A visual editing tool supporting the production of 3D interactive graphics assets for public exhibitions

A Visual Editing Tool Supporting the Production of 3D Interactive Graphics Assets for Public Exhibitions

Alberto Cannavò, Francesco De Pace, Federico Salaroglio, Fabrizio Lamberti*

Dipartimento di Automatica e Informatica of Politecnico di Torino, 10129 Torino, Italy (e-mail: firstname.lastname@polito.it)

Abstract

The introduction of interactive assets in public exhibitions is capable to significantly enhance the visitors' user experience. However, the creation of interactive applications could represent a challenging task, especially for users lacking computer skills. Visual programming languages (VPLs) – one of the instruments belonging to the broad categories of methods and tools devised to support end-user development (EUD) – promise to offer an intuitive way to overcome these limitations, by providing easy-to-use and efficient interfaces for encoding applications' logic. Moving from these considerations, this paper first analyses pros and cons of tools devised so far to support the generation of interactive contents. Then, it presents the design of a new tool named Visual Scene Editor (VSE), which allows users with little to no programming skills to create 3D interactive applications by combining available assets through an interactive, visual process. Both objective and subjective measurements have been collected with both skilled and unskilled users to evaluate the performance of the proposed tool. A comparison with existing solutions shows a reduction in the time required to complete the assigned tasks, of the complexity of the logic created, as well as of the number of errors made, confirming the suitability of the VSE for the said purpose.

Keywords: End-user development (EUD), visual programming languages (VPLs), interactive assets, 3D graphics, virtual and augmented reality, human-machine interaction, natural user interfaces (NUIs)

*Corresponding author

1. Introduction

Today, computer graphics assets are widely used in various application domains, encompassing movie and video-game production (Zyda (2005); Schmalstieg and Stork (2019); Lamberti et al. (in press)), manufacturing (Polvi et al. (2018); De Pace et al. (2019)), data science (Samsel et al. (2018)), etc.

A particularly promising field is represented by interactive applications which are designed to improve the users' experience in public exhibitions (Sanna et al. (2016); Wojciechowski et al. (2004)). Several museums, like the Virtual Archaeological Museum (MAV) in Italy[1], the National Museum of Zurich in Switzerland[2], the National Museum of Singapore[3], and the Cleveland Museum of Art[4] in the USA, among others, are already taking advantage of new technologies to present interactive experiences to their visitors.

Similar solutions are being explored also, e.g., for marketing purposes, in trade fairs (Yoon et al. (2015)), shopping areas (Jo and Kim (2019); Liu (2014)), etc. Examples are, for instance, the virtual reality experience developed by Mastercard and Swarovski for the Atelier Swarovski in the USA[5], the augmented reality app developed by IKEA[6], the virtual tour designed to visualize the new cabins of All Nippon Airways planes[7], the immersive test drive simulation developed for the Volvo XC90[8], etc.

The idea is to make the experience more engaging and enjoyable, by replacing static contents with virtual assets that users can interact with through natural user interfaces (NUIs). Reconstructed historical landmarks, ancient or modern artworks, commercial products and any other item can be visualized and manipulated through projected holograms/walls, virtual/augmented reality systems, tangible/gestural interfaces, etc.

The adoption of interactive technologies is proven to be capable to enhance understanding, giving more people access to knowledge (Bekele et al.

[1] MAV: https://www.museomav.it/museum/?lang=en

[2] Ideas of Switzerland, National Museum of Zurich: https://www.landesmuseum.ch/ideas-of-switzerland

[3] Story of the forest, National Museum of Singapore: https://www.nationalmuseum.sg/our-exhibitions/exhibition-list/story-of-the-forest

[4] ARTLENS, Cleveland Museum of Art: http://www.clevelandart.org/artlens-gallery

[5] Atelier Swarovski: https://mstr.cd/2wZcAce

[6] IKEA's Place app: https://bit.ly/32cbghp

[7] All Nippon Airways's Aeronautics VR: https://mbryonic.com/portfolio/ana/

[8] Volvo's XC90 Test drive VR: http://framestorevr.com/volvo2

(2018)). Moreover, interactive exhibitions can increase the interest of visitors, customers, etc., bringing them back to the museum or the shop several times (Alexander et al. (2013)). Lastly, in Jeon et al. (2019) it was shown that interactive art and interactive experiences, in general, share a number of aspects with the field of human-computer interaction, like creativity, embodiment, affect, and presence. This strict connection ensures that improvements made to one of these fields is also reflected in the other.

