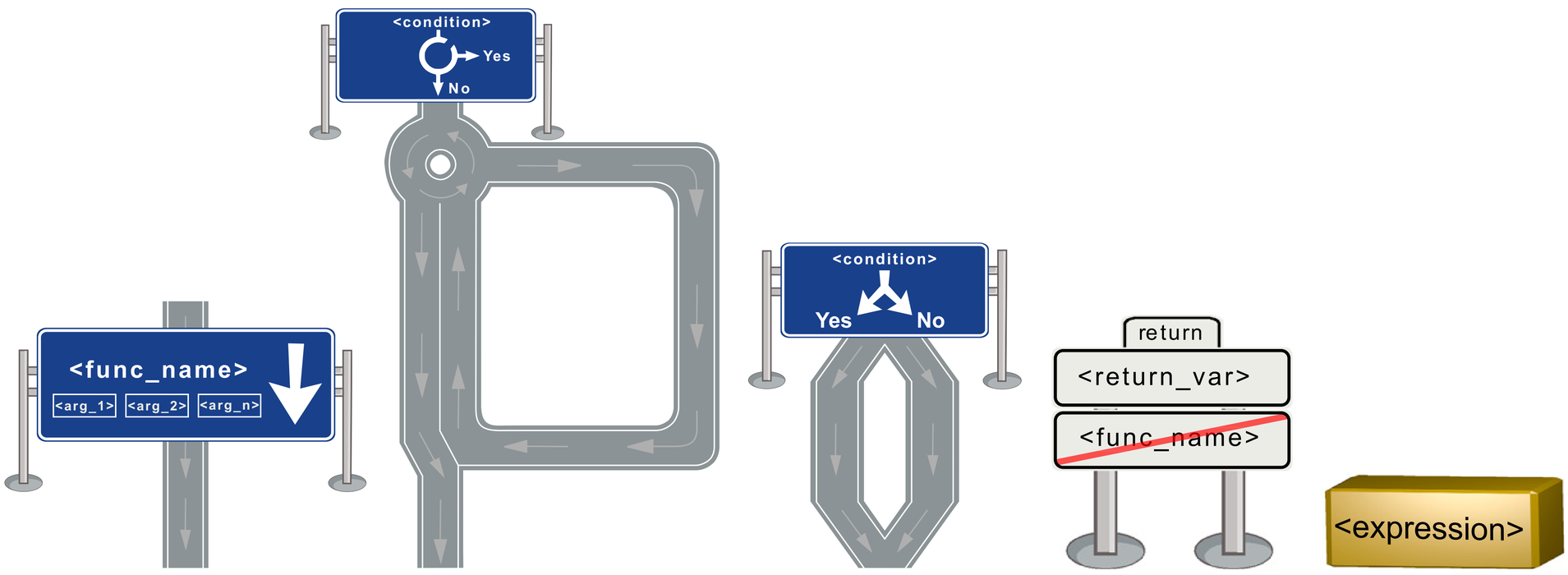


Fig. 9 Evaluation results of the solutions provided by the students when solving the second task. The columns represent the percentage of students who demonstrated an understanding of the criteria defined for each graphic representation. Note that the “Overall identification” column, which exists in Fig. 8, has been removed since a legend detailing the graphic representations was provided during the experiment; thus, the identification of the graphic representation cannot be evaluated in this case



(a) (b) (c) (d) (e)

Fig. 10 Set of improved graphic representations after the group evaluations

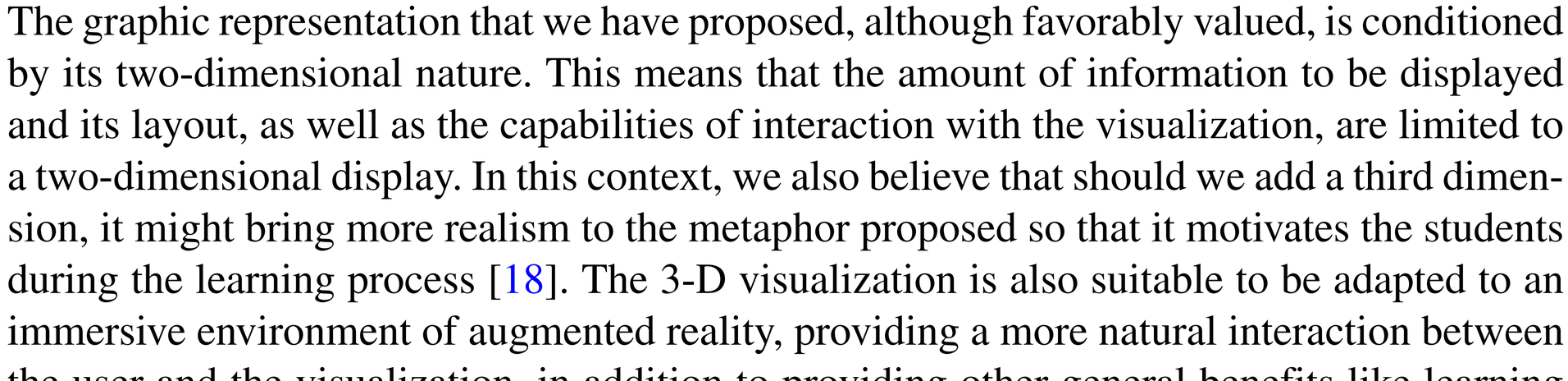
– Evaluation of expressions (Fig. 10e). In this case, the graphic representation in the form of a box is retained.

We believe that these changes bring greater understanding to the graphic representations by explicitly including certain user aids, along with greater simplicity to the interpretation of the visualization as a whole.

5.2 Evolution to a 3-D environment

The graphic representation that we have proposed, although favorably valued, is conditioned by its two-dimensional nature. This means that the amount of information to be displayed and its layout, as well as the capabilities of interaction with the visualization, are limited to a two-dimensional display. In this context, we also believe that should we add a third dimension, it might bring more realism to the metaphor proposed so that it motivates the students during the learning process [18]. The 3-D visualization is also suitable to be adapted to an immersive environment of augmented reality, providing a more natural interaction between the user and the visualization, in addition to providing other general benefits like learning gains and improved attention and enjoyment, among others [1].

Therefore, a 3-D graphic representation is presented, based on the modified designs of the two-dimensional representations previously discussed. From there, the creative design decisions regarding the color, size, and shape of each one have been maintained, as well as those related to the notation based on the road metaphor itself.



(a) (b) (c) (d) (e)

Fig. 11 New 3-D graphic representations for the **a** function definition, **b** loop sentence, **c** condition statement, **d** function return and **e** expression evaluation

Figure 11 shows the new set of graphic representations transferred to a three-dimensional environment in an orthographic projection. Each of them is described individually as follows:

- Function definition (Fig. 11a). The 3-D conversion has been done directly, keeping the traffic sign where the name of the function and up to four arguments are shown. The arrow of the traffic sign includes the reading direction of the visualization.
- Loop sentences (Fig. 11b). The main part of the graphic representation is the roundabout itself, isolated as a single piece so as to facilitate the automatic construction of the visualization. The south exit of the roundabout is twice as wide as the east one in order that it can allow flow in both directions, simulating repeated execution of the code fragment. The traffic sign shows the condition to enter the loop and the direction indicators for each exit.
- Condition statements (Fig. 11c). As before, the main part, which would be where the condition is evaluated, has been isolated in order to simplify the automatic construction by pieces. In this case, the traffic sign shows the condition and some arrows which indicate the direction to take depending on whether the condition is met or not.
- Function return (Fig. 11d). In this case, the traffic sign is supplemented by a fragment of road that facilitates the connection with other elements. The information shown by the traffic sign is maintained in the same way as in the 2-D representation, that is, the variable that is returned and the name of the function to which the sentence belongs.
- Evaluation of expressions (Fig. 11e). As in the previous case, a fragment of road on which the box rests has been added. Furthermore, the box can be opened from above to allow advanced interaction mechanisms, which will be discussed later. The related expression is shown on the side of the box, as in the 2-D representation.

From this set of 3-D graphic representations, complete static visualizations can be constructed, which serve to understand the structure of a program or algorithm.

Figure 12 shows the complete graphic representation of the bubble sort algorithm. The visualization would provide information about the algorithm structure, based on two nested loops and a condition statement that only has logic associated with one of its branches. In this visualization the auxiliary connectors necessary to build the visualization are also observed so that all of its elements are linked correctly.

Thus, the interpretation of the visualization would go through a reading from top to bottom, starting with the definition of the function and the entry to the external loop. This loop uses a counter variable, called *i*, which is created in the visualization by means of a

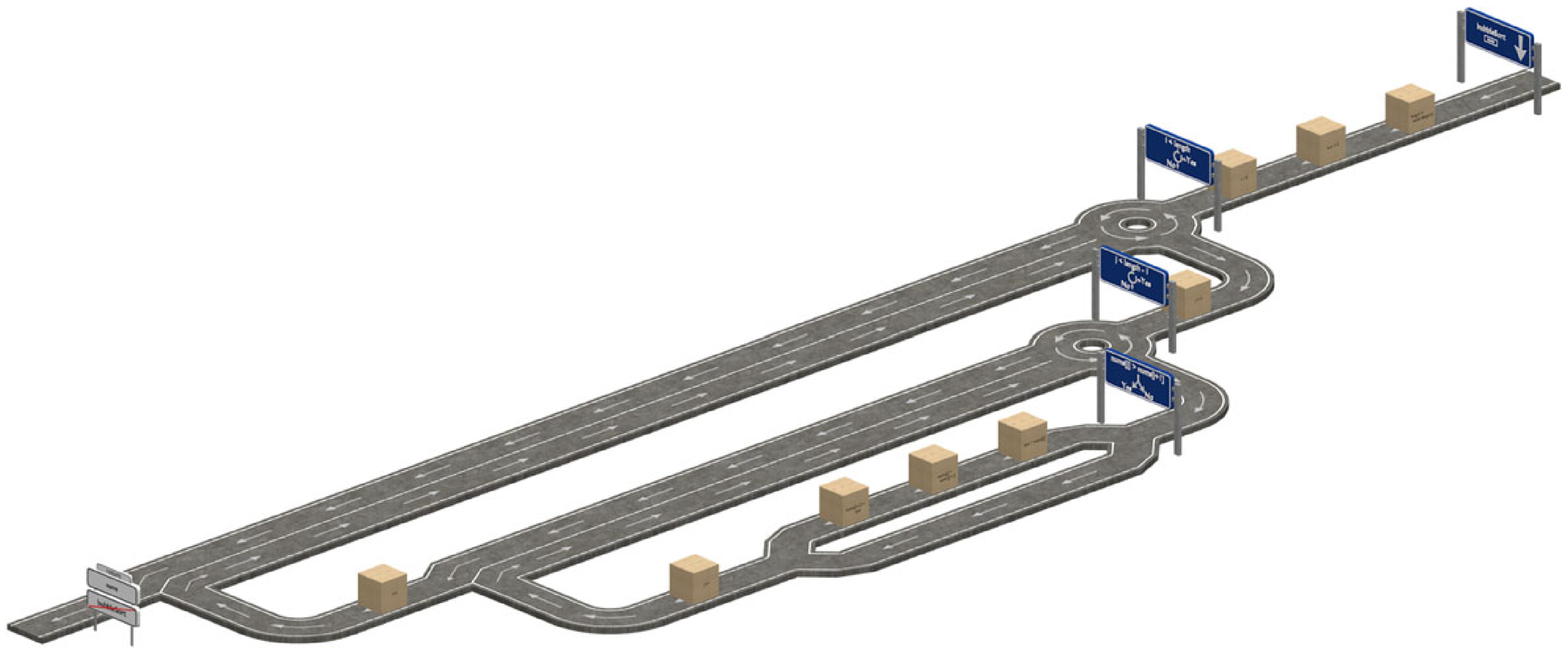


Fig. 12 Bubble sort algorithm visualized as a 3D road-based graphic representation

graphic representation of the type of expression evaluation, before entering the loop condition. Assuming that the condition is met, the reading would enter the inner loop. As before, a new counter variable is used, called *j*, previously defined by means of an expression evaluation representation. Assuming again that the loop condition is met, the display interpretation would now pass to the condition statement, where all the related expressions would be interpreted, were the condition to be met (left bifurcation). Finally, the counter variable *j* of the inner loop would be increased. After this first iteration, the reading would return to the roundabout, where the loop condition would be evaluated again. Provided this were true, the interpretation would be directed again along the east lane of the roundabout, whereas otherwise, it would be directed along the south lane. In the latter case, the counter variable *i* of the outer loop would be increased and the condition of the outer loop would be re-evaluated. If the condition is not met, the loop would be abandoned by the south exit of the roundabout and the reading would pass to the graphic representation of the function return, where the visualization would end.

6 Improving COLLECE-2.0 to support the visualization of programs

As discussed in Section 1, COLLECE-2.0 is designed as a collaborative programming learning environment through problem solving. Among the main features, the environment provides real-time remote editing capabilities of projects, awareness mechanisms such as tele-pointers, blocking of regions in the source code, communication between users, and session management based on problems to be solved, among others [49].

This environment is designed as a scalable system based on the Plugin Development Environment (PDE) technology of Eclipse, which allows its extension with new functionalities that improve the programming learning process. Thus, the aim is to extend the functionality offered by integrating the 3-D graphic representations proposed in this work to visualize complete programs using augmented reality.

The implementation of this environment has been done as an Eclipse plugin that provides communication and synchronization features through a centralized network architecture, in which a server manages the sessions to which clients are connected. This network architecture is based on the use of Eclipse Communication Framework (ECF) channels, a communication middleware that abstracts the developer from the implementation of low-level network mechanisms. Hence, these communication channels are used to i) implement an API that clients can consume so to execute certain operations on the server, ii) synchronize changes made to the files of source code by multiple users in real time, iii) prevent consistency errors between the server and clients, iv) enable communication between clients via text chat and v) notify the clients who are connected to a session of certain events.

Figure 13 shows the main modules of the platform in which it can be observed the functionality provided and the architecture implemented on which it is based [50]. Of these modules, we can highlight those which conform the user interface, built from the SWT toolkit offered by Eclipse for the construction of interfaces; the awareness mechanisms that support the main functionality of the system; the synchronization of projects, performed at a high level through the use of Git repositories and at a low level during the concurrent edition and in real time through the implementation of the Jupiter algorithm [42]; the network communication based on the ECF framework as mentioned above; and those that provide

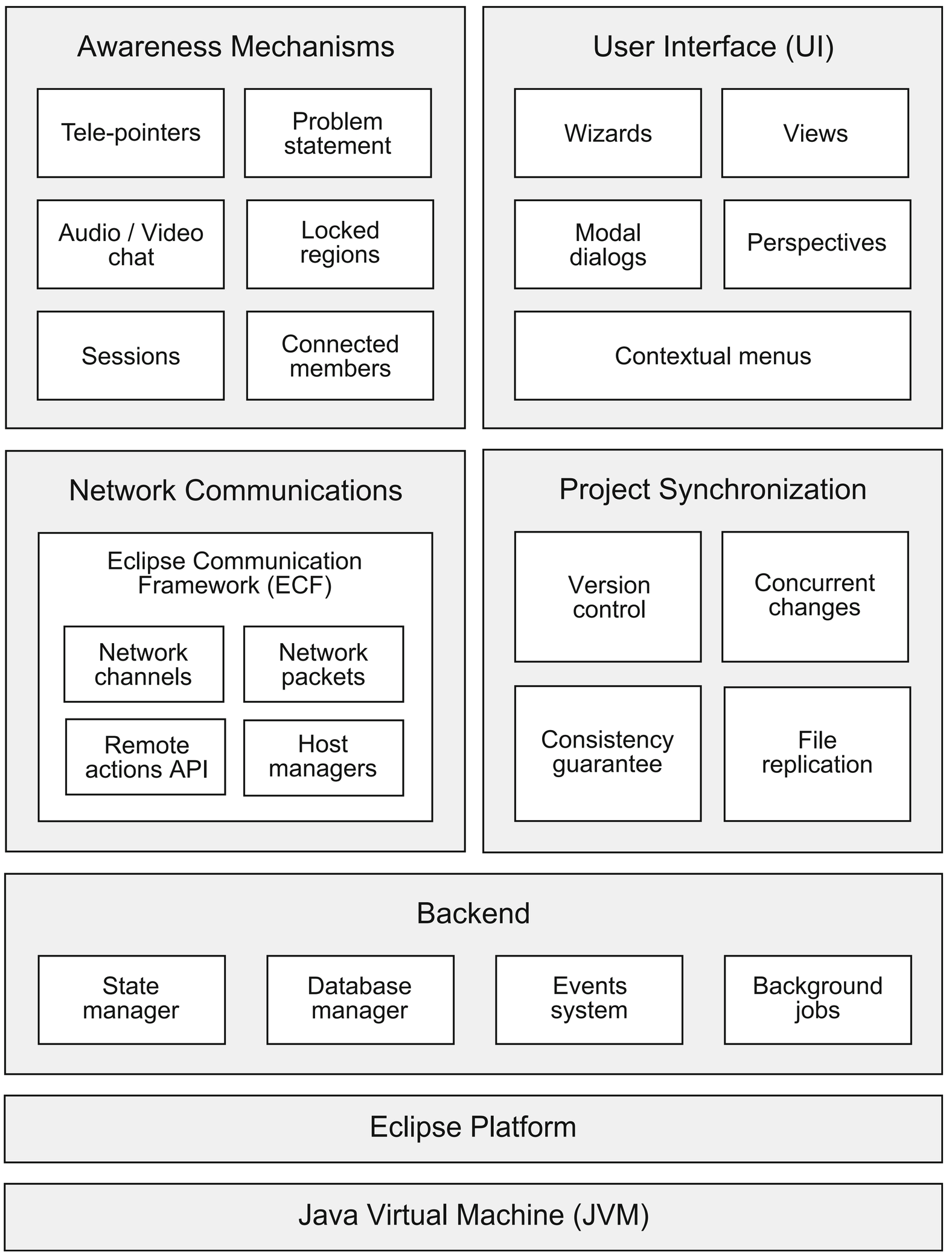


Fig. 13 Modules of the COLLECE-2.0 platform defining the architecture and the features provided

low level support to the system, such as persistence management through databases, context or application state management, etc.

In order to provide mechanisms that enable the expansion of functionality of the platform, COLLECE-2.0 implements an event system that is triggered when certain actions occur. In consequence, other plugins, which subscribe to these events and execute certain actions accordingly, can be implemented in a decoupled way. Some of these events are triggered when a user signs up for a session, joins it or opens a source code file in the IDE, among others. Thanks to this, the implementation of the program visualization system has been done by creating a new plugin for Eclipse that depends on COLLECE-2.0.