However, the design and implementation of interactive applications is a very time-consuming and skill-intensive task, which requires significant computer skills (Sanna et al. (2016); Wojciechowski et al. (2004)). For this reason, the availability of tools able to ease the development steps, independently of the particular application domain as well as of the device chosen for visualization and interaction is extremely important.

Activities in the above direction are generally classified in the broad area of end-user development (EUD). EUD aims at making end-users able to create and manage complex systems/interfaces without the need to possess professional software development skills (Barricelli et al. (2019)). EUD encompasses various methods, techniques and tools, from co-design and simultaneous editing applications (Serim et al. (2015)) to Wiki and Web mashup environments (Aghaee and Pautasso (2014); Ardito et al. (2012)), to name a few. For instance, concerning the graphics domain, Kim and Yoon (2005) exploited EUD approaches to develop a case-based design scheme supporting designers in reusing prior experience. In (Lee et al. (2010)), the authors presented a Web-based editor that allows users to effectively design new interfaces by combining available templates. More recently, in (Swearngin et al. (2018)) a tool was proposed to reconstruct vector graphics from images and let designers rapidly edit them. A review of the most representative works in the EUD domain can be found in Maceli (2017).

Among EUD solutions, a significant role is played by end-user programming (EUP) techniques, which are meant to let users create their own programs by using programming paradigms suited to their skills (Burnett and Scaffidi (2014)). EUP comprises different approaches, such programming by example (Billard et al. (2008)) natural language programming (Zhan and Hsiao (2018)), visual programming (Barricelli et al. (2019)), etc.

Visual programming languages (VPLs), in particular, have increasingly attracted the attention of both researchers and developers (Paternò (2013)). These languages, which are widely used for educational purposes (Broll et al. (2017); Ventura et al. (2015)), remove the need for writing blocks of code us-

ing a given syntax by replacing text with a visual representation. Concepts that are typical of traditional programming languages (like variables, functions, etc.) are replaced, e.g., by colored blocks which can be plugged each other or connected by drawing arrows through an intuitive interface in order to define the intended application's logic. Works in the literature have shown the benefits of VPLs in terms of users' engagement, satisfaction, motivation and performance (Broll et al. (2017); Pinto-Llorente et al. (2018)).

Moving from these considerations, this paper presents a tool, called Visual Scene Editor (VSE), which allows users with limited or no programming skills (later referred to also as unskilled users, for brevity) to create interactive 3D applications by adding to available graphics assets the required behaviors and interaction capabilities. Newly generated interactive contents can then be visualized, in principle, on any traditional computer or large display, as well as in immersive virtual reality environments, on augmented reality headsets, etc. Visitors of the exhibition may interact with the generated contents by using both traditional interfaces (mouse and keyboard) or some kind of NUIs, e.g., based hand/body gestures, voice, gaze, etc. In the current implementation, computer and holographic displays were considered, together with hand gesture-based interactions.

The design of the VSE builds upon a previous experience with a VPL-based tool developed for the considered purpose, named Leap Embedder (LE) and presented in Sanna et al. (2016). The tool was designed with the aim to simplify the usage of the Blender Game Engine (BGE)[9], a real-time environment which has been already exploited to create interactive 3D applications in various domains, including cultural heritage (Herrmann and Pastorelli (2014); Bustillo et al. (2015)), production control (Lind and Skavhaug (2012)), molecular modeling (Waldon et al. (2014)), etc. The BGE natively provides users with an integrated VPL, which allows them to define the connections between events recognized by the system (e.g., an interaction on a virtual asset) and actions to perform within the graphics environment (e.g., in response to that interaction).

The LE was meant, at the same time, to ease the original graphics notation of the BGE and to let the users easily integrate hand gesture-based interactions (gathered through the Leap Motion sensor[10], hence the name

[9]BGE: https://docs.blender.org/manual/en/latest/game_engine/

[10]Leap Motion sensor: https://www.leapmotion.com/

of the tool). Despite innovations that were introduced in the LE, it still maintained the native *object-centric* paradigm used in the BGE (and other tools) to define the application's logic, based on so called "logic bricks" that are attached to individual interactable objects communicating via message passing. This choice limited the improvement in terms of usability that could be guaranteed by the new interface and the simplified notation.