Plus, the visualization can be displayed through an augmented reality device. Among the current devices, Microsoft HoloLens excels thanks to the immersion and interaction capabilities that it offers, which is the reason why this device has been selected as a visualization

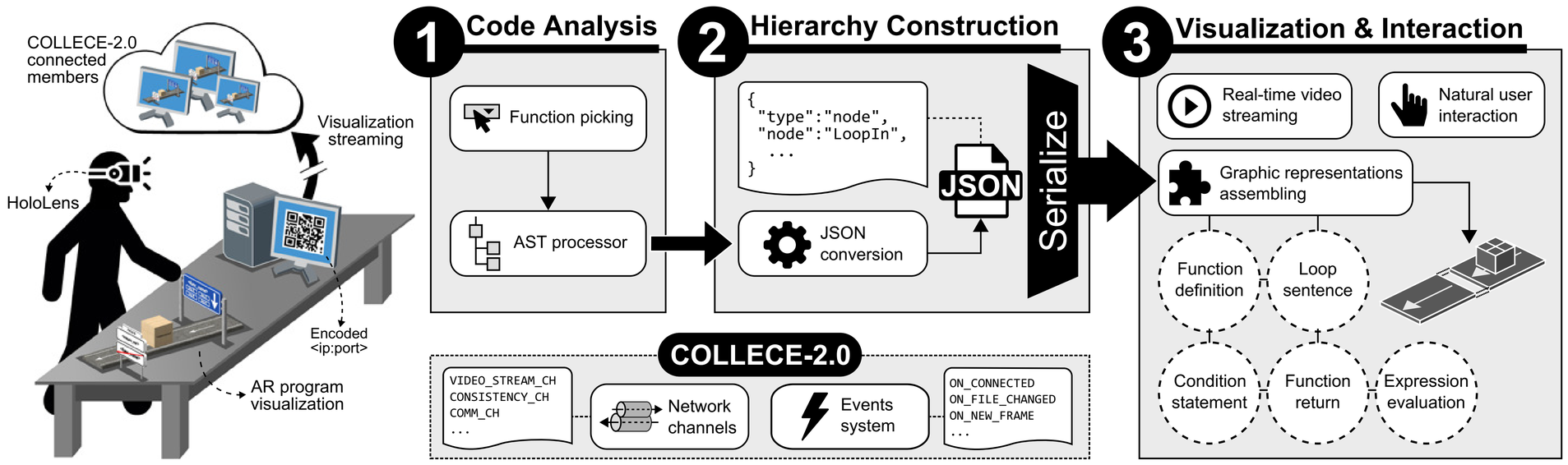


Fig. 14 General overview of the execution flow to generate a visualization from a function selection

medium. Nonetheless, the developed implementation is independent of the device chosen, thus it can be easily replaced by another.

6.1 Visualization Plugin

In order to support visualization of programs, the Eclipse plugin infrastructure has been used to build a plugin that leverages the features provided by COLLECE-2.0. This new plugin contributes to COLLECE-2.0 with functionalities to analyze the source code of a program, manage the network connection with the augmented reality device and broadcast the visualization to the rest of the clients connected to the session in real time, provided that they do not have the visualization device physically. Figure 14 shows a diagram which represents the workflow used to build the visualization of a program from its source code. Similarly, Fig. 15 shows the transformation of the information that happens since the source code is obtained until the visualization is built.

From the source code of the program, a simplified hierarchy is generated in JSON format that can be easily processed by the augmented reality device. This hierarchy contains all the information necessary to construct the visualization. To do this, the plugin analyzes the Abstract Syntax Tree (AST) generated by Eclipse from the source code and identifies each of the language elements related to the proposed 3-D graphic representations (see Fig. 11). The hierarchy generated in JSON format keeps each of these associations in a tree-like form, along with the connectors needed to construct the visualization in a linked way. An example of the sub-hierarchies generated for each of the proposed graphic representations is shown in Fig. 16. The final visualization would comprise the complete hierarchy from these sub-hierarchies.

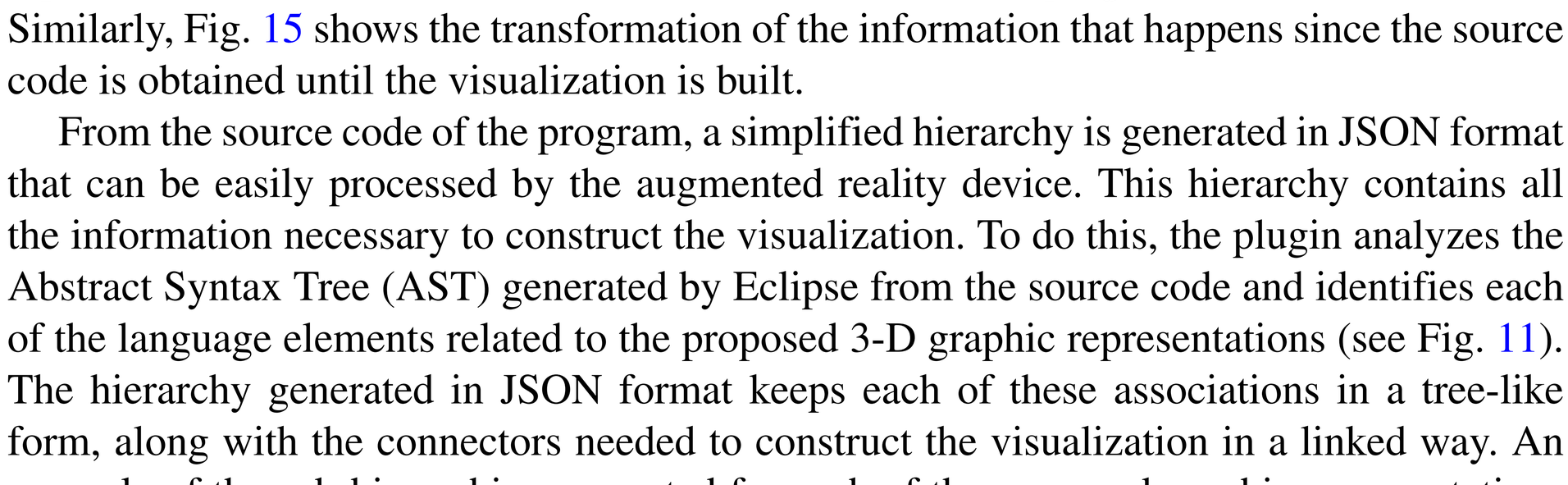


Fig. 15 Data conversions occurring until the final graphic representation is obtained


```
{
  "node": "FunctionIn",
  "name": "bubbleSort",
  "args": [ "nums" ],
  "children": [ ... ]
}
```

(a)

```
{
  "node": "LoopIn",
  "condition":
    "i < nums.length()",
  "children": [ ... ]
}
```

(b)

```
{
  "node": "IfIn",
  "condition":
    "nums[j] > nums[j+1]",
  "children_true": [ ... ],
  "children_false": [ ... ]
}
```

(c)

```
{
  "node": "FunctionOut",
  "name": "bubbleSort",
  "return": "nums"
}
```

(d)

```
{
  "node": "Expression",
  "content": "i = 0"
}
```

(e)

Fig. 16 JSON sub-hierarchies along with their related 3-D graphic representations for the **a** function definition, **b** loop sentence, **c** condition statement, **d** function return and **e** expression evaluation. The green labels define the related identifier of the graphic representation to be rendered when displaying the full visualization

Once the complete hierarchy is generated in JSON format, it is sent to the augmented reality device for visualization. To do this, a client/server architecture is followed through TCP network sockets and an API that is consumed through requests in JSON format which identify the type of request and the request itself. Hence, a field is added to the hierarchy to indicate the device that this request is identified as a "visualization request" (i.e., { `"type": "visualization"` }) and then, it is sent as a JSON message through this socket. This message is received by the augmented reality device for interpretation; first the type of message is identified, in this case visualization, and then the hierarchy is processed to render each one of the identified graphic representations of the visualization.

To facilitate the prior connection between the device and the server, the plugin displays a panel that generates a QR code with the necessary encrypted network information (i.e., IP address and listening port). This QR code is then scanned using the camera of the device to automatically establish the TCP socket connection.

In addition, the plugin also offers the ability to share the visualization with other users connected to the session, in case they do not have an augmented reality device. So as to do this, ECF channels are used to send the frames received from the device already compressed in JPEG format to the COLLECE-2.0 server, which relays them to the rest of the clients of the session leveraging those same channels. The frames are then drawn continuously in the client instances and in a new view provided by the visualization plugin. At any time, the users of the system can stop receiving this stream on demand.

6.2 Automatic generation of the 3-D visualization

The 3-D visualization is automatically built into the augmented reality device by analyzing the hierarchy received in JSON format and generated by the visualization plugin.

For this purpose, the process responsible for automatic generation has been implemented as an application developed in Unity for Universal Windows Platform (UWP), which is suitable for the Microsoft HoloLens device. This application receives the hierarchy in JSON format through the network socket and processes it to build the visualization considering the 3-D graphic representations of the notation.

Each of the 3-D representations designed has the same dimensions as the rest, so as to facilitate the connection between them. In this way, each piece has concretely a length of 130 units, being the meaning of the chosen measurement unit directly related to the display device used; the width and height of the graphic representations are not required as the

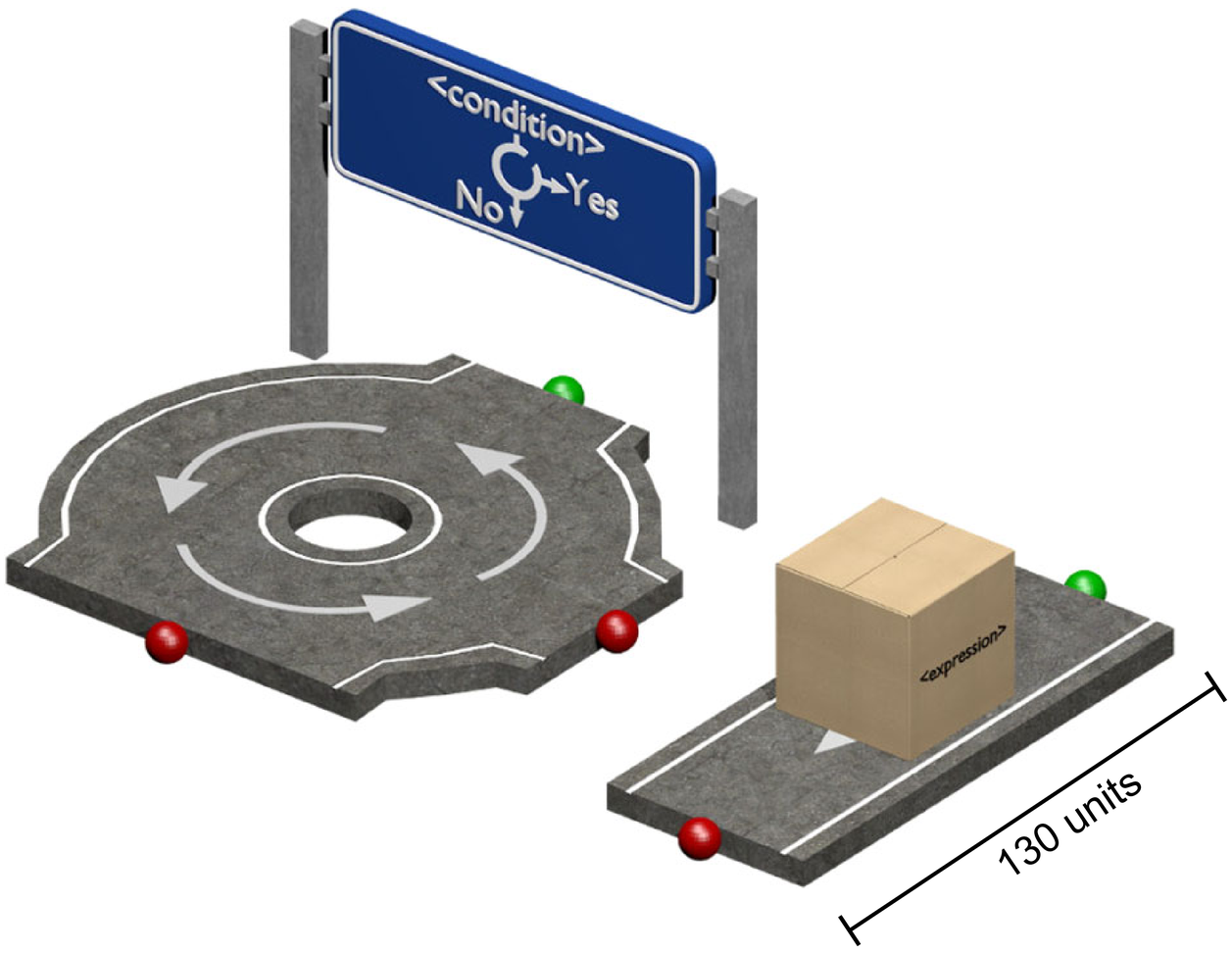


Fig. 17 Sample 3-D graphic representation parts of a loop and an expression with their connection joints and their length

display is built without taking them into account. Likewise, each piece defines its connection points in the design stage, in order to construct the 3-D visualization in a scalable way, should one of the proposed graphic representations be improved or replaced by others in the future. In Fig. 17, it can be observed both the measurements of an example of graphic representation and the connection points that it defines; the green joints represent the input connectors, i.e. those which enable other parts to be coupled before the graphic representation; on the other hand, the red joints, or output connector, enable other parts to be coupled after the graphic representation.

Likewise, there are some graphic representations that include other special connectors, used during animations. For instance, this is the case of the expressions represented by a box.

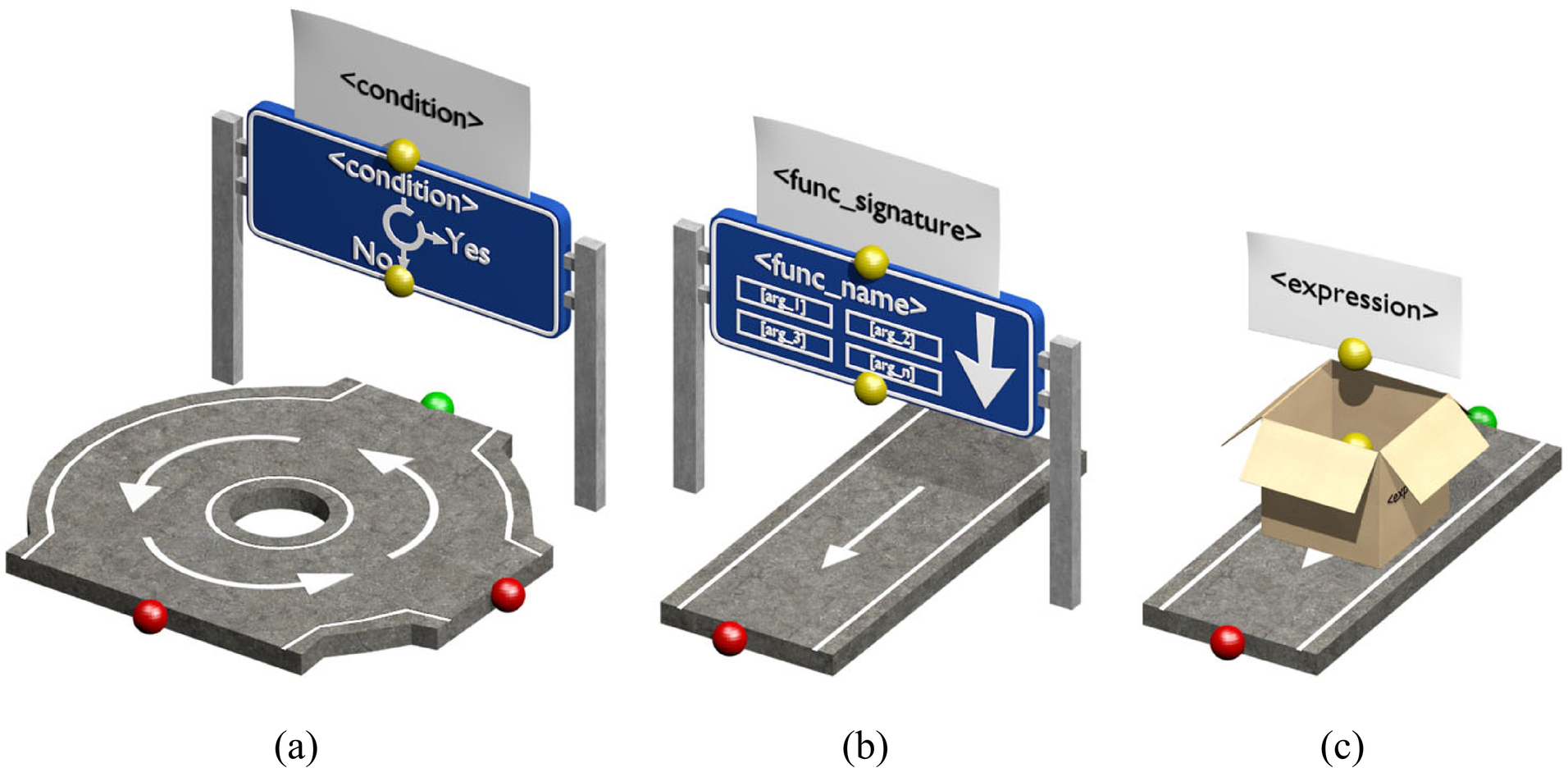


Fig. 18 Graphic representation for the **a** loop statement, **b** function definition and **c** expression evaluation. The yellow joints define the starting and ending position of the informative panels during the animation

This box can be opened when the user interacts with it, in such a way that an informative panel appears with the complete and extended information contained in the expression. The appearance of the panel is animated, and is indicated by special beginning and end connectors, which define the transformation, both scaling and translation, that the panel performs during the animation. Figure 18 shows some of the representations that contain informative panels next to these connectors.

It is interesting to highlight Fig. 18b and c, since due to the small amount of space available, we decided to show information on the data types in pseudocode notation on the informative panels. Although originally the improvement of the notation does not consider displaying data types in graphic representations, this way of displaying such information would not conflict with initial interests by remaining hidden until the user interacts with the representation, thus resulting in additional secondary information that is not visible at first sight and which does not distract the student from the purpose of the visualization. Furthermore, in the case of function definitions (see Fig. 18b), it would also show the rest of arguments which could not be included in the sign itself.

This informative panel shows the content of the expression that should appear on the side of the box, but that for reasons of space, cannot be fully displayed. This way, the panel is automatically scaled to be able to contain the text of the entire expression.

6.3 Software metrics visualization

Given the nature of the static visualization, the system can leverage previous static analysis to provide information about software metrics that help students in their learning process. One of those metrics is the one called "cyclomatic complexity", proposed by McCabe [34] and used to measure the complexity of a program quantitatively. To do this, the cyclomatic complexity (CC) of a program with a single entry point and a single exit point can be calculated in a simplified way, according to [19], as $CC = NB + 1$, where $NB = number\ of\ branches\ in\ a\ program$.

In this context, the system automatically calculates this cyclomatic complexity for the method to be represented, counting the total number of occurrences of the following keywords: *if*, *for*, *while*, *case*, *return*, *&&*, || and *?* (for the conditional ternary operator). Once this value has been calculated, the result is shown on the display by changing the color to red for those graphic representations that increase the cyclomatic complexity of the method over the recommended value: 10. This value has been selected from that proposed by McCabe in his work, identified as a recommended limit and interpreted as the program being "simple and easy to understand".

Figure 19 shows an example of a graphic representation colored in red because cyclomatic complexity has increased over the recommended value, assuming that the display belongs to a larger method. In this way, students can reconsider their solution and modify it to reduce its complexity.

6.4 Augmented reality support

As previously stated, the visualization is shown through the Microsoft HoloLens augmented reality device. The device supports recognition of gestures and voice commands, therefore these features have been used to include natural interaction mechanisms in the platform.