The VSE tool that is introduced in this paper represents an evolution of the LE. In fact, it leverages the lessons learned with that experience and integrates them with hints coming from the literature related to the use of EUD approaches and, specifically, of VPLs in various application contexts. The result is a different interaction paradigm, in the following referred to as *scene-centric* (hence, the acronym), which proved to be capable to support the creation of 3D interactive applications achieving better performance with respect to both the LE and BGE. It is worth observing that, although the paper's focus is on public exhibitions, interactive applications created with the VSE (as well as with the other tools mentioned in the paper) can be targeted in principle to any other domain, especially when required application-development skills are missing.

The remaining of the paper is organized as follows. Section 2 reviews relevant works found in the literature pertaining interactive applications for public exhibitions, EUD and VPLs, by also discussing the main features of the tools mentioned above. Section 3 introduces the VSE, by providing details on the design and implementation of its interface and notation. Section 4 shows how to practically use the VSE for creating an interactive application for a possible exhibition. Section 5 and Section 6 illustrate the setup used in the experimental evaluation and analyze the results, respectively. Lastly, Section 7 concludes the paper, by suggesting possible directions that are worth further investigation.

2. Related Work

In this section, the use of interactive applications in public exhibitions is briefly discussed. Afterwards, starting from the literature concerning EUD applications, the state of the art related to the use of VPLs in the EUP domain is presented and analyzed. Finally, details concerning the LE (and the BGE) are provided.

2.1. Applications for Public Exhibitions

Several works demonstrated already how interactive applications can be exploited to boost the effectiveness of public exhibitions.

For instance, works like (Song et al. (2004)) and (Rubino et al. (2015)) focused on museum visits. In (Song et al. (2004)), a virtual environment was developed to allow users to virtually visit a reconstructed heritage scenario. The experience was enhanced by means of a virtual tour guide which was capable to present different information depending on the actual visitor's interests. In (Rubino et al. (2015)), the authors presented an augmented reality location-based mobile game to support the exploration of a real museum. Users were requested to earn as many points as possible by completing a set of objectives (e.g., by physically reaching some specific points of interest or by solving small quizzes). Although adults seemed less enticed in using the application due to the adopted cartoon-style interface, results showed that the proposed system provided a compelling learning experience, proving to be more effective than a traditional portable audiovisual guide.

Other works focused on outdoor settings. For instance, Christou et al. (2006) and Cassidy et al. (2019) presented two different solutions to improve the fruition and enjoyment of archaeological sites. In particular, in (Christou et al. (2006)), a CAVE-like system was proposed to enable virtual visits to an ancient Greek temple. Besides the 3D visualization, immersive sounds and haptic interactions were exploited to improve the user experience. In (Cassidy et al. (2019)), an immersive virtual reality system was proposed to support interaction with archaeological remains.

Works presented so far demonstrated the potential of using these new technologies to facilitate learning and boost the experience with public exhibitions targeted, e.g., to cultural heritage. However, as said, the growing diffusion of these alternative ways to communicate information is posing new challenges to the research community concerning the process of developing the required interactive applications.

The above aspect is confirmed by various works in the literature which proposed solutions to help developers create, manipulate and validate interactive exhibitions. For instance, in (Wojciechowski et al. (2004)), a tool targeted to the museum staff or the curators of an exhibition aiming to simplify the creation of Web-based as well as of virtual and augmented reality-based cultural heritage applications based on predefined templates was proposed. Only a few visualization templates were available, namely, a tree-based one with metadata, and a virtual gallery walkthrough-based one with on-request

visualization of artwork details. The authoring interface only let the user select which elements to include in the exhibition and where to position them (in the latter visualization). Other interactions that could be required in (slightly) more complex applications serving the purpose were not supported. Ibrahim and Ali (2018) developed a set of guidelines to be considered in the design of virtual environments supporting cultural learning scenarios. A similar problem was tackled in (Andreoli et al. (2018)), where the design steps to follow during the whole development cycle of a serious game for cultural heritage is presented. Other works focused on investigating which are the attributes that can affect the user experience, e.g., in an interactive virtual reality-based showroom (Yoon et al. (2015)), on identifying the set of senses to be stimulated in a multi-sensory art exhibition (Vi et al. (2017)), on how to reduce new phenomena like cybersickness in public, virtual environments (Liu (2014)), on how visitors move (Loke and Robertson (2009)) or behave (Nakanishi (2004); Gault et al. (2015)) in such spaces, etc.

2.2. End-User Development

EUD techniques have been effectively employed in a number of domains. Many applications have been developed, for instance, in the field of smart environments and Internet of Things (IoT), where a huge amount of devices (and generated data) have to be handled (Tetteroo et al. (2015)). As a matter of example, in (Desolda et al. (2017)), a set of tools that enable users with no programming skills to manage and customize smart home spaces is presented. Different wizard-based interfaces easing the definition of rules controlling the activation of smart devices have been compared and evaluated by both technical and non-technical end-users. Ghiani et al. (2017) proposed a similar environment to customize the behavior of IoT devices which can be applied to different use cases. Experiments indicated that end-users appreciated the devised environment, especially the possibility to have a preview of the effect of rules being created.

The cultural heritage context adopted EUD techniques to let domain experts directly intervene in the creation of interactive exhibitions. The assumption is that their strong involvement should help to guarantee the quality of the generated output (McDermott et al. (2014)). For instance, Ardito et al. (2018) presented a visual composition paradigm letting end-users with no computer skills to program smart devices used in personalized tours. Experiments carried out in the archaeological domain showed that experts were able to create the intended experiences, but also recorded moderate values

of mental effort which could have been caused by the need to continuously switch among different windows to define the application logic. In (Ghiani et al. (2009)), EUD tools were exploited to provide end-users with the ability to edit the main information of a museum guide, as well as to create interactive experiences and games with museum digital assets by arranging museum virtual rooms, managing RFID localized artworks, etc.

EUD methods have been also applied to the creation of general-purpose video-games. For instance, in (Ioannidou et al. (2009)), an EUD tool was designed to let 10-year-old children easily develop interactive video-games using an agent-based framework. Menestrina and De Angeli (2017) proposed an EUD framework to let end-users with no programming skills to model the behavior of Non-Player Characters, whereas Protopsaltis et al. (2011) presented a solution to support the retargeting of existing serious games to other domains.

There are also interesting applications of EUD to the programming of autonomous systems. As a matter of example, Leonardi et al. (2019) introduced an EUD environment to control a humanoid robot. By means of a simplified rule-based notation, end-users can easily program complex robot actions leveraging available sensors and actuators without knowing the underlying technical details.

EUD methods have been exploited also in the field of mobile application development. For instance, in (Danado and Paternò (2014)), the authors proposed an EUD environment to create mobile applications on handheld devices. Puzzle metaphors and drag-and-drop interactions greatly helped users to work on small-size devices, hiding unnecessary complexity. A similar work is presented in (Francese et al. (2017)), where EUD tools leveraging simple drag-and-drop interactions were exploited to combine available services and develop complex pervasive applications. Results showed that the proposed paradigm was judged as intuitive and easy to manage, although concerns were raised about element identification methods.

Finally, EUD techniques have been applied to the creation of Web sites and Web applications. In (Kumar et al. (2011)), a tool aimed to simplify the production of Web pages based on provided examples is presented. The tool leverages human-generated training data to automatically retarget contents for new pages. A similar approach based on templates is reported in (Kim (2010)). In (Valderas et al. (2006)), an ontology-based strategy is designed to ease the development of Web applications by generating functional prototypes from formal specifications.

2.3. Visual Programming Languages

As said, in the context of EUD, a relevant role is played by VPLs. A VPL is a high-level language that allow users to create software programs and other kinds of computer-based artifacts by means of visual graphic elements, without the need to use a canonical text-based syntax (Jost et al. (2014)).

One of the first acknowledged VPLs dates back to the sixties, when Sutherland at MIT developed a system named Sketchpad to support Computer Aided Drafting (CAD) (Sutherland (1964)). The system was meant to let users generate 2D contents by moving an optically tracked probe on a computer screen. Some years later, Ellis et al. (1969) developed an ancestor of flowchart-based VPLs. Like in the prototype by Sutherland, this system allowed users to define flowcharts by employing a tracked probe, and resulting diagrams were displayed on a computer screen.

Thanks to technological progresses, from the preliminary examples above VPLs rapidly improved their efficiency and effectiveness, passing from managing a few simple graphics primitives (like lines, circles, etc.) to several distinct complex shapes, thus providing users with the ability to create richer representations and to deal with larger problems.