Through gesture recognition, the user can interact with the graphic representations to expand information about the program through the informative panels. We have also

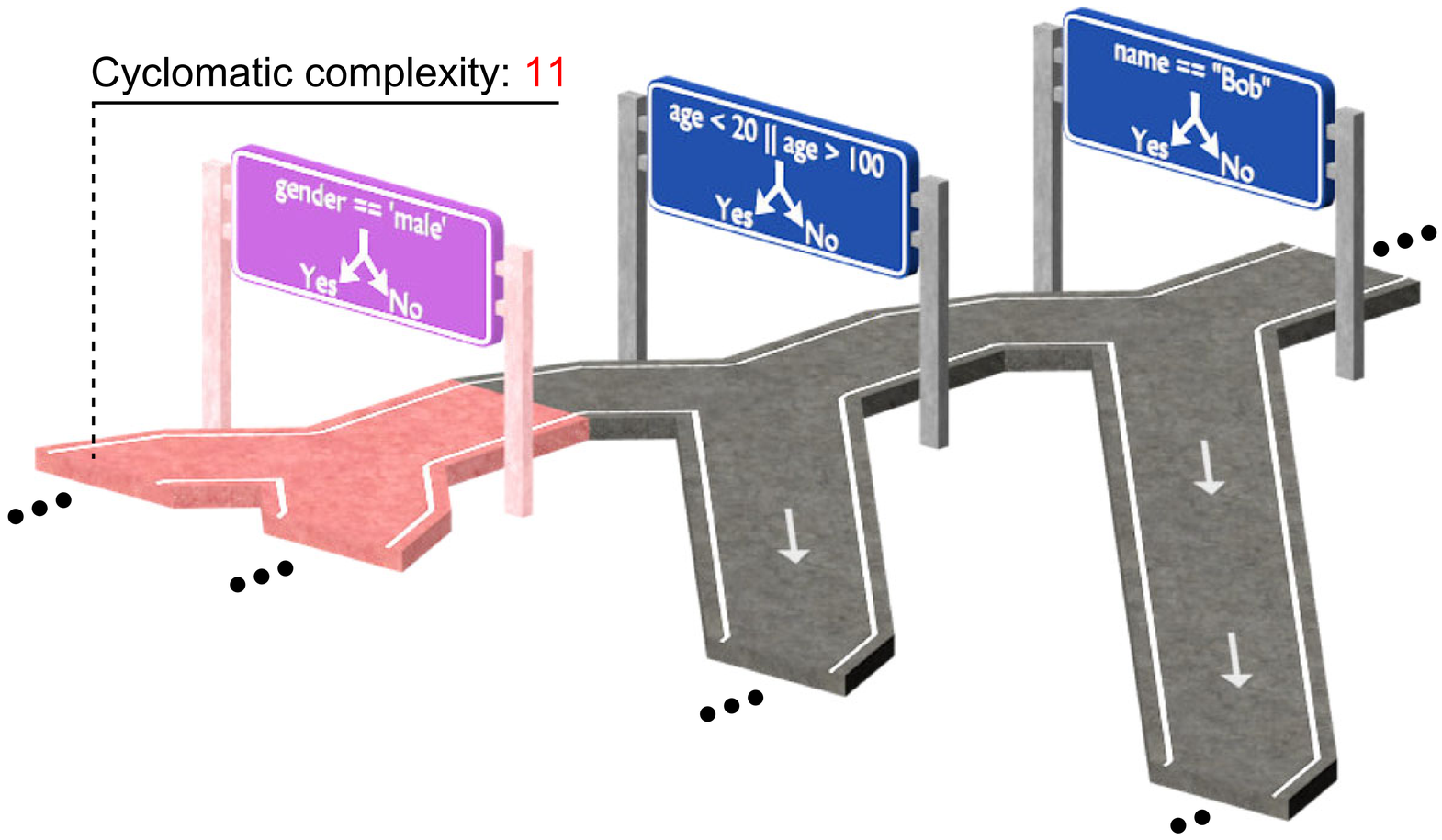


Fig. 19 Cyclomatic complexity shown as a red shading which highlight the graphic representation related to the statement that increases the complexity over the recommended limit

included support so to transform the graphic representations, using handles that allow the user to rotate and scale them, as well as moving them and placing them anywhere else in the space. For the latter, we have made use of the recognition of physical surfaces offered by the device, in order to allow the graphic representations to be placed on real-world objects.

On the other hand, it has been added support for voice commands so as to facilitate the interaction with the visualization. One of these commands would attend to the sentence "face visualization", which once recognized, would rotate the visualization over itself so that it is oriented towards the direction in which the user faces. This is useful when the user has rotated the visualization and wants to restore its initial orientation to be able to easily read the content of the graphic representations. Another of the commands recognized by the system would be the one that reacts to the sentence "remove visualizations", to erase all the visualizations loaded by the user.

Other voice commands have also been included to display certain technical aspects of the visualization. One of these commands is in charge of showing and hiding the shadow projection planes ("toggle shadows"), invisible to the user, however, for reasons of implementation, they are shown during the recordings and captures of images. Another of these commands allows to show the triangle mesh constructed during the reconstruction of surfaces made by the device ("toggle surfaces"), in order to detect possible errors when recognizing the physical environment. The execution of these commands can be seen in Fig. 20.

Finally, Fig. 21 shows, through the augmented reality device, the example of the simulated visualization in Section 5.2, where the bubble sort algorithm was shown. For reasons of performance, the quality of the display decreases when we access the RGB camera of the device, either to capture images or to record video, hence the difference with the original rendering.

6.5 Use case

The new 3-D notation has been tested by beginner students through its use in a scenario similar to the real-world. This scenario was formed by a teacher who generated the visualization

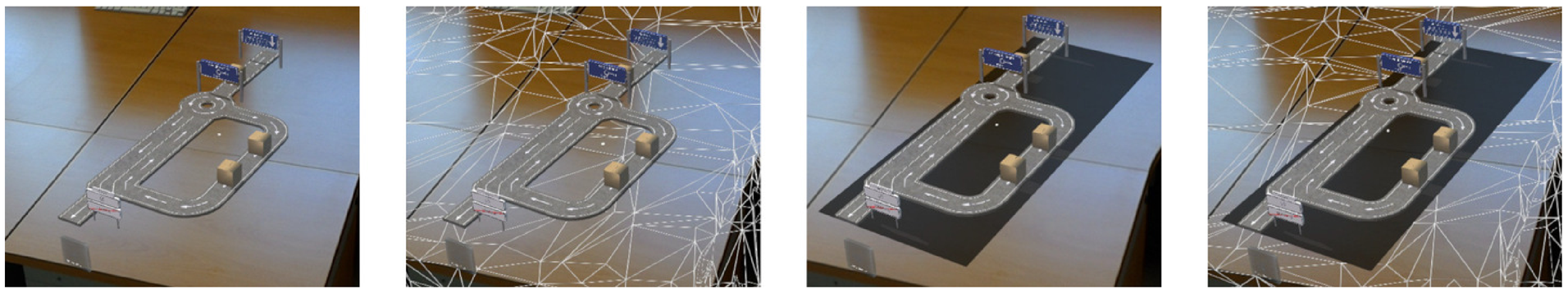


Fig. 20 Appearance after the execution of voice commands to toggle the visualization of shadows and surfaces; **a** invisible shadows and surfaces, **b** invisible shadows and visible surfaces, **c** visible shadows and invisible surfaces and **d** visible shadows and visible surfaces

on the augmented reality device using the COLLECE-2.0 visualization plugin. Once generated, students connected to the COLLECE-2.0 session to watch a live stream of the device. Thus, the teacher explained the structure of the program represented on the visualization itself, explaining the existing programming concepts in the program. In this way, the visualization of several programs was generated to explain programming concepts gradually increasing the complexity of the initial program by means of introducing variables, conditional sentences and loops.

After explaining the basic concepts of programming, each student performed a source code transcription of the bubble sort algorithm into Java or pseudocode from a visualization displayed through the augmented reality device; the algorithm was unknown to them. In addition, the students were provided with a user guide which explained the interaction capabilities of the device with the visualization.

The students performed the transcripts while they were exploring the visualizations. At the same time, they emphasized how easy it was to approach the visualization or move away

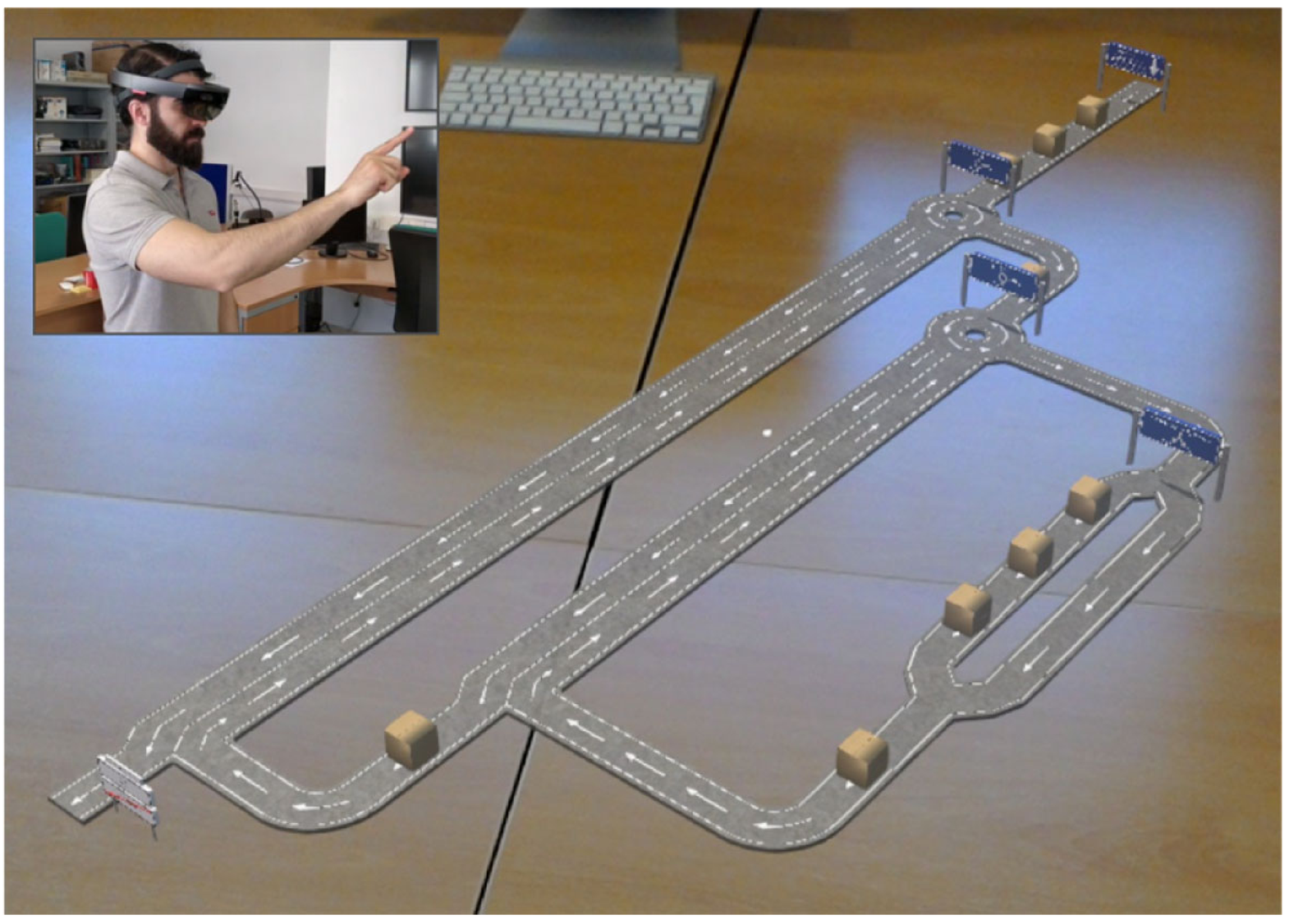


Fig. 21 Bubble sort algorithm automatically generated from the source code, as seen through the augmented reality device

to have a complete view. When they were unable to see an element due to its size, they interacted with it to get an enlarged view (e.g. by opening boxes or displaying informative panels). The rest of the students waiting to take the test could watch in real time a video stream of what the user of the augmented reality device was watching using the COLLECE-2.0 visualization plugin.

At the end of the activity, students shared several comments that were categorized into strengths, weaknesses and proposals for change. These comments are summarized below, with the common ones being rephrased and grouped together:

- Strengths:
  - "You can quickly understand the flow/trace of program execution".
  - "Good similarity to roundabouts and bifurcations".
  - "Once you become familiar with the language it is very easy to understand. It can be very practical when teaching programming".
  - "Easy to understand the condition loops and parameter input and output".
  - "The representation of loops with roundabouts and conditions with the diversions is easy to understand".
- Weaknesses:
  - "The boxes do not fit well with the metaphor of the roads".
  - "I needed the legend. In my case, even though I understood every graphic representation, I wasn't sure if the elements were what I thought they were, since I had not seen them before".
  - "I found it a little strange to see each of the assignments on the sides of the boxes and having to open them step by step".
  - "Direction of reading; you must realize that the exit of the town is the end of the algorithm".
- Proposals for change:
  - "I would include data types in the representation of expressions and improve the degree of expressiveness of the function definition (especially in the type of function return)".
  - "It would be nice to have some way to open all the boxes at once or to be able to see the full assignments at once".
  - "Represent the different operations with different elements such as declaration, initialization, function invocation, etc.".

The provided feedback about weaknesses focuses mainly on changing the graphic representations for boxes, as they are hard to understand. Moreover, several students agreed that the direction of the roads seemed confusing to them, since the traffic signs could be seen from a direction contrary to the movement of the supposed vehicle. All these problems, along with the proposals for change, are being addressed in newer versions of the graphic representations.

For now, these results may be considered as preliminary so as to the reader has a first impression of the system in action. Also, the overall impressions were positive enough to motivate a more formal evaluation in the future.

7 Discussion and limitations

The results obtained from the evaluation of the 2-D notation have motivated certain changes with the aim of improving the notation that have resulted in simplifying the graphic representations, migrating them to a 3-D environment and visualizing them through augmented reality. However, there are certain limitations related to the notation, the evaluation and the generated tools that should be discussed:

- The proposed notation is scalable enough to represent any type of program. However, there are certain limitations that prevent the visualization from being effectively usable in complex programs. That is why the notation is oriented to be used in learning environments where the complexity of the programs to be visualized is reduced.
- In relation to the transcription tasks conducted, the completion of the first task may have affected the second task in terms of learning, even if it was minimal as the participants did not have time to learn between the first and second tasks.
- In this first assessment we have only evaluated the understandability and suitability of the metaphor. In addition to both, other quality attributes such as learnability, modifiability, operability, expressiveness, cognitive effectiveness, etc. can be evaluated. In this sense, there are also some methods of evaluation of notations and visual languages thus as the CDN (Cognitive Dimensions of Notations) framework [3] or the PON (Physics of Notations) [37], which we plan to apply in the future to improve the evaluation of the proposed notation.
- The use case for the use of notation in an augmented reality environment in the classroom is limited by the augmented reality device itself. In this case, an ideal scenario would be one where all students have a visualization device.
- The plugin developed for COLLECE-2.0 is fully usable. However, it is only available for the Java programming language. Using it with others languages (such as C++ through the CDT plugin[2]) should work as well, but has not been tested.

8 Conclusions and future work

In this work we have presented a system for visualizing programs through augmented reality, proposed as an improvement to COLLECE-2.0, a platform oriented to learn programming in a collaborative way through problem solving. To this end, a set of graphic representations had been previously designed to visualize static and dynamic programs, based on a metaphor of roads and traffic signs. The static graphic representation, which is the one that has been approached in this work, has been evaluated with beginner and intermediate CS students in order to measure their perception in terms of understanding and suitability to represent the related programming concepts. The analysis of the results has shown that the notation proposed was well received by the students, making it suitable to represent programs and identifying opportunities for improvement. That is the reason why, after the initial evaluation, new improvements to the graphic representations were proposed, moving them to a 3-D environment which facilitates their deployment using augmented reality techniques and implementing various functionalities that promote the interaction with the system. Furthermore, this new environment has been tested with some students, whose

[2] https://www.eclipse.org/cdt/

first impressions of the 3-D notation have been positive. Thus, the final result has been integrated into COLLECE-2.0 thanks to its extensibility capabilities based on plugins and to the network architecture that it offers.

The implementation described in this paper allows the static visualization of the program structure. These visualization can be generated by teachers during their classes or by students to review the structure of programs. The graphic notation is scalable to generate visualizations of any type of program, although in principle it is oriented to visualize basic programs used during the learning of the programming and that have a reduced number of lines of code. The possibilities of interaction with such visualizations are limited, hence we propose to extend the visualization capabilities to dynamically represent the execution of these programs and algorithms. This would allow to add debugging features such as examining the value of the variables, stopping the execution at a particular point in time, advancing the execution step by step, or exploring concurrency, among others. This extension of functionalities could be accomplished by including, in the metaphor, a vehicle that travels along the roads simulating the execution of the program.

We also consider other uses of the metaphor that move away from static visualization and demonstrate its true potential through, for example, serious game environments or visualization of specific programming concepts (e.g. concurrent programming or recursive programming). In this way, we intend to develop new implementations using the same metaphor of roads and traffic signs proposed in this work.

In addition, we will explore other mechanisms that help students learn, such as choosing other software metrics, allowing to visualize comments in code as additional explanations in the visualization itself, or represent syntax errors in it.

Plus, although all the 3-D representations are expected to be favorably received by users, as discussed in the presented use case, it is necessary to conduct a more formal evaluation of the system and the 3-D notation. Therefore, new experiments are proposed with groups of students, which will serve to accomplish an in-depth analysis of how the use of the whole system affects the learning process of programming. Then, we will compare the road metaphor to other existing approaches.

Acknowledgements This work 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. Bacca J, Baldiris S, Fabregat R, Graf S, Kinshuk (2014) Augmented reality trends in education:, A systematic review of research and applications. Educ Technol Soc 17(4):133–149
2. Bischoff R, Kazi A, Seyfarth M (2002) The morpha style guide for icon-based programming. In: Proceedings of the 11th IEEE International Workshop on Robot and Human Interactive Communication, Berlin, Germany, pp 482–487, https://doi.org/10.1109/ROMAN.2002.1045668
3. Blackwell A, Green T (2003) Notational systems–the cognitive dimensions of notations framework. HCI models, theories, and frameworks: toward an interdisciplinary science. Morgan Kaufmann, Burlington
4. Bloom BS (1956) Taxonomy of educational objectives, vol 1. McKay, New York, pp 20–24. Cognitive domain
5. Bosse Y, Gerosa MA (2017) Why is programming so difficult to learn? ACM SIGSOFT Softw Eng Notes 41(6):1–6
6. Bravo C, Redondo MA, Verdejo MF, Ortega M (2008) A framework for process-solution analysis in collaborative learning environments. Int J Hum Comput Stud 66(11):812–832. https://doi.org/10.1016/j.ijhcs.2008.08.003
7. Bravo C, Duque R, Gallardo J (2013) A groupware system to support collaborative programming: Design and experiences. J Syst Softw 86(7):1759–1771. https://doi.org/10.1016/j.jss.2012.08.039

8. Burgess N, Maguire EA, O'Keefe J (2002) The human hippocampus and spatial and episodic memory. Neuron 35(4):625–641. https://doi.org/10.1016/S0896-6273(02)00830-9
9. Chang SK (1987) Visual languages: a tutorial and survey. IEEE Softw 4(1):29–39. https://doi.org/10.1109/MS.1987.229792
10. Cisar SM, Pinter R, Radosav A, Cisar P (2011) Effectiveness of program visualization in learning java: A case study with jeliot 3. Int J Comput Commun Control 6(4):669–682. https://doi.org/10.15837/ijccc.2011.4.2094
11. Costagliola G, De Rosa M, Fuccella V, Perna S (2018) Visual languages: a graphical review. Inf Vis 17(4):335–350. https://doi.org/10.1177/1473871617714520
12. Dann W, Cooper S, Pausch R (2001) Using visualization to teach novices recursion. ACM SIGCSE Bulletin 33(3):109–112. https://doi.org/10.1145/377435.377507
13. Diehl S (2010) Software visualization: visualizing the structure, behaviour, and evolution of software. Springer, Berlin
14. Dishman L (2020) Why coding is still the most important job skill of the future. https://www.fastcompany.com/3060883/why-coding-is-the-job-skill-of-the-future-for-everyone, accessed: 2020-02-19
15. Dunleavy M, Dede C (2014) Augmented reality teaching and learning. Springer, Berlin, pp 735–745. https://doi.org/10.1007/978-1-4614-3185-5_59
16. Gajraj RR, Williams M, Bernard M, Singh L (2011) Transforming source code examples into programming tutorials. In: Proceedings of the 6th international multi-conference on computing in the global information technology, Luxembourg, pp 160–164
17. GlassDoor (2020) Best Jobs in America 2018. https://www.glassdoor.com/List/Best-Jobs-in-America-2018-LST_KQ0,25.htm, accessed: 2020-02-19
18. Halabi O (2019) Immersive virtual reality to enforce teaching in engineering education. Multimed Tools Appl 79(3):2987–3004. https://doi.org/10.1007/s11042-019-08214-8
19. Hansen WJ (1978) Measurement of program complexity by the pair: (cyclomatic number, operator count). SIGPLAN Not 13(3):29–33. https://doi.org/10.1145/954373.954375
20. Hidalgo-Céspedes J, Marín-Raventós G, Lara-villagrán V (2016) Learning principles in program visualizations: a systematic literature review. In: Proceedings of the 46th Annual Frontiers in Education (FIE) conference, Erie, PA, USA. IEEE, pp 1–9, 10.1109/FIE.2016.7757692
21. Hundhausen CD, Douglas SA, Stasko JT (2002) A meta-study of algorithm visualization effectiveness. J Vis Lang Comput 13(3):259–290. https://doi.org/10.1006/jvlc.2002.0237
22. Jimenez-Diaz G, Gonzalez-Calero PA, Gomez-Albarran M (2012) Role-play virtual worlds for teaching object-oriented design: the virplay development experience. Softw Pract Exp 42(2):235–253. https://doi.org/10.1002/spe.1071
23. Jurado F, Molina AI, Redondo MA, Ortega M, Giemza A, Bollen L, Hoppe HU (2009) Learning to program with coala, a distributed computer assisted environment. J Univers Comput Sci 15(7):1472–1485. https://doi.org/10.3217/jucs-015-07-1472
24. Jurado F, Molina AI, Redondo MA, Ortega M (2013) Cole-programming: Shaping collaborative learning support in eclipse. IEEE Revista Iberoamericana de Tecnologias del Aprendizaje 8(4):153–162. https://doi.org/10.1109/RITA.2013.2284953
25. Kann C, Lindeman RW, Heller R (1997) Integrating algorithm animation into a learning environment. Comput Educ 28(4):223–228. https://doi.org/10.1016/S0360-1315(97)00015-8
26. Karp RM (1972) Reducibility among combinatorial problems. Complex Comput Comput 1(1):85–103. https://doi.org/10.1007/978-1-4684-2001-2_9
27. Knight C, Munro M (2000) Virtual but visible software. In: Proceedings of the IEEE international conference on information visualisation, Londonm United Kingdom, pp 198-205, https://doi.org/10.1109/IV.2000.859756
28. Knuth DE (1997) The art of computer programming: sorting and searching, vol 3. Pearson Education, London
29. Koschmann T (1996) Paradigm shifts and instructional technology:, An introduction, Book section, vol 1, pp 1–23
30. Lacave C, Molina AI, Giralt J (2013) Identificando algunas causas del fracaso en el aprendizaje de la recursividad: análisis experimental en las asignaturas de programación. In: Proceedings of the XIX Jornadas sobre la Enseñanza Universitaria de la Informática, Castellón de la Plana. Universitat Jaume I. Escola Superior de Tecnologia i Ciències Experimentals, Spain, pp 225-232
31. Lacave C, Paredes-Velasco M, Àngel Velázquez-Iturbide J, Hernán I (2017) Experiencia para la evaluación de visback, una herramienta para la visualización de algoritmos de backtracking. Informá,tica Educativa Comunicaciones 23(26):83–94

32. Levy RBB, Ben-Ari M, Uronen PA (2003) The jeliot 2000 program animation system. Comput Educ 40(1):1–15. https://doi.org/10.1016/S0360-1315(02)00076-3
33. Mathur AS, Ozkan BK, Majumdar R (2018) Idea: an immersive debugger for actors. In: Proceedings of the 17th ACM SIGPLAN International Workshop on Erlang, St. Louis, MO, USA, ACM, vol 3242762, pp 1–12. https://doi.org/10.1145/3239332.3242762
34. McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng SE 2(4):308–320. https://doi.org/10.1109/TSE.1976.233837
35. Milne I, Rowe G (2004) Ogre: Three-dimensional program visualization for novice programmers. Educ Inf Technol 9(3):219–237. https://doi.org/10.1023/B:EAIT.0000042041.04999.17
36. Molina AI, Gallardo J, Redondo MA, Bravo C (2014) Evaluating the awareness support of collece, a collaborative programming tool. In: Proceedings of the XV International Conference on Human Computer Interaction, Puerto de la Cruz. Tenerife, Spain, pp 74-81, https://doi.org/10.1145/2662253.2662264
37. Moody D (2009) The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Trans Softw Eng 35(6):756–779
38. Myers BA (1990) Taxonomies of visual programming and program visualization. J Vis Lang Comput 1(1):97–123. https://doi.org/10.1016/S1045-926X(05)80036-9
39. Naps TL, Rößling G, Almstrum V, Dann W, Fleischer R, Hundhausen C, Korhonen A, Malmi L, McNally M, Rodger S (2002) Exploring the role of visualization and engagement in computer science education. In: Proceedings of the 7th annual conference on innovation and technology in computer science education, Aarhus, Denmark, ACM, vol 35, pp 131-152. https://doi.org/10.1145/960568.782998
40. Nassi I, Shneiderman B (1973) Flowchart techniques for structured programming. ACM SIGPLAN Not 8(8):12–26. https://doi.org/10.1145/953349.953350
41. National Center for Education Statistics (NCES) (2020) IPEDS Completions Survey. https://nces.ed.gov/ipeds/, accessed: 2020-02-19
42. Nichols DA, Curtis P, Dixon M, Lamping J (1995) High-latency, low-bandwidth windowing in the jupiter collaboration system. In: Proceedings of the 8th annual ACM symposium on User interface and software technology, Pittsburgh, Pennsylvania, USA. ACM, pp 111–120, https://doi.org/10.1145/215585.215706
43. Ortega M, Redondo MÁ, Molina AI, Bravo C, Lacave C, Arroyo Y, Sánchez S, García MÁ, Collazos CA, Toledo JJ, Luna-García H, Velázquez-Iturbide JÁ, Gómez-Pastrana RA (ACM) Iprog: development of immersive systems for the learning of programming. In: Proceedings of the XVIII international conference on human computer interaction, Cancun, Mexico, vol 3123874, pp 1–6. https://doi.org/10.1145/3123818.3123874
44. Price B, Baecker R, Small I (1998) An introduction to software visualization, book section, vol 1. MIT Press, Cambridge, pp 3–27
45. Robertson GG, Card SK, Mackinlay JD (1993) Information visualization using 3d interactive animation. Commun ACM 36(4):57–71. https://doi.org/10.1145/255950.153577
46. Rowe G, Thorburn G (2000) Vince—an on-line tutorial tool for teaching introductory programming. Br J Educ Technol 31(4):359–369. https://doi.org/10.1111/1467-8535.00168
47. Sajaniemi J, Kuittinen M (2003) Program animation based on the roles of variables. In: Proceedings of the 2003 ACM symposium on Software visualization, San Diego, California, USA, ACM, pp 7–ff. https://doi.org/10.1145/774833.774835
48. Sanchez S, Garcia MA, Lacave C, Molina AI, Gonzalez C, Vallejo D, Redondo MA (2018) Applying mixed reality techniques for the visualization of programs and algorithms in a programming learning environment. In: Kush JC, Lester C (eds) Proceedings of the 10th International Conference on Mobile, Hybrid, and On-line Learning, Rome, Italy, IARIA XPS Press, vol 1, pp 84-89
49. Sánchez S, García MÁ, Bravo C, Redondo MÁ (2017) Sistema collece mejorado para soportar aprendizaje colaborativo de la programación en tiempo real sobre eclipse. IE Comunicaciones 23(26):72–81
50. Sánchez S, Redondo MA, Vallejo D, González C, Bravo C (2017) Collece 2.0: A distributed real-time collaborative programming environment for the eclipse platform. In: Proceedings of the 11th International Conference on Interfaces and Human Computer Interaction, Lisbon, Portugal, IADIS, pp 1-7
51. Simonak S (2016) Algorithm visualizations as a way of increasing the quality in computer science education. In: Proceedings of the 14th international symposium on applied machine intelligence and informatics, Herlany, Slovakia, pp 153-157, https://doi.org/10.1109/SAMI.2016.7422999
52. Teng CH, Chen JY (2012) An augmented reality environment for learning opengl programming. In: Proceedings of the 9th international conference on ubiquitous intelligence and computing and 9th international conference on autonomic and trusted computing, Fukuoka, Japan. IEEE, pp 996-1001, https://doi.org/10.1109/UIC-ATC.2012.57

53. Teng CH, Chen JY, Chen ZH (2017) Impact of augmented reality on programming language learning: Efficiency and perception. J Educ Comput Res 56(2):254–271. https://doi.org/10.1177/0735633117706109
54. Teyseyre AR, Campo MR (2009) An overview of 3d software visualization. IEEE Trans Vis Comput Graph 15(1):87–105. https://doi.org/10.1109/TVCG.2008.86
55. UNECE (1968) Convention on road signs and signals. United Nations Treaty Series, vol. 1091 pp 3
56. US Bureau of Labor Statistics (2020) Employment by detailed occupation. https://www.bls.gov/emp/tables/emp-by-detailed-occupation.htm, accessed: 2020-02-19
57. Urquiza-Fuentes J, Velázquez-iturbide JÁ (2012) A long-term evaluation of educational animations of functional programs. In: Proceedings of the 12th International Conference on Advanced Learning Technologies, Rome, Italy. IEEE, pp 26-30, https://doi.org/10.1109/ICALT.2012.50
58. Urquiza-Fuentes J, Velázquez-Iturbide JA (2013) Toward the effective use of educational program animations: the roles of student's engagement and topic complexity. Comput Educ 67(Supplement C):178–192. https://doi.org/10.1016/j.compedu.2013.02.013
59. Vasilopoulos IV, van Schaik P (2018) Koios: Design, development, and evaluation of an educational visual tool for greek novice programmers. J Educ Comput Res. https://doi.org/10.1177/0735633118781776
60. Velázquez-Iturbide JA, Hernán-Losada I, Paredes-Velasco M (2017) Evaluating the effect of program visualization on student motivation. IEEE Trans Educ 60(3):238–245. https://doi.org/10.1109/TE.2017.2648781
61. Vujošević-Janičić M, Tošić D (2008) The role of programming paradigms in the first programming courses. Teach Math 11(2):63–83
62. Wang P, Bednarik R, Moreno A (2012) During automatic program animation, explanations after animations have greater impact than before animations. In: Proceedings of the 12th Koli calling international conference on computing education research, New York, USA, pp 100–109, https://doi.org/10.1145/2401796.2401808
63. Yoel A, MAI, Carmen L, RMA, Manuel O (2018) The greedex experience: Evolution of different versions for the learning of greedy algorithms. Comput Appl Eng Educ 26(5):1306–1317. https://doi.org/10.1002/cae.22023

2.2 Other articles published in indexed scientific journals

A novel approach to learning music and piano based on mixed reality and gamification

- Title: A novel approach to learning music and piano based on mixed reality and gamification [26]
- Authors: Diego Molero, Santiago Sánchez, David Vallejo, Carlos González, and Javier Alonso Albusac
- Type: Journal
- Journal: Multimedia Tools and Applications
- Publisher: Springer Nature
- E-ISSN: 1573-7721
- Year: 2020
- DOI: 10.1007/s11042-020-09678-9
- Category: Computer Science, Software Engineering
- Impact Factor (2019): 2.313
- JCR Ranking: Q2
- Related to the current research topic: No, although the work conducted during the course of the referenced research has allowed to extend the knowledge on the use of AR immersive devices and the different communication protocols and codification of the information, necessary to satisfactorily perform the implementations of the software solutions developed for the work of this doctoral dissertation.
- Related figure(s): 2.1
- Abstract: Learning music has been demonstrated to provide many benefits for children. However, music students, especially beginners, often suffer from lack of motivation and even can be frustrated if their musical skills do not improve as they practice over and over. In such situations, they usually end up dropping out of music school. To face this challenge, in this work a novel approach based on mixed reality and gamification is proposed to motivate music students. This approach has been validated thanks to HoloMusic XP, a multimedia tool that helps students learn music and piano. The devised architecture that supports HoloMusic XP has been designed and developed to scale when new music concepts must be addressed. Thanks to the use of mixed reality, the usually steep learning curve for

beginner students can be mitigated and complex music concepts can be simplified due to the use of visual metaphors. The system has been evaluated in a real environment by teachers and students to measure its effectiveness and usability. After conducting the experiments, an increase in the students' motivation and a general understanding of the multimedia representation have been achieved.

A Distributed Gamified System Based on Automatic Assessment of Physical Exercises to Promote Remote Physical Rehabilitation

- Title: A Distributed Gamified System Based on Automatic Assessment of Physical Exercises to Promote Remote Physical Rehabilitation [45]
- Authors: Santiago Sánchez, David Vallejo, Dorothy Ndedi Monekosso, Carlos González, and Paolo Remagnino
- Type: Journal
- Journal: IEEE Access
- Publisher: IEEE
- E-ISSN: 2169-3536
- Year: 2020
- DOI: 10.1109/ACCESS.2020.2995119
- Category: Computer Science, Information Systems
- Impact Factor (2019): 3.745
- JCR Ranking: Q1
- Related to the current research topic: No, although the work done to publish the results in this research article has allowed to expand the knowledge related to computer graphics representation, voice and gesture interaction mechanisms, signal processing and analysis, and high performance computing in devices with reduced capacities. This knowledge has been applied during the course of this research to the software solutions implemented and the visual prototypes made. Likewise, the experiments performed with stroke survivors in the referenced research have served to correct and improve the aspects to be evaluated in the evaluations made with students in the subject of the doctoral dissertation.
- Related figure(s): 2.2

- Abstract: Physical rehabilitation aims at improving the functional ability and quality of life of patients affected by physical impairments or disabilities. Neurological diseases represent the largest cause of disability worldwide. For many, there is no cure and physiotherapy allows symptoms to be managed. Physiotherapy is based on the daily execution of exercises, traditionally under the supervision of a therapist. However, performing these exercises requires that both the patient and the physiotherapist are together so that the physiotherapist can assist the patient while exercising. For patients with a neurological condition, rehabilitation is a long term process, lasting months or even years. Not withstanding the personal costs, the cost of care/physiotherapy is high and represents 27,711 € per year in Spain. This is compounded by a shortage of qualified therapists, often cited as one reason why stroke survivors do not received the recommended amount of therapy. The challenge is even greater in low to mid-income countries where there is a lack of trained personnel as well as under-served and remote regions. Technology can be employed to alleviate these problems by remotely monitoring a rehabilitation session taking place at home or anywhere in the community. This paper presents a computer vision-based system for home-use that automatically assesses how well the patient performs the exercises and transmits the information back to the clinic. The patient and physiotherapist do not need to be co-located. Gamification methods and techniques are used to engage patients when carrying out the rehabilitation routines. To this end, we propose a distributed gamified system that automatically evaluates the performance of exercises by analyzing and comparing motion curves using the DTW (Dynamic Time Warping) algorithm.

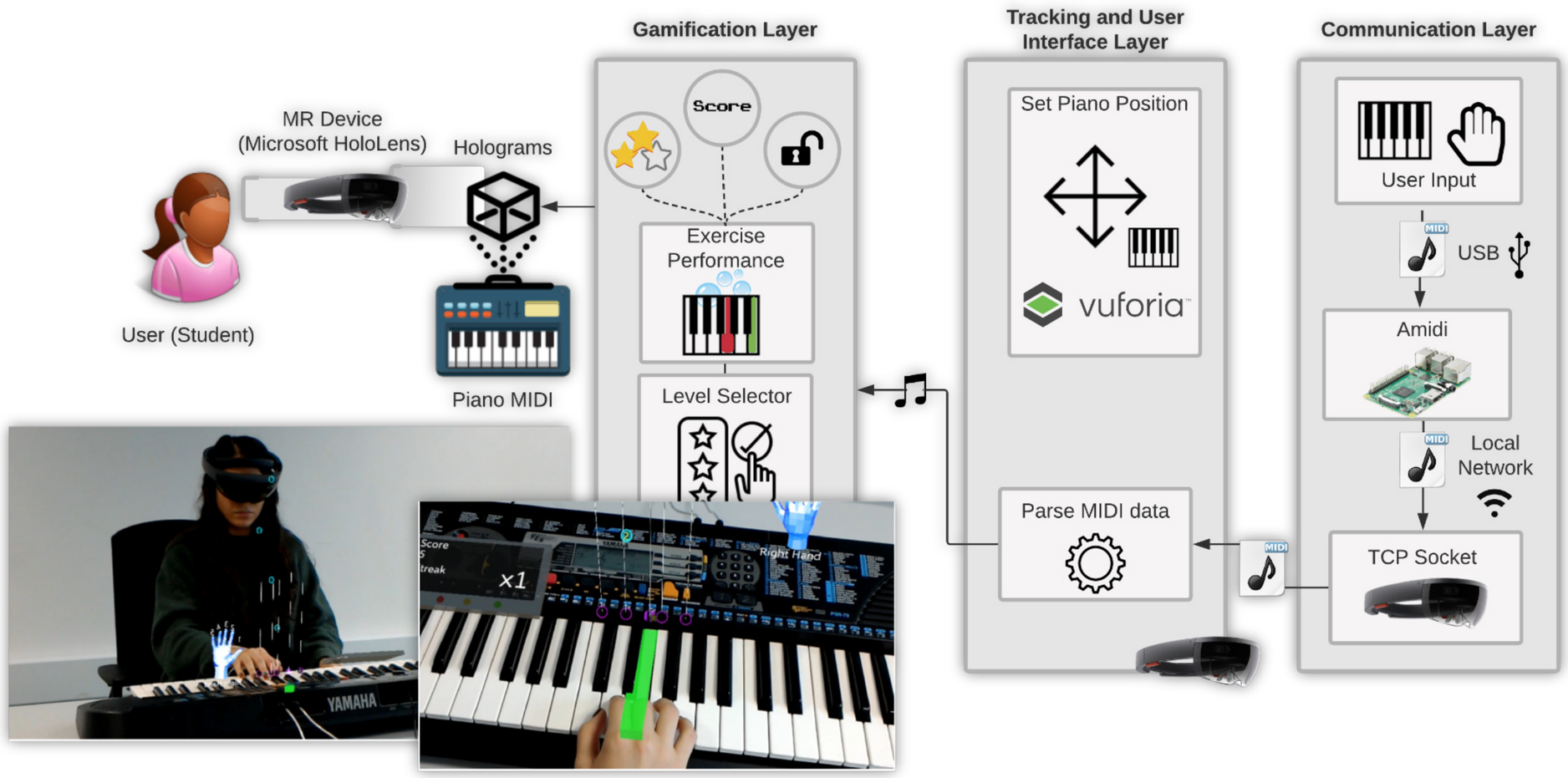


Figure 2.1: One of the evaluated user playing HoloMusic XP (bottom-left) and the main workflow diagram of the system (background).

Figure 2.2: Main snapshot of the created solution (left) along with a movement curve comparison (top-right) and the recognition of two physical exercises over time (bottom-right).