This evolution contributed at extending the range of possible application scenarios for VPLs, which started to be used also for education purposes, e.g., to develop problem-solving skills and make programming more accessible. Thus, for instance, in (Kato and Tominaga (2009)), the Lego Mindstorms' VPL-based environment named NXT Software[11] was employed to teach mechanics and robotics concepts. In (Pinto-Llorente et al. (2018)), another educational environment by Lego named WeDo[12] was used to teach younger students the basics of computer programming. Experiments confirmed that students learned more concepts with that tool than in a traditional lesson. In (Ventura et al. (2015)), a gaming environment named CodeCraft was proposed to teach the fundamentals of coding. The environment fosters a problem-based learning approach, which requires the users to solve a series of puzzle games involving 3D virtual elements by using a VPL. In (Broll et al. (2017)), another VPL-based framework was developed to teach distributed programming concepts. The framework allows users to invoke code executed remotely on a different machine, by sending messages with structured data

[11] NXT: https://education.lego.com/en-us/middle-school/intro/mindstorms-ev3
[12] WeDo: https://education.lego.com/en-us/elementary/shop/wedo-2

payload. Experiments carried out with users with no programming skills confirmed the effectiveness of the framework, as they were able to develop simple but functioning applications.

In this paper, the interest is especially on the application of VPLs to EUP. Thus, for instance, in (Weintrop et al. (2018)), a VPL-based paradigm for controling a robotic arm is presented. The interface allows end-users to program the robot by dragging-and-dropping predefined routines and combining them in a single sequence of instructions. The VPL was compared with two commercial programming environments (by ABB and Universal Robots). Results indicated that the proposed VPL allowed users to program the robot faster than with the other environments, keeping the same level of accuracy. Concerning public exhibitions, Stratton et al. (2017) presented a VPL-based tool to generate engaging exhibits by using visual blocks. The tool exploits an event-based approach combined with pluggable, colored blocks that act as callback. The proposed tool was evaluated in a cultural heritage scenario, in which experts had to create a Web-based application letting visitors interact with 2D contents through simple hand gestures.

2.4. Leap Embedder

The literature reviewed above shows that EUD applications encompass a number of heterogeneous domains. However, it seems that a field that has received yet only little attention is that concerning the creation of 3D interactive applications targeted to public exhibitions.

Tools devised so far to address this specific domain either do not support 3D elements, or only manage a few predefined application configurations. Indeed, general-purpose tools devised, e.g., for 3D video-game development could be exploited to this purpose, but they may be too complex to use by unskilled users (users could be required to directly handle 3D geometries, manage lighting and cameras, etc.). Alternatively, special-purpose tools targeting other domains could be considered, but contents and interactions required in the application of interest may not be supported.

To the best of the authors' knowledge, one of the few tools that was developed to fill this gap is the LE (Sanna et al. (2016)). According to the authors, the design of the LE tool started from considering that the development of a 3D application relies on contents that are produced by using modeling and animation tools. Among the various alternatives, the

authors found that there was at least one graphics suite, namely Blender[13], which also integrated a real-time physics engine (the BGE) that could be used to develop 3D interactive applications and games by working with created or imported contents. With the BGE, application logic can be created using an event-driven, VPL-based syntax that relies on three different types of logic bricks, namely "sensors", "controllers" and "actuators" ("wired" together). Logic is "attached" to game objects, which can communicate by exchanging "messages". Users can also define custom, scripted controllers using Python, if needed.

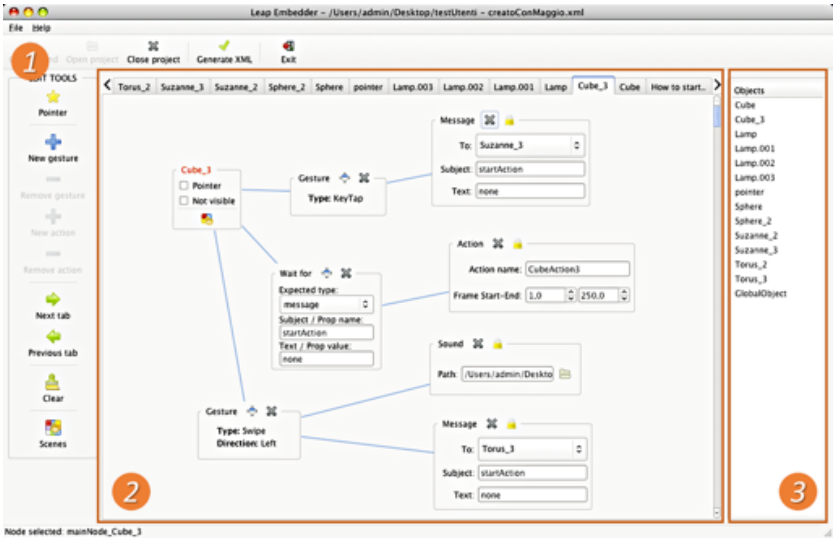
The basic idea behind LE was to replace the above notation with another visual syntax encompassing a reduced, redesigned set of logic bricks, simply referred to as "blocks". Since the main application domain envisaged for produced contents was represented by public exhibitions, LE was also designed to ease the integration of NUI-based interaction modalities (hand gestures were managed, in particular). Application logic created with the stand-alone LE tool can then be imported in the BGE for execution.