2.3 Articles presented in national and international conferences

Propuesta y Evolución Multidimensional de una Metáfora Visual para Facilitar el Aprendizaje de la Programación

- Title: Propuesta y Evolución Multidimensional de una Metáfora Visual para Facilitar el Aprendizaje de la Programación [46]
- Authors: Santiago Sánchez, Maria de los Ángeles García, Cristian Gómez, Carlos González, David Vallejo, Javier Alonso Albusac, and Miguel Ángel Redondo
- Type: Conference
- Conference: XX International Conference on Human-Computer Interaction
- Location: Donostia, Spain
- Year: 2019
- Awards: **Jesús Lorés award to the best research article**
- Related to the current research topic: Yes.
- Related figure(s): 2.3
- Abstract: Computer programming is a complex task and a challenge for students who are starting to take an interest in it. Specifically, students in the first year of the Bachelor of Engineering in Computer Science show certain difficulties in understanding programming concepts due to the high level of abstraction required for their learning. This process of learning programming can be facilitated by graphical representations that allow the student to establish analogies between the concepts it seeks to understand and other elements of the real world. The current literature proposes certain approaches that provide different alternatives to visualize programs and algorithms, either statically, showing their structure, or dynamically, showing their execution. Some of these approaches limit the potential of visualization by focusing on showing the source code of programs over a virtual world; others try to explain specific concepts of programming in isolation, causing the student to lose the context of the entire program. This work introduces the proposal and evolution of a new set of graphic representations towards a 3D environment of augmented reality, based on the metaphor of roads and traffic signs, and which aims to facilitate the learning of programming to beginner students. The visualizations generated from these graphical representations can be constructed automatically thanks to their modular design and used by teachers in order to explain programming concepts during master

classes. The proposal has been evaluated with students in order to validate whether the proposed notation is appropriate to represent the concepts it tries to abstract and easy for students to understand.

Personalising Exergames for the Physical Rehabilitation of Children Affected by Spine Pain

- Title: Personalising Exergames for the Physical Rehabilitation of Children Affected by Spine Pain [32]
- Authors: Cristian Gómez, Carmen Lacave, Ana Isabel Molina, David Vallejo, and Santiago Sánchez
- Type: Conference
- Conference: XXII International Conference on Enterprise Information Systems (ICEIS)
- Location: Tomar, Portugal
- Year: 2020
- Awards: **Best Paper Award in the Area of Human-Computer Interaction**
- DOI: 10.5220/0009574005330543
- Related to the current research topic: No, although the achievement of this research allowed to consolidate the knowledge related to the statistical evaluation and presentation of the results of the present research of this doctoral dissertation.
- Related figure(s): 2.4
- Abstract: Injuries or illnesses related to the lumbar spine need great clinical care as they are one of the most prevalent medical conditions worldwide. The use of exergames has been widespread in recent years and they have been put forward as a possible solution for motivating patients to perform rehabilitation exercises. However, both customizing and creating them is still a task that requires considerable investment both in time and effort. In this project we present a language with which we have designed a system based on the physical rehabilitation of patients suffering from bone-marrow injuries, which enables customization and generation of exergames. To assess the system, we have designed an experiment with an exergame based on the physical rehabilitation of the lumbar spine. The purpose of this was to assess its understanding and suitability, whose result reveals that the tool is fun, interesting and easy to use. It is hoped that this approach can be used to considerably reduce the complexity of creating new exergames, as well as supporting the physical rehabilitation process of patients with lower back pain.

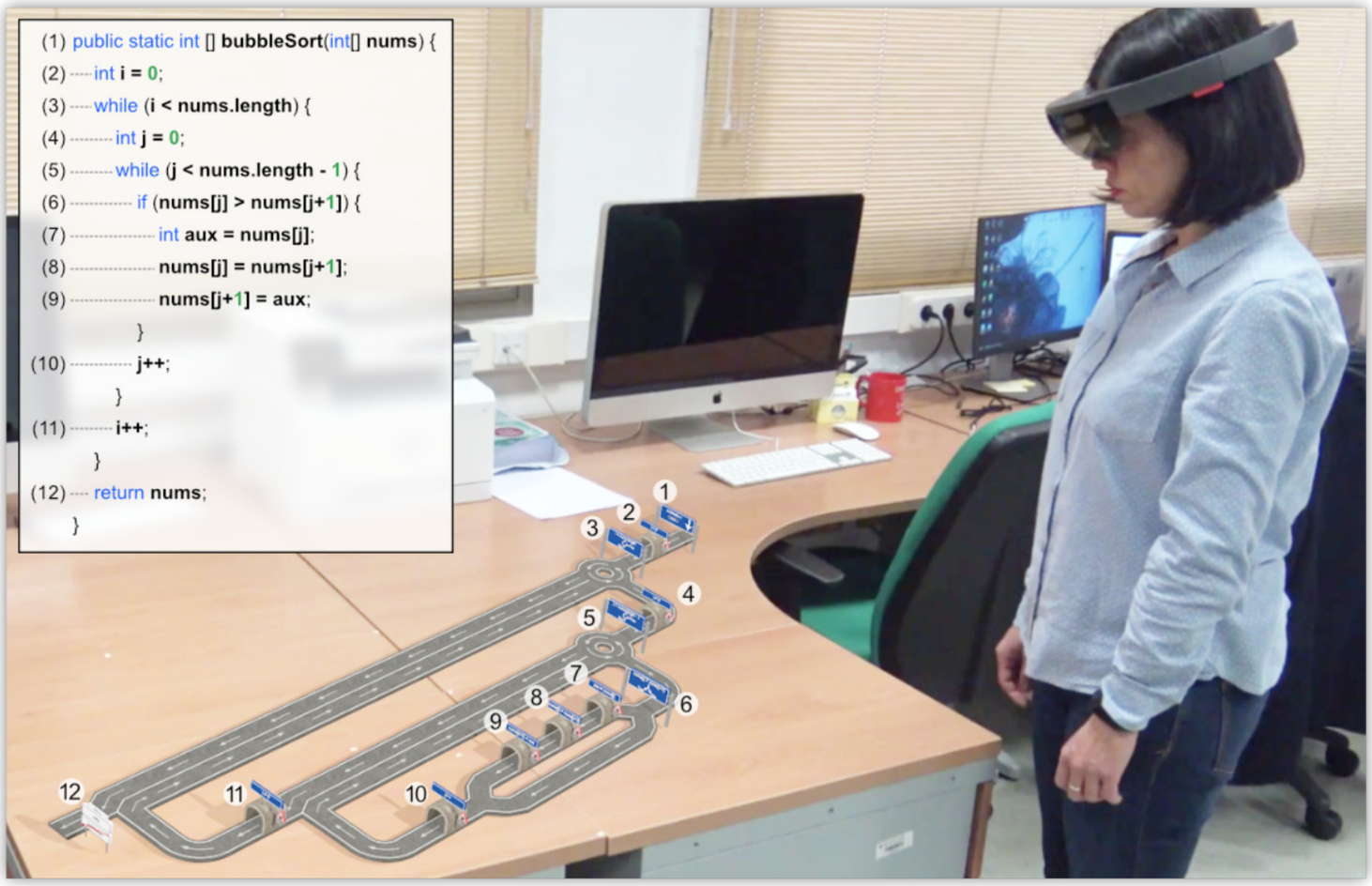


Figure 2.3: A photo taken of a user testing the system to visualize programs represented through the ANGELA notation along with the overlayed processed source code relating each graphic representation from the visualization.

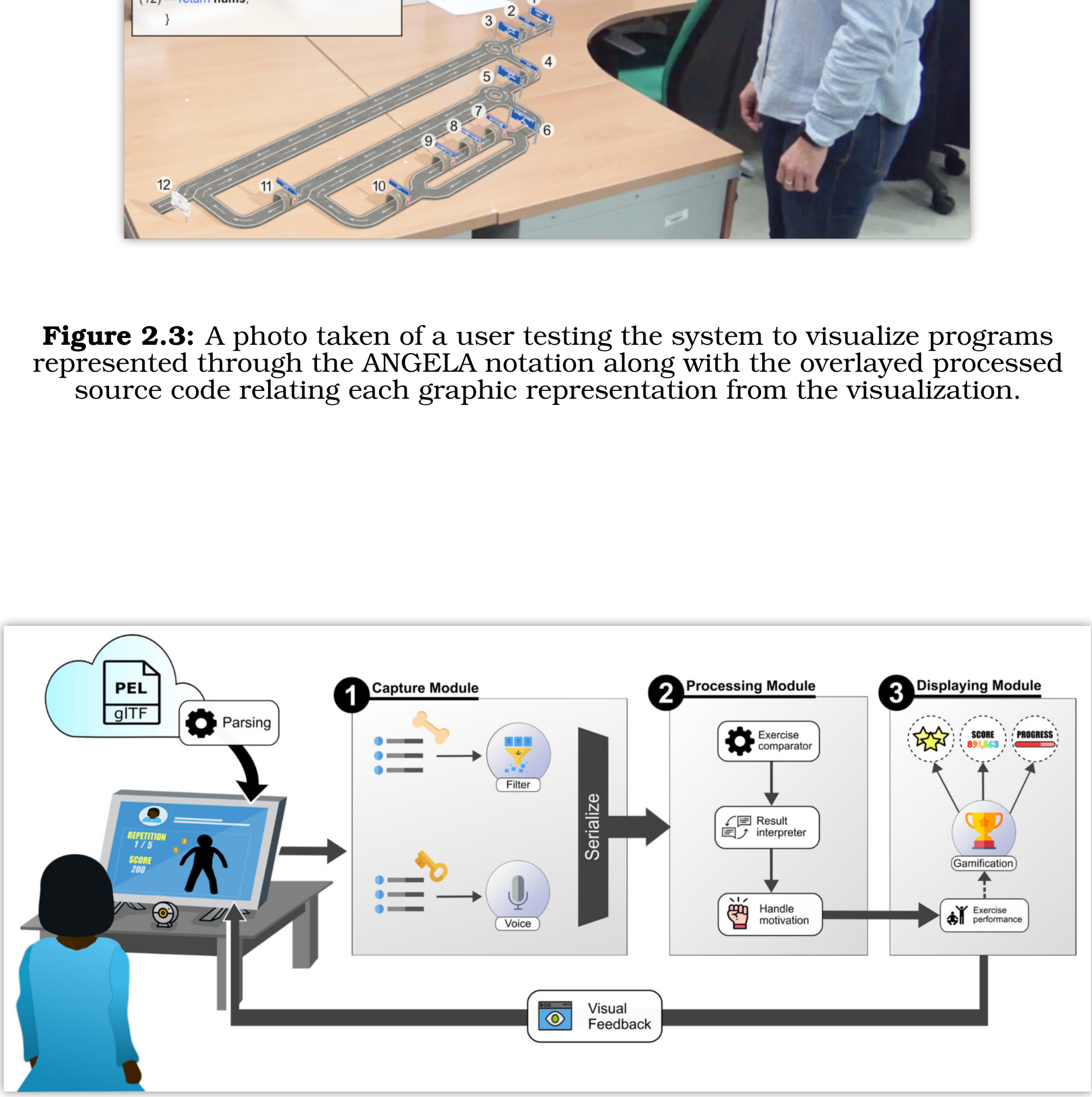


Figure 2.4: Proposed architecture for rehabilitation of patient suffering from bone-marrow injuries.

Collece 2.0: A distributed real-time collaborative programming environment for the eclipse platform

- Title: Collece 2.0: A distributed real-time collaborative programming environment for the eclipse platform [39]
- Authors: Santiago Sánchez, Miguel Ángel Redondo, David Vallejo, Carlos González, and Crescencio Bravo
- Type: Conference
- Conference: XI International Conference on Interfaces and Human Computer Interaction
- Location: Lisbon, Portugal
- Year: 2017
- ISBN: 978-989-8533-64-7
- Related to the current research topic: Yes.
- Related figure(s): 2.5
- Abstract: Collaboration with other users to solve programming problems has an effect on the speed of development and the final software quality. This is achieved through techniques like pair programming when users are co-located, or otherwise through real-time remote collaboration. Existing work for remote working on a programming project has failed to address the integration of new emerging technologies, like augmented reality. These new technologies would provide a way to improve the final user learning experience if integrated in a working programming environment. This paper describes the implementation of COLLECE 2.0, a plug-in for the Eclipse platform which provides a distributed real-time collaborative programming environment, and integrates support for augmented reality and mobile device capabilities. We discuss two specific use scenarios where the solution would work, and how the implemented awareness mechanisms improve the user experience. The paper closes with the conclusions, limitations and future work, and a first sign on what on-going works would look like.

iProg: development of immersive systems for the learning of programming

- Title: iProg: development of immersive systems for the learning of programming [29]
- Authors: Manuel Ortega, Miguel Ángel Redondo, Ana Isabel Molina, Crescencio Bravo, Carmen Lacave, Yoel Arroyo, Santiago Sánchez, María de los Ángeles García, César Alberto Collazos, Javier Alejandro Jiménez, Huizilopoztli Luna, José Ángel Velázquez, and Raúl Abad
- Type: Conference
- Conference: XVIII International Conference on Human-Computer Interaction
- Location: Cancun, Mexico
- Year: 2017
- DOI: 10.1145/3123818.3123874
- Related to the current research topic: Yes.
- Related figure(s): 2.6
- Abstract: Basic “computing literacy” is said to be deemed necessary for all citizens, and provides an opportunity to prepare, over longer periods of time, future computing engineers. The iProg Project intends to achieve computing literacy research objectives by means of a number of applications for Programming Education, based on different techniques for advanced Computer-Human Interaction and visualization, including augmented reality and gesture-based interaction. The evaluation of Usability and User Experience of these new forms of interaction will require the use of advanced interaction techniques, including eye tracking and the gathering of biometric information.

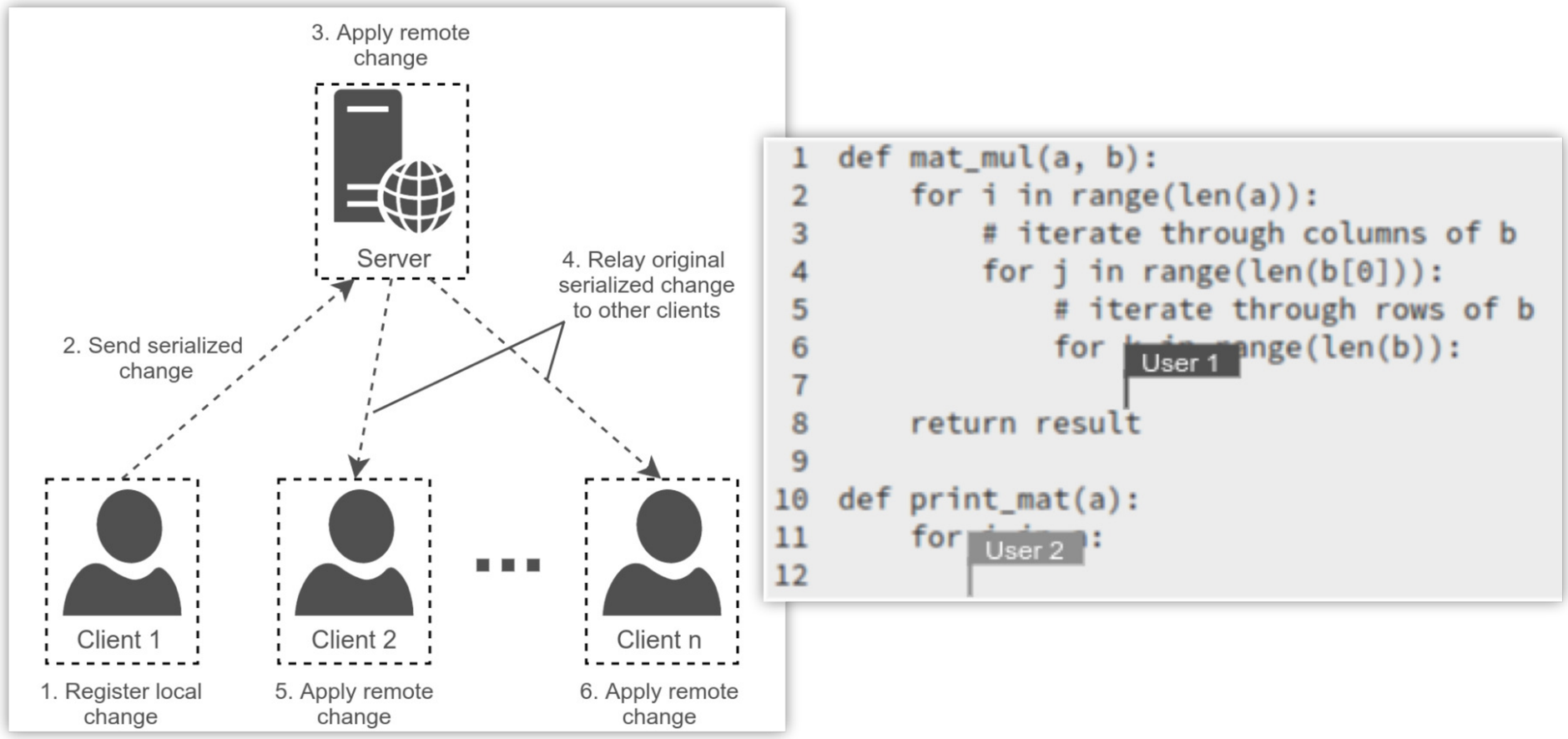


Figure 2.5: High level algorithm used to synchronize changes among different users editing the same file (left) and a representation of what would be the future editor of COLLECE-2.0 (right).

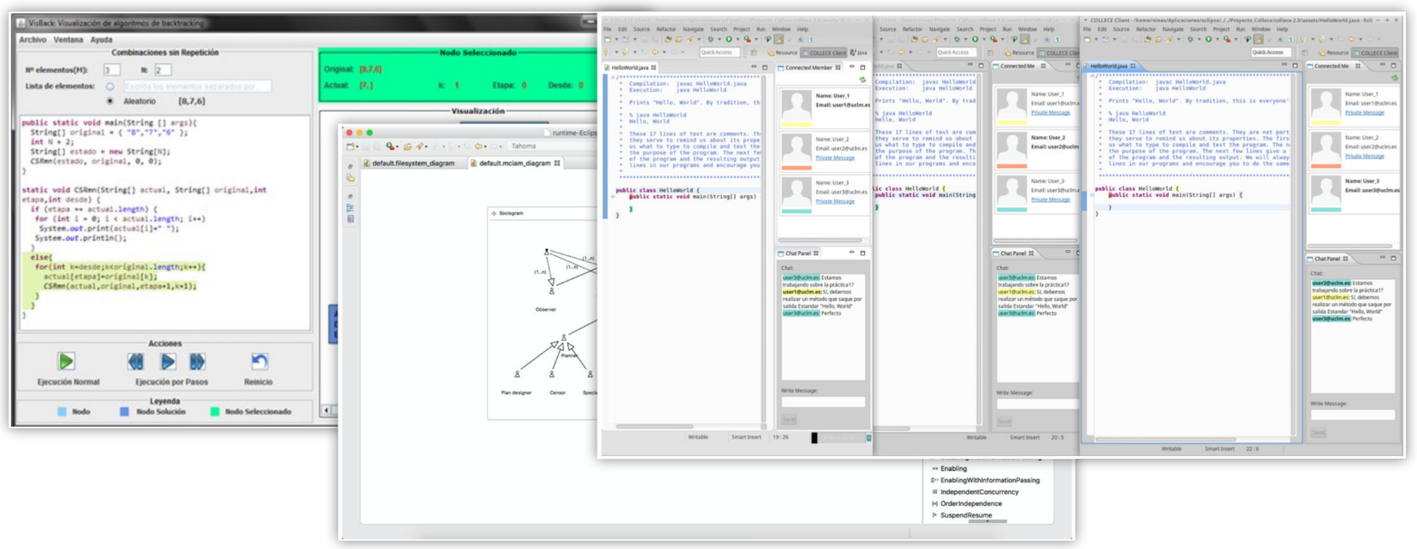


Figure 2.6: Three of the systems presented by the CHICO research group: VisBack, for the visualization of recursive programs (left); Learn CIAT tool for the CIAM methodology (middle); COLLECE-2.0 for the collaborative learning of programming (right).

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

- Title: Applying Mixed Reality Techniques for the Visualization of Programs and Algorithms in a Programming Learning Environment [40]
- Authors: Santiago Sánchez, Maria de los Ángeles García, Carmen Lacave, Ana Isabel Molina, Carlos González, David Vallejo, and Miguel Ángel Redondo
- Type: Conference
- Conference: X International Conference on Mobile, Hybrid, and On-line Learning (eLmL)
- Location: Rome, Italy
- Year: 2018
- ISSN: 2308-4367
- Related to the current research topic: Yes.
- Related figure(s): 2.7
- 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.

Automatic recognition of physical exercises performed by stroke survivors to improve remote rehabilitation

- Title: Automatic recognition of physical exercises performed by stroke survivors to improve remote rehabilitation [44]
- Authors: Santiago Sánchez, Dorothy Ndedi Monekosso, Paolo Remagnino, David Vallejo, and Carlos González
- Type: Conference
- Conference: II International Conference on Multimedia Analysis and Pattern Recognition (MAPR)
- Location: Ho Chi Minh City, Vietnam
- Year: 2019
- DOI: 10.1109/MAPR.2019.8743535
- Related to the current research topic: No, although the research done for this conference publication allowed to expand the knowledge about concurrent programming, communication and synchronization between immersive devices and network servers, and graphic rendering in web applications. Likewise, the work conducted had an impact on one of the scientific publications in the JCR journal mentioned in the previous section.
- Related figure(s): 2.8
- Abstract: Strokes are the second cause of death and the third cause of disability in the world. Currently, there is not an actual cure for stroke victims, but physiotherapy can be used to restore as much mobility as possible until a plateau is reached again. However, performing these exercises implies that both patient and therapist are together in the same place, so that the latter can guide the former through a correct execution of physical exercises. This raises additional difficulties when it comes to continuously monitoring the recovery of stroke patients, due to the economic costs involved and the requirements of both geographical and temporal availability. These issues can be addressed by leveraging technology, specifically computer vision-based assistive systems and remote rehabilitation tools, so as to the affected person can check whether the exercises are being performed correctly. This paper is focused on the automatic classification of exercises, within the context of a gamification-based remote rehabilitation tool used to automatically assess the performance of stroke patients when making physical rehabilitation. To this end, we use the DTW algorithm for analyzing and comparing open-ended motion curves, so that the exercises do not have to be fully performed until existing candidates can be matched. This increases the system flexibility and offers an interaction mechanism much more simple, which is usually a needed requirement by stroke patients.

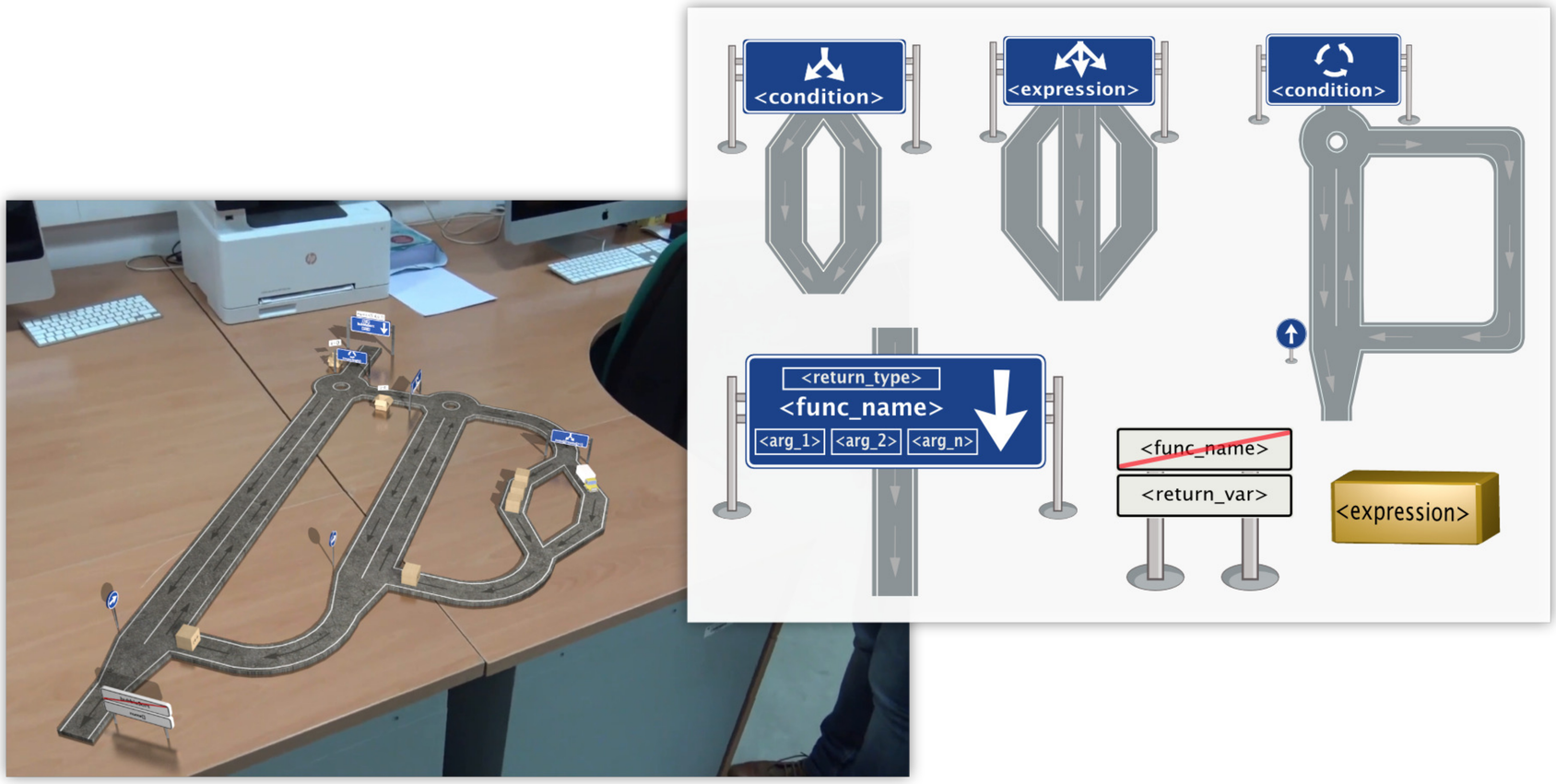


Figure 2.7: A 3-D visualization of the bubble sort algorithm (left) using a preliminary version of the ANGELA notation (right).

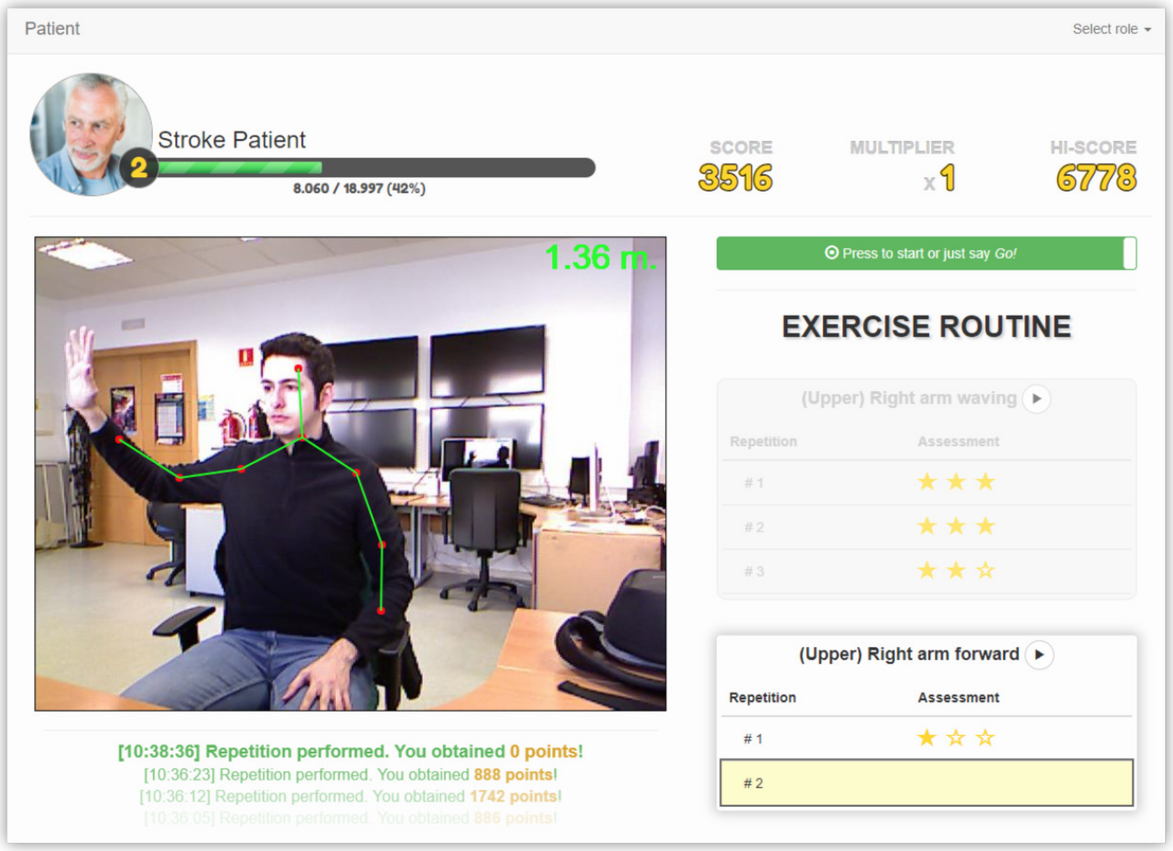


Figure 2.8: The software solution created to help during rehabilitation of stroke survivors through analyzing physical exercises and providing information about the patient's evolution.

ANGELA: A Novel Approach of Graphic Notation Based on the Metaphor of Road Signs to Facilitate the Learning of Programming

- Title: ANGELA: A Novel Approach of Graphic Notation Based on the Metaphor of Road Signs to Facilitate the Learning of Programming [47]
- Authors: Santiago Sánchez, María de los Angeles García, Cristian Gómez, David Vallejo, Ana Isabel Molina, Carmen Lacave, Carlos González, Javier Alonso Albusac, and Miguel Ángel Redondo
- Type: Conference
- Conference: VII International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM)
- Location: León, Spain
- Year: 2019
- DOI: 10.1145/3362789.3362871
- Related to the current research topic: Yes.
- Related figure(s): 2.9
- Abstract: Programming is a field that influences other disciplines in a transversal way, so its learning is necessary considering the emergence of new jobs that will require programming knowledge in the future. However, programming raises certain difficulties during its learning, especially in understanding programming concepts due to the high level of abstraction required. This level of abstraction can be reduced by using graphic representations that motivate the student and facilitate the understanding of certain programming concepts that arise at the beginning of the learning process. Therefore, this paper introduces ANGELA, a graphic notation based on the metaphor of roads and traffic signs that is meant to complement the learning process of beginner students who are starting to program by visualizing programs. These visualizations can be automatically generated from the source code of the programs, thanks to the modular and scalable design of the notation, and used by teachers to explain programming concepts during classes. The proposal has been evaluated with students in order to validate if the notation is appropriate to represent the concepts it tries to abstract from and if it results easy for the students to understand. Additionally, some use cases are presented in real-world scenarios in order to demonstrate the flexibility of the proposal.

COLLECE-2.0: A real-time collaborative programming system on Eclipse

- Title: COLLECE-2.0: A real-time collaborative programming system on Eclipse [23]
- Authors: Carmen Lacave, María de los Ángeles García, Ana Isabel Molina, Santiago Sánchez, Miguel Ángel Redondo, and Manuel Ortega
- Type: Conference
- Conference: XXI International Symposium on Computers in Education (SIIE)
- Location: Tomar, Portugal
- Year: 2019
- DOI: 10.1109/siie48397.2019.8970132
- Related to the current research topic: Yes.
- Related figure(s): 2.10
- Abstract: There is evidence that group learning techniques facilitate programming learning. Existing tools have limited functionalities, mainly in relation to the synchronization and consistency of shared documents. The COLLECE-2.0 system, a plug-in for the Eclipse platform, provides a collaborative programming environment, distributed and in real time. Its interface has been designed with a special emphasis on aspects related to awareness and collaboration. This work describes a first experience developed to evaluate this system. In general, all aspects of the evaluated interface have been well appreciated, especially the public chat between all members of the group and the possibility that areas of code can be locked to avoid unwanted changes during collaborative editing.

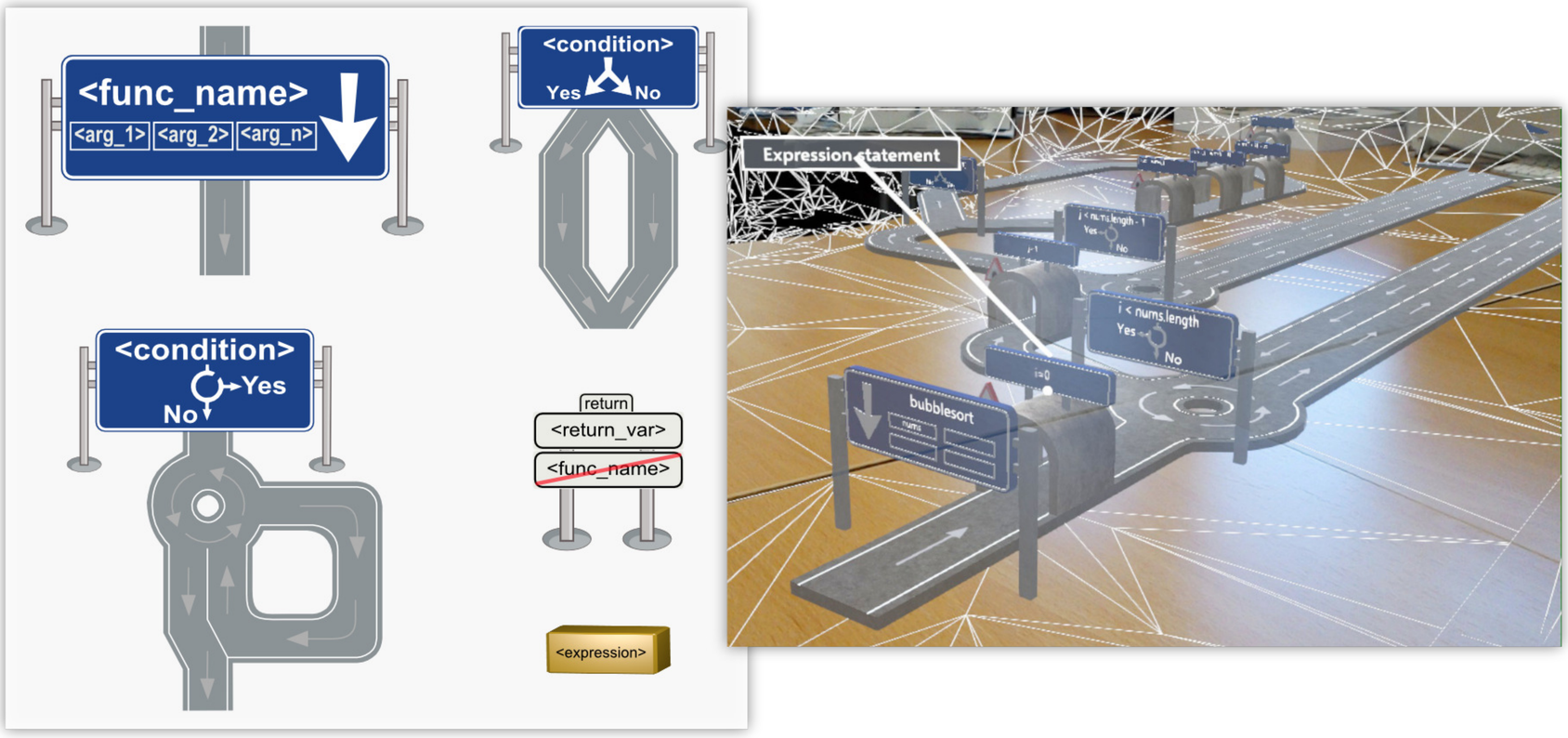


Figure 2.9: New visual and usage improvement to the 2-D (left) and 3-D (right) ANGELA notation.

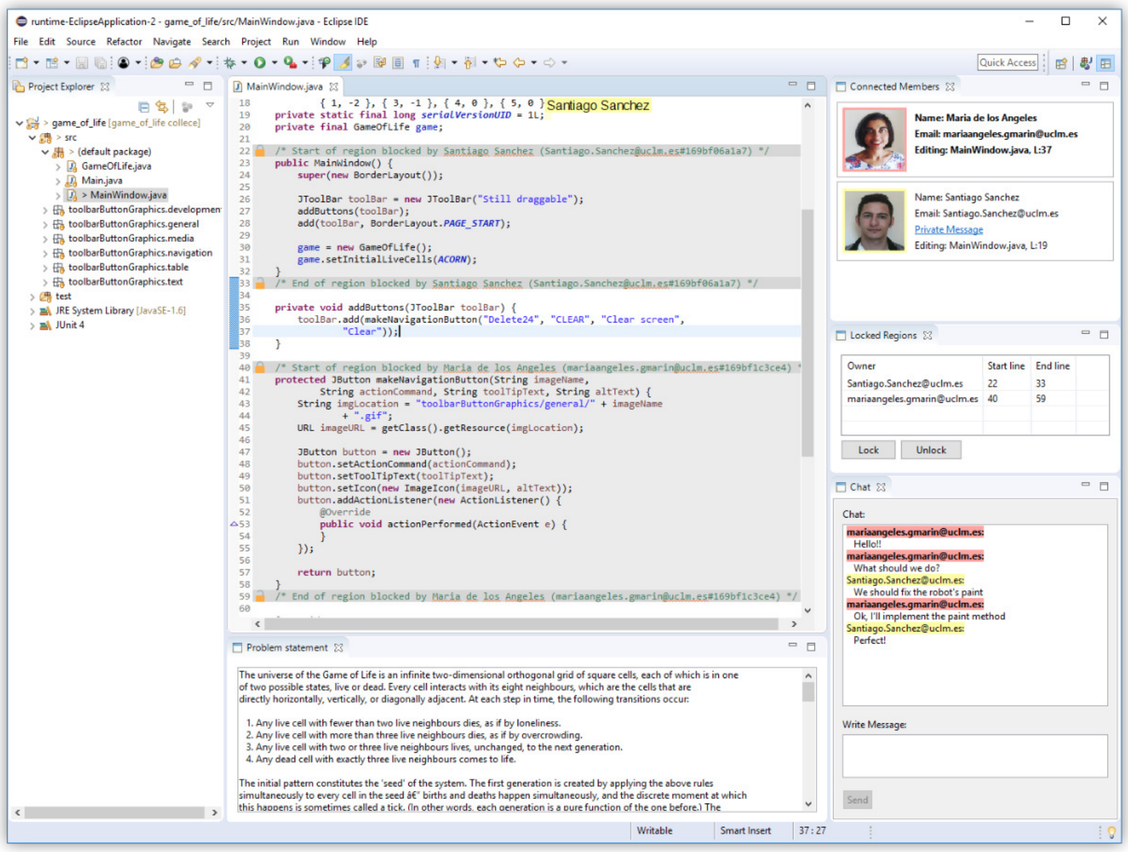


Figure 2.10: A snapshot showing some of the main features available in COLLECE-2.0.