The Graphics User Interface (GUI) of the LE is shown in Fig. 1 (a high-resolution version of this figure is available for download[14]). On the left and top sides (1), the general settings are shown. The center area (2) consists of several tabbed panels: each panel corresponds to an object, and it allows the user to define object's behavior by combining blocks. Finally, on the right side (3), the list of available 3D objects is shown. Like in the BGE, a per-object visual organization of the logic is adopted. Event sensing is performed with so called "Wait for" blocks and "Gesture" blocks: the former block type replaces all the BGE's sensors, which are managed by means of a selector (together with associated parameters) directly in the block widget, whereas the latter block type is meant to support the definition of user interaction. Controllers are removed: support for scripted logic is not provided, given the target represented by users lacking programming skills. Actuators are those of the BGE. However, since communication with several different objects is very common in interactive applications, the "Message" actuator was redesigned to support multiple destinations for a given message (still requesting user configuration).

In Sanna et al. (2016), a comparison between the LE and the BGE was

[13]Blender: https://www.blender.org/

[14]High-resolution version of Fig. 1: http://tiny.cc/gqoqmz



(a)

Figure 1: GUI of the LE.

performed by involving three user categories, i.e., beginner, intermediate and skilled (based on their previous knowledge of Blender). Experimental results obtained in creating an interactive application with the two tools showed that beginner and intermediate users where not able to complete the task with the BGE, which was found to use a very sophisticated notation and interaction paradigm for defining the application logic. Skilled users were able to complete the task with both the tools, but were much faster with the LE than with the BGE. With the LE, intermediate and beginner users were five to 10 times slower than skilled users.

3. Visual Scene Editor

This section illustrates the main limitations of LE and introduces the solutions that have been devised to cope with them. Afterwards, the architecture of the VSE tool and its GUI are discussed in detail, by describing the process followed in designing and implementing them.

3.1. LE Limitations and Proposed Solutions

Two experts in the computer interaction field with years of experience in using 3D computer graphics software (like Blender, Autodesk's Maya[15], etc.) and video-game/interactive application development environments (like Unity, Unreal Engine, etc.) were asked to evaluate the LE.

They replicated the same tasks proposed in (Sanna et al. (2016)), with the aim to identify which aspects of the LE could be enhanced to improve the performance of unskilled users and the usability of the tool. A significant difficulty in finding/distinguishing elements used in the definition of the application logic was lamented. Concerns were also raised on the amount of parameters which had to be configured in each block (basically moving the original complexity of the BGE there). In fact, although the LE design simplified inter-object (especially one-to-many) communications, the adopted paradigm is still based on messages explicitly defined by the users. Lastly, and most importantly, the LE's object-centric approach was criticized; the effect of using this approach is not only that the logic is distributed onto single objects (which could be regarded as a plus from a programmer's perspective), but that also visualization is "centered" (like in the BGE) onto selected object. Thus, users are not allowed to observe the behavior of multiple objects at a time, making it difficult to define and monitor relations among them.

Considering the above comments, three areas requiring intervention were identified, namely, *visual elements identification*, *inter-object communication*, and *relations visualization*.

The design of the VSE moved from these considerations, which led to the creation of a VPL-based tool exploiting a scene-centric approach, in which the users are provided, at development time, with a high-level view of an entire "scene". A scene is intended as a container of 3D elements that are visible and users can possibly interact with at a certain time when the application is executed. These elements are shown all together in the tool's GUI, so that connections and interactions among them can always be immediately spotted. Moreover, the LE message-based communication mechanism was simplified, introducing an event-based approach that lets the users define behaviors involving multiple objects without having to manually specify all the possible parameters. Finally, taking into account also hints coming from

[15] Autodesk Maya: https://www.autodesk.com/products/maya/overview

the review of the state of the art, a color-based scheme has been implemented to simplify element identification, a multi-panel interface has been adopted, and a preview functionality has been developed. More details on how the above solutions have been implemented are provided in the following.