An Agent-based Approach to Physical Rehabilitation of Patients affected by Neurological Diseases

- Title: An Agent-based Approach to Physical Rehabilitation of Patients affected by Neurological Diseases [58]
- Authors: David Vallejo, Santiago Sánchez, Javier Alonso Albusac, José Jesús Castro, and Carlos González
- Type: Conference
- Conference: IX International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH)
- Location: Coimbra, Portugal
- Year: 2019
- DOI: 10.1016/j.procs.2019.11.081
- Related to the current research topic: No, although the knowledge acquired during this work had a direct impact on the publications related to the serious game for learning concepts of programming called RoboTIC, where an agent-based approach was proposed for the implementation of the prototype.
- Related figure(s): 2.11
- Abstract: Neurological disorders such as strokes, dementia, or ABI, among others, represent a major burden on European and worldwide healthcare systems, and can be understood as an unmet clinical need, recognised as a global challenge. This work discusses how agent technology can contribute to increase the autonomy of patients affected by neurological diseases who require physical rehabilitation. This approach can reduce face-to-face rehabilitation and the associated costs. We propose an architecture composed of software agents that play different roles to support the rehabilitation process from the point of view of patients and clinicians. This architecture has been used to deploy a multi-agent system for the physical rehabilitation of patients affected by a stroke. The involved rehabilitation exercises can be automatically assessed, classified and supervised. The conducted performance tests allow to conclude that the system can provide feedback to the patients in real-time.

Toward Precision Rehabilitation for Neurological Diseases: Data-Driven Approach to Exergame Personalization

- Title: Toward Precision Rehabilitation for Neurological Diseases: Data-Driven Approach to Exergame Personalization [50]
- Authors: Santiago Sánchez, David Vallejo, Carlos González, José Jesús Castro, and Javier Alonso Albusac
- Type: Conference
- Conference: XIII International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI)
- Location: Toledo, Spain
- Year: 2019
- DOI: 10.3390/proceedings2019031010
- Related to the current research topic: No, although the research conducted and the data oriented solutions implemented made it possible to outline the foundations of what would be the data exchange format between the systems developed during the execution of this doctoral dissertation and the immersive visualization devices.
- Related figure(s): 2.12
- Abstract: Physical rehabilitation of patients affected by neurological diseases is currently an unmet clinical need due to the high cost of health systems and the impact that rehabilitation has on patients and their families. This paper introduces an approach based on the idea of precision rehabilitation, where personalization of the physical rehabilitation for each patient and accessibility to technological tools that support it are pursued. A general architecture that contemplates functional modules related to multiple work areas and different roles is proposed. One of these modules is related to gamification and delves into the design and development of exergames, defined from a general data model, in order to increase the level of patient motivation when exercising. The paper discusses the creation and evaluation of an exergame designed for the rehabilitation of lower limbs.

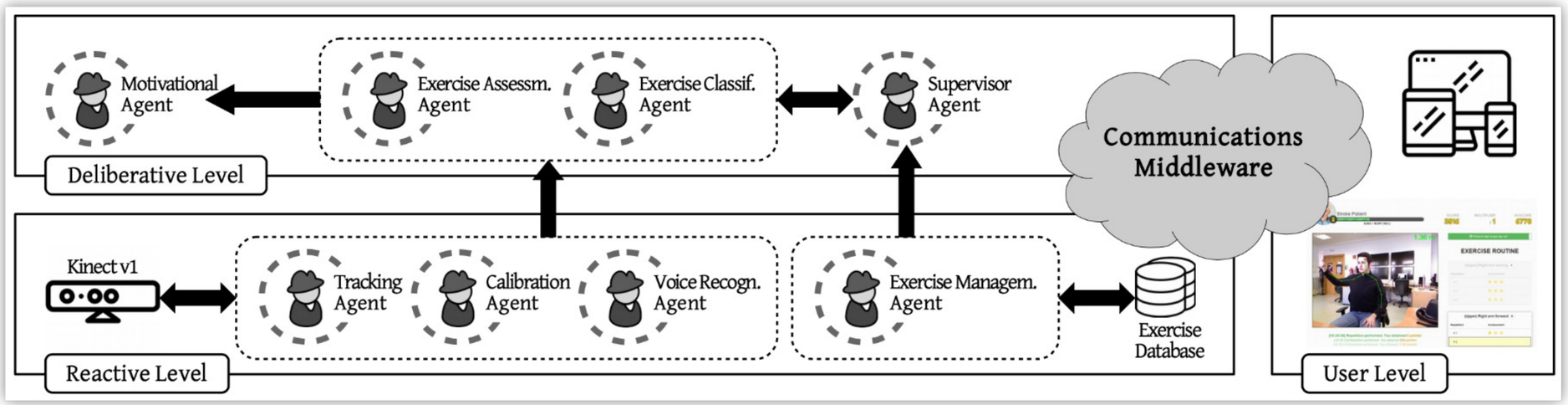


Figure 2.11: Overview of the proposed multi-agent architecture.

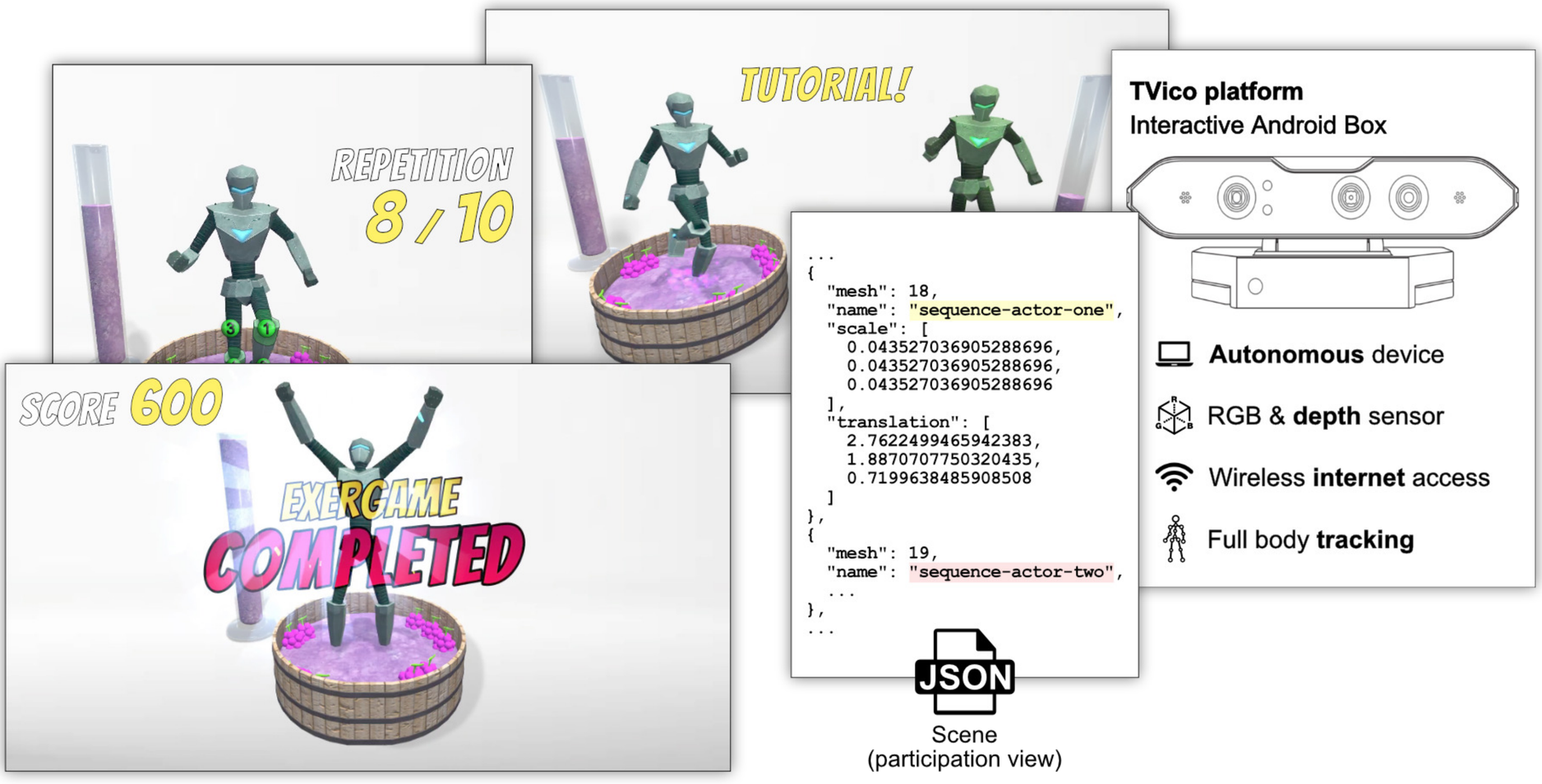


Figure 2.12: Snapshots of the rehabilitation game generated (left) through a data-driven approach (middle) and using a low-cost skeleton tracking device (right).

Aplicación de una metáfora flexible y extensible para la visualización de programas en el contexto del aprendizaje de la programación

- Title: Aplicación de una metáfora flexible y extensible para la visualización de programas en el contexto del aprendizaje de la programación [17]
- Authors: Cristian Gómez, Santiago Sánchez, María de los Ángeles García, Miguel Ángel Redondo, Javier Alonso Albusac, and Manuel Ortega
- Type: Conference
- Conference: XXI International Symposium on Computers in Education (SIIE)
- Location: Tomar, Portugal
- Year: 2019
- ISBN: 978-989-8840-39-4
- Related to the current research topic: Yes.
- Related figure(s): 2.13
- Abstract: El aprendizaje de la programación es un campo de investigación con publicaciones relevantes de más de 25 años. Desde sus inicios, se ha demostrado que su dificultad depende del alto nivel de abstracción que ciertos conceptos de programación son requeridos por los estudiantes. Sin embargo, dicho nivel puede reducirse utilizando representaciones gráficas que motiven a los estudiantes y faciliten su comprensión, asociando elementos del mundo real con términos específicos de programación. Así, este trabajo muestra la notación ANGELA, una metáfora de carreteras y señales de tráfico utilizada para visualizar dinámicamente la ejecución de un programa con un vehículo viajando por su estructura ejecutando sentencias. Dichas visualizaciones se pueden ejecutar en una variedad de entornos, los cuales soportan diferentes mecanismos de interacción dependiendo de la dimensionalidad de las representaciones gráficas. Además, este trabajo destaca la flexibilidad y la extensibilidad de la propuesta mediante su aplicación en diferentes casos de uso, junto con una metodología que hemos seleccionado a modo de ejemplo para mostrar cómo la notación podría ser explotada en un escenario real de aprendizaje.

COLLECE 2.0: Un sistema para el aprendizaje colaborativo de la programación sobre Eclipse, con una metáfora multidimensional para la visualización de programas

- Title: COLLECE 2.0: Un sistema para el aprendizaje colaborativo de la programación sobre Eclipse, con una metáfora multidimensional para la visualización de programas [36]
- Authors: Miguel Ángel Redondo, Santiago Sánchez, Cristian Gómez, Carmen Lacave, Ana Isabel Molina, and Manuel Ortega
- Type: Conference
- Conference: XXVI Jornadas sobre la Enseñanza Universitaria de la Informática (JENUI)
- Location: Valencia, Spain
- Year: 2020
- Related to the current research topic: Yes.
- Related figure(s): 2.14
- Abstract: Computer programming is a complex task and a challenge for novice programmers. There are a wide range of difficulties in understanding programming concepts due to the high level of abstraction required to learn them. In order to address these difficulties, we have developed the COLLECE-2.0 system, a plug-in for the Eclipse platform, which provides a real-time, distributed, collaborative programming environment. Its interface has been designed to enhance aspects related to support for group learning. In addition, our proposal makes a special emphasis on the program visualization, incorporating a set of multidimensional graphic representations based on a metaphor. These representations are applicable to a variety of scenarios that support different interaction mechanisms, depending on the dimensionality of the graphic representations and the devices used for their visualization. This paper describes the main details of the COLLECE-2.0 system and how it can be used in different scenarios by visualizing and interacting with structural aspects of the programs and algorithms.

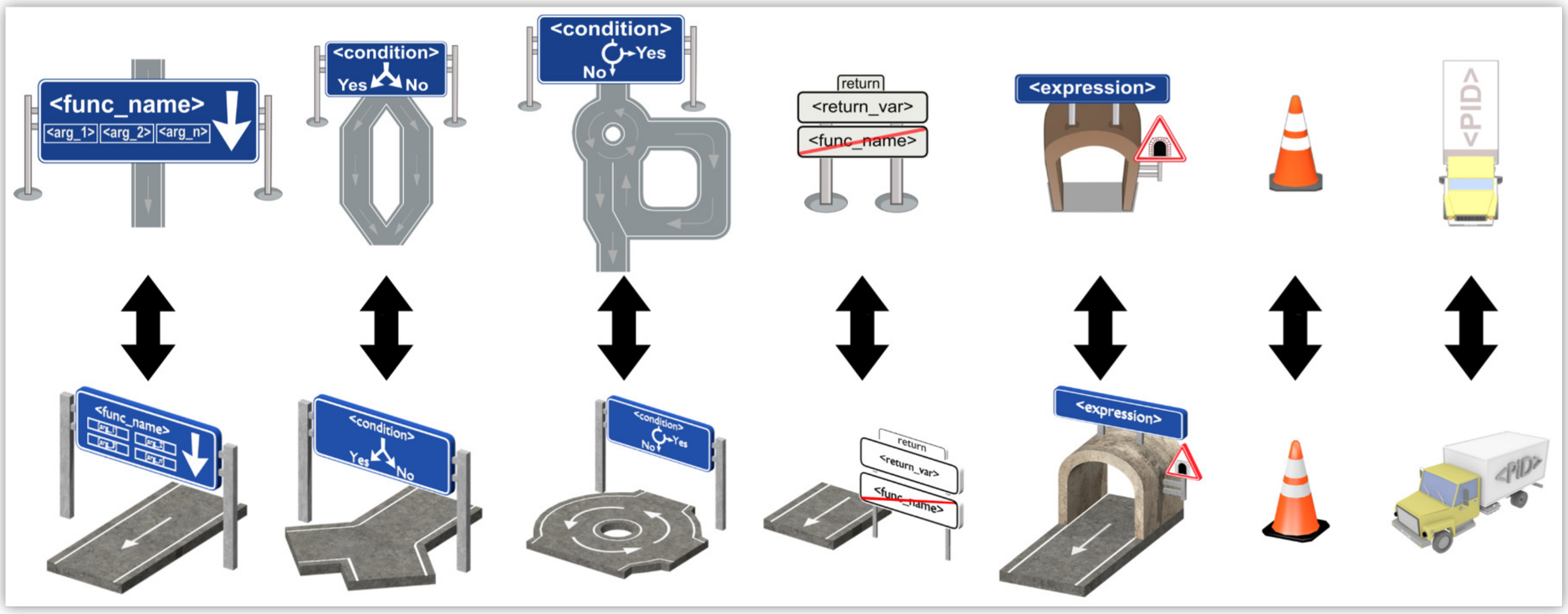


Figure 2.13: Equivalence between the 2-D (top) and 3-D (bottom) ANGELA notation.

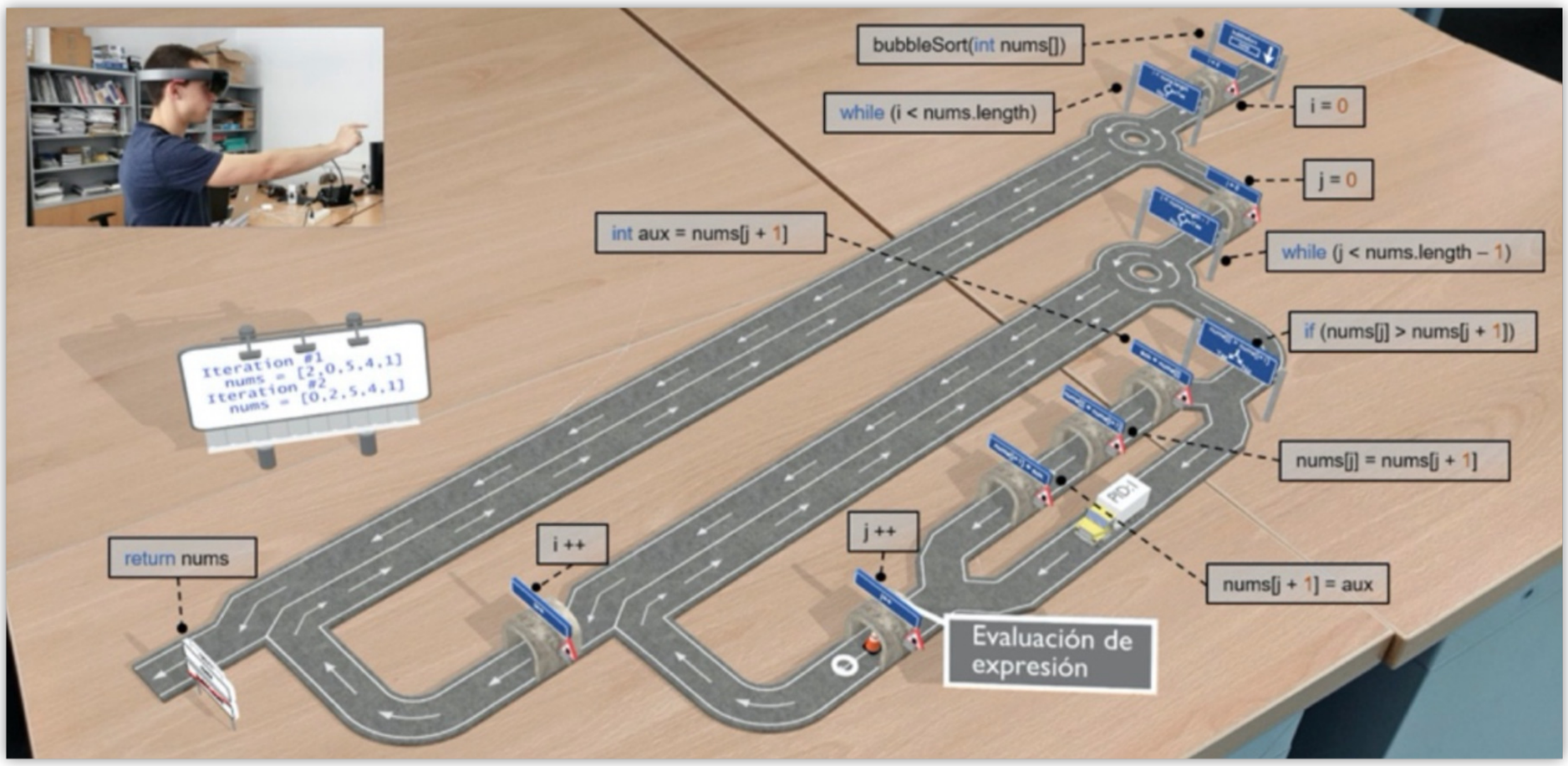


Figure 2.14: Dynamic visualizacion of the bubble sort algorithm as seen through a Mixed Reality device.

3

Conclusions and Future Work

In the present doctoral dissertation, different environments and prototypes have been presented, oriented to facilitate the learning of programming in users with different profiles. The totality of the results has been exposed by means of the set of scientific articles that accompanies this dissertation by compendium of publications, where the proposals and methods followed to reach the results are detailed in depth.

In this chapter an analysis of the work conducted is made, starting from the objectives stated in the section 1.3 and returning to the research questions initially formulated to answer them according to what is proposed in this doctoral dissertation.

3.1 Achievement of objectives

The proposal for this doctoral dissertation arises from the research questions posed in the section 1.3.1 and which are again included here for convenience:

- **RQ**. Does the use of programming learning tools based on emerging technologies improve the overall learning experience?
  - **RQ1**. Does understanding and knowing the proposed tools and how they work intrinsically increase interest in programming?
  - **RQ2**. Does using inmersive interfaces, e.g. AR, improve motivation towards programming?
  - **RQ3.** Generally, what is the subjective perception of the ease of use, usefulness and intention of use of the proposed tools?