3.1.1. Visual Elements Identification

In order to improve the recognizability of the visual elements, VSE draws inspiration from VPL-based tools that rely on visual-blocks. Such tools use shapes and colors to make the users capable to uniquely identify user interface's elements and logic's components. Shapes usually provide text fields that describe their function, whereas colors are used to cluster the shapes depending on their purpose. Moreover, they usually provide a layout that helps users to cluster similar interface's elements, by using panels and windows.

Many shape-color based tools exist, such as miniBloq[16], ToonTalk (Kahn (1995)), Snap![17], etc. Among them, Scratch (Maloney et al. (2010)) has indeed drawn most of the attention. Scratch was designed to assist primary school students to learn computer programming by creating 2D graphics programs.

Scratch's notation, which has been exploited also for the creation of interactive exhibits (Stratton et al. (2017)), relies on several distinct colored blocks, which can be combined in an indented "script" to generate a functional program. Each block represents a specific operation, and the shape of the block indicates which are the other blocks it can be connected to. Blocks are divided in several "categories", each represented by a specific color which allows them to be clearly identified by the users. The Scratch's notation and the panel-based organization of its GUI proved to be particularly effective. For instance, in (Ouahbi et al. (2015)), a comparison with a traditional programming environment exploiting a text-based language was performed. Results indicated that unskilled users preferred to create programs with Scratch, showing interest in continuing to develop with it. Similar findings were obtained by Sáez-López et al. (2016), whose results showed benefits in using Scratch for both learning coding concepts and developing computational thinking abilities.

Thus, inspired by design choices above, the VSE's GUI has been split

[16]miniBloq: http://blog.minibloq.org/
[17]Snap!: https://snap.berkeley.edu/

in panels, each providing different functionalities. Users can visualize all the panels at once, without having to switch among different windows or tabs (as in the LE or other tools, like, e.g., that in (Ghiani et al. (2017))). Furthermore, as it will be shown in detail in the following, colors and indentation have been used to improve the readability of the application's logic elements, helping users to visually assign objects to scene and easily recognize scene-to-object relations.

3.1.2. Inter-object Communication

Although, as said, the LE improved the messaging system compared to the BGE, messages still have to be manually specified in terms of both sender and receiver, making it difficult for the user to manage communications between objects.

VSE overcomes this limitation by adopting the *event-condition-action* approach (Desolda et al. (2017)). This approach is exploited by tools like, e.g., AgentCubes (Ioannidou et al. (2009)), GameSalad[18], Click Team[19], etc. Among the various alternatives, the implementation adopted in Kodu[20] was specifically considered.

Kodu is a VPL-based environment that has been created by Microsoft Research to make video-game programming accessible to children. With Kodu, users can create interactive 3D scenarios by developing programs that can exploit also sophisticated features which are typical of professional game design, like camera control, collision detection, etc. (MacLaurin (2011)). Its effectiveness as a tool for exploring other computer science concepts rather than just video-game programming has also been investigated in the past (Stolee and Fristoe (2011)).

The key feature of Kodu is that it is completely event-driven (Fowler et al. (2012)). Users define so called "rules" (1), which are evaluated according to the "if this happens, do that" paradigm. As illustrated in Fig. 2 (a high-resolution version of this figure is available for download[21]), rules are specified using graphics "tiles" (2), the building blocks of the language, which can be assembled in "When-Do" strips. Like the Scratch's scripts, Kodu's rules are attached to active objects. The "+" operator (3) is used to connect the

[18]GameSalad: https://gamesalad.com/
[19]Click Team: www.clickteam.com
[20]Kodu: https://www.kodugamelab.com/
[21]High-resolution version of Fig. 2: http://tiny.cc/4uvqmz

Figure 2: GUI of Kodu.

"When" and "Do" parts of a rule: when the events described by the tiles on the left side of the "+" operator occur, the actions on the right side are executed.

In the VSE, the Kodu's event-driven approach has been directly integrated in each application logic's elements. Users can define complex rules by clicking on the "When-Do" buttons of an object, then wiring an element's output to the input of another one. In fact, Kodu's rule strips have been replaced by the *linking-and-wiring* paradigm, which is used in many environments like the BGE, but also AudioMulch[22], Audulus[23], Nuke[24], Node-RED[25], and SpaceBrew[26], among others, and whose effectiveness has been already evaluated in various domains (Lizcano et al. (2016); Spahn and Wulf (2009)).