To answer these questions, a main objective was set, which consisted in

"*offering a set of tools and prototypes that facilitate the learning of programming and guide the complete formative process of students with different educational profiles through the use of immersive and emerging technologies that exploit new learning paradigms*".

Firstly, all the publications that accompany this doctoral dissertation, and which are already cited in the chapter 2, have presented in detail multiple reviews of the state of the art on lines of research as diverse as the visualization of information, graphic representation of algorithms and programs, collaborative software development environments, immersive technologies and the benefits of learning to program, among others. This has made it possible to achieve the secondary objective **[Obj01] Review of the state of the art**.

The study of the state of the art has led to the development of several solutions aimed at facilitating the learning of programming, such as the platform COLLECE-2.0 aimed at solving programming problems in a collaborative manner, the graphic notation ANGELA for the representation of programs and algorithms, and the serious game RoboTIC to introduce programming concepts in children. Furthermore, these solutions can be used by means of immersive AR devices through the use of 3-D graphics in dedicated learning environments. This has made it possible to achieve the secondary objectives **[Obj02] Development of software systems**, **[Obj03] Programming metaphors and other abstractions**, **[Obj04] Use of immersive technologies** and **[Obj05] Learning environments**.

Also, the different evaluations of the proposed solutions presented throughout the various scientific publications have allowed validation in the context of motivation, ease of use, usefulness and intention of use. This has made it possible to achieve the secondary objective **[Obj06] Proposal evaluation and validation**.

Finally, it is possible to answer affirmatively to the research questions formulated initially thanks to the results exposed in the publications [48] and [51], where evaluations on the proposed solutions are conducted in the scope of universities and primary education. The results of these evaluations confirm that knowing the tools and how they work intrinsically increase interest in programming (**RQ1**), using inmersive interfaces improve motivation of the users toward programming (**RQ2**) and the perception of the solutions by the users in the context of the ease of use, usefulness and intention of use has been positive (**RQ3**). Therefore, the main research question formulated in this doctoral dissertation (**RQ**) is answered affirmatively, confirming that using emerging programming learning tools improves the learning experience of users.

3.2 Future lines of work

The result of this doctoral dissertation establishes a starting point for exploring new lines of future work related to learning of programming. Thus, new experiments and use cases are proposed that allow to continue the research in new lines of work, to consider:

- New evaluations of COLLECE-2.0 and ANGELA that allow the establishment of new hypotheses in the context of learning programming, with more long-term experiments that cover the development of complete courses (experimental group) and allow the comparison of results obtained from courses taught traditionally (control group).

- Extensions of COLLECE-2.0 that allow the definition of new use cases related to visual programming by means of the ANGELA notation, thus opening new lines of research that make possible the construction of applications abstracting the user from knowing any programming language.

- New additions to the ANGELA notation to support the visualization of recursion. Currently, advances have been made in this line of work through proposals and experimental designs that allow for the proper representation of recursive programs.

- Exploiting the augmented 3-D space by deploying the program visualizations and RoboTIC on low-cost mobile devices, thus increasing the number of users who could make use of these solutions and allowing to launch large-scale evaluation experiments.

- New mechanisms for learning programming through the development of cross reality (XR) environments that allow the integration of physical devices whose interaction impacts on virtual elements of the visualizations.

Bibliography

[1] J. C. Almenara and J. Osuna. The educational possibilities of augmented reality. *Journal of New Approaches in Educational Research*, 5(1):44–50, 2016. doi:10.7821/NAER.2016.1.140.

[2] Jorge Bacca-Acosta, Silvia Baldiris, Ramón Fabregat, Sabine Graf, and Dr Kinshuk. Augmented reality trends in education: A systematic review of research and applications. *Educational Technology and Society*, 17(4):133–149, 10 2014.

[3] Yorah Bosse and Marco Aurélio Gerosa. Why is programming so difficult to learn? patterns of difficulties related to programming learning mid-stage. *SIGSOFT Softw. Eng. Notes*, 41(6):1–6, January 2017. doi:10.1145/3011286.3011301.

[4] Crescencio Bravo, Rafael Duque, and Jesús Gallardo. A groupware system to support collaborative programming: Design and experiences. *Journal of Systems and Software*, 86(7):1759–1771, 2013. doi:10.1016/j.jss.2012.08.039.

[5] Neil Burgess, Eleanor A Maguire, and John O'Keefe. The human hippocampus and spatial and episodic memory. *Neuron*, 35(4):625 – 641, 2002. doi:10.1016/S0896-6273(02)00830-9.

[6] M. Chandramouli, M. Zahraee, and C. Winer. 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*, pages 581–586, Milwaukee, WI, USA, 2014. doi:10.1109/EIT.2014.6871829.

[7] A. Deusany de Carvalho. Cooperative live coding as an instructional model. In *2015 Latin American Computing Conference (CLEI)*, pages 1–7, Arequipa, Peru, 2015. doi:10.1109/CLEI.2015.7359464.

[8] Matt Dunleavy and Chris Dede. *Augmented Reality Teaching and Learning*, pages 735–745. Springer New York, New York, NY, 2014. doi:10.1007/978-1-4614-3185-5_59.

[9] Gil Ebel and Mordechai Ben-Ari. Affective effects of program visualization. In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, page 1–5, Canterbury, United Kingdom, 2006. Association for Computing Machinery. doi:10.1145/1151588.1151590.

[10] Stine Ejsing-Duun and Helle Marie Skovbjerg. Gamification of a higher education course: What's the fun in that? pages 92–98, Berlin, Germany, 10 2014. Academic Conferences International Limited.

[11] European Commision. Coding - the 21st century skill. https://ec.europa.eu/digital-single-market/en/coding-21st-century-skill, 2020. [Online; accessed 20-October-2020].

[12] H Fan and C Sun. Achieving integrated consistency maintenance and awareness in real-time collaborative programming environments: The CoEclipse approach. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2012*, pages 94–101, Wuhan, China, 2012. doi:10.1109/CSCWD.2012.6221803.

[13] Hongfei Fan and Chengzheng Sun. Dependency-based automatic locking for semantic conflict prevention in real-time collaborative programming. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, page 737–742, Trento, Italy, 2012. Association for Computing Machinery. doi:10.1145/2245276.2245417.

[14] Pedro Feijóo-García and Fernando De la Rosa. Roblock - web app for programming learning. *International Journal of Emerging Technologies in Learning (iJET)*, 11(12):45–53, 2016. doi:10.3991/ijet.v11i12.6004.

[15] José Figueiredo and Francisco José García-Peñalvo. Building skills in introductory programming. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM'18, page 46–50, Salamanca, Spain, 2018. Association for Computing Machinery. doi:10.1145/3284179.3284190.

[16] Glassdoor, Inc. Glassdoor. best jobs in america 2020. https://www.glassdoor.com/List/Best-Jobs-in-America-LST_KQ0,20.htm, 2020. [Online; accessed 21-October-2020].

[17] Cristian Gmez-Portes, Santiago Schez-Sobrino, María Á. García, Miguel Á. Redondo, Javier A. Albusac, and Manuel Ortega. Aplicación de una metáfora flexible y extensible para la visualización de programas en el contexto del aprendizaje de la programación. In *Proceedings of the 21st International Symposium on Computers in Education (SIIE)*, Tomar, Portugal, 2019. SIIE.

[18] Debra Hensgen, Raphael Finkel, and Udi Manber. Two algorithms for barrier synchronization. *International Journal of Parallel Programming*, 17(1):1–17, 1988.

[19] J. Hidalgo-Céspedes, G. Marín-Raventós, and V. Lara-Villagrán. Learning principles in program visualizations: A systematic literature review. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Erie, PA, USA, 2016. doi:10.1109/FIE.2016.7757692.

[20] Gwo-Jen Hwang, Po-Han Wu, Chi-Chang Chen, and Nien-Ting Tu. Effects of an augmented reality-based educational game on students' learning achievements and attitudes in real-world

observations. *Interactive Learning Environments*, 24(8):1895–1906, 2016. doi:10.1080/10494820.2015.1057747.

[21] Colleen Kehoe, John Stask, and Ashley Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *International Journal of Human-Computer Studies*, 54(2):265 – 284, 2001. doi:10.1006/ijhc.2000.0409.

[22] T. Koschmann. *CSCL: Theory and practice of an emerging paradigm*. Computers, cognition, and work. Lawrence Erlbaum Associates, Inc, Hillsdale, NJ, US, 1996.

[23] C. Lacave, M. A. García, A. I. Molina, S. Sánchez, M. A. Redondo, and M. Ortega. Collece-2.0: A real-time collaborative programming system on eclipse. In *Proceedings of the 21st International Symposium on Computers in Education (SIIE)*, pages 1–6, Tomar, Portugal, 2019. SIIE, IEEE.

[24] L. Layman. Changing students' perceptions: An analysis of the supplementary benefits of collaborative software development. In *19th Conference on Software Engineering Education Training (CSEET'06)*, pages 159–166, Turtle Bay, HI, USA, 2006. doi:10.1109/CSEET.2006.10.

[25] J. Lee. The effects of visual metaphor and cognitive style for mental modeling in a hypermedia-based environment. *Interacting with Computers*, 19(5-6):614–629, 2007. doi:10.1016/j.intcom.2007.05.005.

[26] D. Molero, S. Schez-Sobrino, D. Vallejo, C. Glez-Morcillo, and J. Albusac. A novel approach to learning music and piano based on mixed reality and gamification. *Multimedia Tools and Applications*, 2020. In Press. doi:10.1007/s11042-020-09678-9.

[27] National Center for Education Statistics (NCES). Ipeds completions survey. `https://nces.ed.gov/ipeds/`, 2019. [Online; accessed 20-October-2020].

[28] John T. Nosek. The case for collaborative programming. *Commun. ACM*, 41(3):105–108, March 1998. doi:10.1145/272287.272333.

[29] M. Ortega, M.A. Redondo, A.I. Molina, C. Bravo, C. Lacave, Y. Arroyo, S. Sánchez, M.A. García, C.A. Collazos, J.J. Toledo, H. Luna-García, J.A. Velázquez-Iturbide, and R.A. Gómez-Pastrana. Iprog: Development of immersive systems for the learning of programming. In *ACM International Conference Proceeding Series*, volume Part F131194, Cancun, Mexico, 2017.

[30] Andrew Ortony. Metaphor and thought. 1979.

[31] Martinha Piteira and Carlos Costa. Learning computer programming: Study of difficulties in learning programming. ISDOC '13, page 75–80, Lisboa, Portugal, 2013. Association for Computing Machinery. doi:10.1145/2503859.2503871.

[32] Cristian G. Portes, Carmen Lacave, Ana I. Molina, David Vallejo, and Santiago Sánchez-Sobrino. Personalising exergames for the physical rehabilitation of children affected by spine pain. In *Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 2: ICEIS*, pages 533–543, Online, 2020. INSTICC, SciTePress.

[33] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1):1–24, October 2017. doi:10.1145/3077618.

[34] K. Ramírez-Benavides and L. A. Guerrero. Modebots: Environment for programming robots for children between the ages of 4 and 6. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 10(3):152–159, 2015. doi:10.1109/RITA.2015.2452692.

[35] Miguel Á. Redondo, Santiago Sánchez, Gó Cristian, Carmen Lacave, Ana I. Molina, and Manuel Ortega. Abandono de primer año en la ingeniería informática. In *Actas de las XX Jornadas de Enseñanza Universitaria de la Informática (JENUI)*, page 109–116, Oviedo, Spain, 2014. AENUI, Asociación de Enseñantes Universitarios de la Informática.

[36] Miguel Á. Redondo, Santiago Sánchez, Gómez Cristian, Carmen Lacave, Ana I. Molina, and Manuel Ortega. Collece 2.0: Un sistema para el aprendizaje colaborativo de la programación sobre eclipse, con una metáfora multidimensional para la visualización de programas. In *Actas de las Jornadas sobre Enseñanza Universitaria de la Informática, Vol 5*, page 109–116, Online, 2020. AENUI, Asociación de Enseñantes Universitarios de la Informática.

[37] George G. Robertson, Stuart K. Card, and Jack D. Mackinlay. Information visualization using 3d interactive animation. *Commun. ACM*, 36(4):57–71, April 1993. doi:10.1145/255950.153577.

[38] Stephan Salinger, Christopher Oezbek, Karl Beecher, and Julia Schenk. Saros: An eclipse plug-in for distributed party programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '10, page 48–55, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1833310.1833319.

[39] 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. In *Proceedings of the XI International Conference on Interfaces and Human Computer Interaction*, pages 136–142, Lisbon, Portugal, 2017.

[40] Santiago Sánchez, María Ángeles García, María del Carmen Lacave, Ana Isabel Molina, Carlos González, David Vallejo, and Miguel Ángel Redondo. Applying mixed reality techniques for the visualization of programs and algorithms in a programming learning environment. In

Proceedings of the Tenth International Conference on Mobile, Hybrid, and On-line Learning (eLmL, pages 84–89, Rome, Italy, 2018.

[41] Sunil Sarin and Irene Greif. Computer-based real-time conferencing systems. *Computer*, 18(10):33–45, October 1985. doi:10.1109/MC.1985.1662711.

[42] S. P. Sarkar, B. Sarker, and S. K. A. Hossain. Cross platform interactive programming learning environment for kids with edutainment and gamification. In *2016 19th International Conference on Computer and Information Technology (ICCIT)*, pages 218–222, Dhaka, Bangladesh, 2016. doi:10.1109/ICCITECHN.2016.7860198.

[43] Ronny Scherer, Fazilat Siddiq, and Bárbara Sánchez Viveros. The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, 111(5):764, 2019. doi:10.1037/edu0000314.

[44] S. Schez-Sobrino, D. N. Monekosso, P. Remagnino, D. Vallejo, and C. Glez-Morcillo. Automatic recognition of physical exercises performed by stroke survivors to improve remote rehabilitation. In *Proceedings of the 2019 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, pages 1–6, Ho Chi Minh City, Vietnam, 2019. doi:10.1109/MAPR.2019.8743535.

[45] S. Schez-Sobrino, D. Vallejo, D. N. Monekosso, C. Glez-Morcillo, and P. Remagnino. A distributed gamified system based on automatic assessment of physical exercises to promote remote physical rehabilitation. *IEEE Access*, 8:91424–91434, 2020. doi:10.1109/ACCESS.2020.2995119.

[46] Santiago Schez-Sobrino, María Á. García, Cristian Gómez, Carlos Glez-Morcillo, David Vallejo, Javier A. Albusac, and Miguel Á. Redondo. Propuesta y evolución multidimensional de una metáfora visual para facilitar el aprendizaje de la programación. In *Proceedings of the XX International Conference on Human-Computer Interaction*, Donostia, Spain, 2019. AIPO.

[47] Santiago Schez-Sobrino, María Á. García, Cristian Gómez, David Vallejo, Ana I. Molina, Carmen Lacave, Carlos Glez-Morcillo, Javier A. Albusac, and Miguel Á. Redondo. Angela: A novel approach of graphic notation based on the metaphor of road signs to facilitate the learning of programming. In *Proceedings of the 7th International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'19)*, page 822–829, León, Spain. ACM. doi:10.1145/3362789.3362871.

[48] Santiago Schez-Sobrino, María Á. García, Carmen Lacave, Ana I. Molina, Carlos Glez-Morcillo, David Vallejo, and Miguel Á. Redondo. A modern approach to supporting program visualization: from a 2d notation to 3d representations using augmented reality. *Multimedia Tools and Applications*, 2020. In Press. doi:10.1007/s11042-020-09611-0.

[49] Santiago Schez-Sobrino, Cristian Gmez-Portes, David Vallejo, Carlos Glez-Morcillo, and Miguel Á. Redondo. An intelligent tutoring system to facilitate the learning of programming through the usage of dynamic graphic visualizations. *Applied Sciences*, 10(4):1518–1531, Feb 2020. doi:10.3390/app10041518.

[50] Santiago Schez-Sobrino, David Vallejo, Carlos Glez-Morcillo, Jose Jesus Castro-Schez, and Javier Albusac. Toward precision rehabilitation for neurological diseases: Data-driven approach to exergame personalization. In *Proceedings of 13th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI)*, volume 31, page 10, Toledo, Spain, Nov 2019. MDPI AG. doi:10.3390/proceedings2019031010.

[51] Santiago Schez-Sobrino, David Vallejo, Carlos Glez-Morcillo, Miguel Á. Redondo, and José Jesús Castro-Schez. Robotic: A serious game based on augmented reality for learning programming. *Multimedia Tools and Applications*, 2020. In Press. doi:10.1007/s11042-020-09202-z.

[52] Haifeng Shen and Chengzheng Sun. RECIPE: a prototype for Internet-based real-time collaborative programming. In *Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems*, Philadelphia, Pennsylvania, USA, 2000. Citeseer.

[53] Simon. Soloway's rainfall problem has become harder. In *Proceedings of the 2013 Learning and Teaching in Computing and Engineering*, LATICE '13, page 130–135, USA, 2013. IEEE Computer Society. doi:10.1109/LaTiCE.2013.44.

[54] Donna Teague and Paul Roe. Collaborative learning: Towards a solution for novice programmers. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, page 147–153, AUS, 2008. Australian Computer Society, Inc.

[55] Jaime Urquiza-Fuentes and J. Angel Velázquez-Iturbide. A long-term evaluation of educational animations of functional programs. In *Proceedings of the 12th International Conference on Advanced Learning Technologies*, ICALT '12, page 26–30, Rome, Italy, 2012. IEEE Computer Society. doi:10.1109/ICALT.2012.50.

[56] Jaime Urquiza-Fuentes and J. Ángel Velázquez-Iturbide. Toward the effective use of educational program animations: The roles of student's engagement and topic complexity. *Computers & Education*, 67:178 – 192, 2013. doi:10.1016/j.compedu.2013.02.013.

[57] U.S. Bureau of Labor Statistics. Employment by detailed occupation. `https://www.bls.gov/emp/tables/emp-by-detailed-occupation.htm`, 2019. [Online; accessed 20-October-2020].

[58] D. Vallejo, S. Schez-Sobrino, J. Albusac, J. J. Castro-Schez, and C. Glez-Morcillo. An agent-based approach to physical rehabilitation of patients affected by neurological diseases. In *Proceedings of the 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019)*, volume 160, pages 346–353, Coimbra, Portugal, 2019. doi:10.1016/j.procs.2019.11.081.

[59] J. Á. Velázquez-Iturbide. Towards an analysis of computational thinking. In *2018 International Symposium on Computers in Education (SIIE)*, pages 1–6, Jerez, Spain, 2018. doi:10.1109/SIIE.2018.8586710.

[60] J. Á. Velázquez-Iturbide, I. Hernán-Losada, and M. Paredes-Velasco. Evaluating the effect of program visualization on student motivation. *IEEE Transactions on Education*, 60(3):238–245, 2017. doi:10.1109/TE.2017.2648781.

[61] L. Williams. Pair programming. In *Encyclopedia of Software Engineering*, pages 651–659, 2010.

[62] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, March 2006. doi:10.1145/1118178.1118215.

[63] Gary Ka-Wai Wong and Ho-Yin Cheung. Exploring children's perceptions of developing twenty-first century skills through computational thinking and programming. *Interactive Learning Environments*, 28(4):438–450, 2020. doi:10.1080/10494820.2018.1534245.

[64] Min Xu, Jeanne M David, Suk Hi Kim, et al. The fourth industrial revolution: Opportunities and challenges. *International journal of financial research*, 9(2):90–95, 2018. doi:10.5430/ijfr.v9n2p90.

This document was edited with *Emacs*
and formatted with LaTeX en *macOS*

Ciudad Real, 27th November 2020