3.1.3. Relations Visualization

Lastly, as anticipated, in order to make it easier for the users to visualize the communications occurring among objects, the VSE has introduced a scene-centric approach that overcomes the object-centric one adopted by the

[22] AudioMulch: http://www.audiomulch.com/
[23] Audulus: audulus.com/
[24] Nuke: https://www.foundry.com/products/nuke
[25] Node-RED: https://nodered.org/
[26] SpaceBrew: https://docs.spacebrew.cc/

LE (and BGE). Basically, besides serving as a container of virtual, interactive objects to be displayed at interaction time, a scene is expected to be a convenient way to organize the user's work. Each VSE project can consist of multiple scenes, each containing one or more interactive/not-interactive assets. Each scene is associated with a specific color (chosen by the user) that, as said, helps the user to identifying the assets' relations. Depending on the complexity of the project and on the user's approach, the number of scenes and assets can be so large to make it impossible to simultaneously visualize all the scenes with their assets and relations. Thus, the VSE shows all the assets and relations of a scene plus the relations of the scene's assets with objects in other scenes at once. In this way, users do not need to change panel or window to work on a scene. Moreover, they are not forced to select a specific asset to visualize its "When-Do" relations, which are always displayed (possibly summarized, or collapsed).

3.2. Architecture and Usage Workflow

The high-level architecture of the tool and the intended usage patterns are illustrated in Fig. 3. The creation and visualization of the interactive assets takes place through the steps reported in the following.

First, a file containing assets (like video and audio clips, text descriptions, static or animated 3D models, etc.) is loaded. The creation of these assets does not require any programming skills, since it involves the use of modeling and animation suites like Blender or Maya, etc. Assets do not need to be created in Blender: the only requirement is that they can be exported to a Blender- (VSE-) compatible *.blend* file. Assets may be also purchased or downloaded for free from a number of online catalogues like, e.g., BlendSwap[27], Free3D[28], and Blender for Architecture[29] (for 3D models), or Bendsound[30] and Free Music Archive[31] (for audio files). Imported assets will constitute the library of objects that will possibly appear in the 3D interactive application being created.

Once the library is loaded, the user can start to organize the available objects into scenes, by choosing objects to show at a given time. Once the

[27]BlendSwap: https://www.blendswap.com/
[28]Free3D: https://free3d.com/it/3d-models/blender
[29]Blender for Architecture: http://blender-archi.tuxfamily.org
[30]Bendsound: https://www.bensound.com/
[31]Free Music Archive: https://freemusicarchive.org/

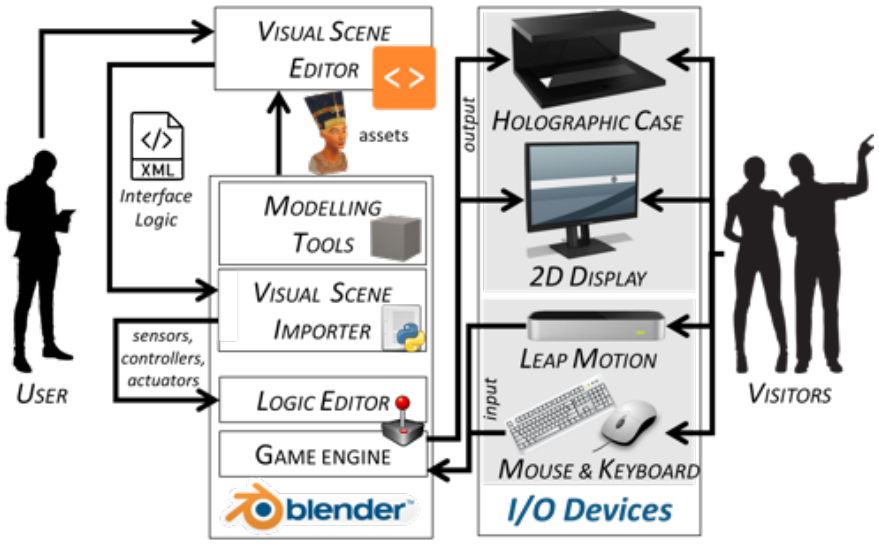


Figure 3: High-level architecture and usage workflow of the proposed tool.

scenes are created, the user defines the methods for transitioning from one scene to another, and the behaviors (like the playback of an animation or the appearance of a text description) to be activated when specific events (like an input provided by the user, a collision between two objects, etc.) occur. In the VSE, the set of relationships between events recognized and actions to be activated is named "Interface Logic".

Once the Interface Logic (or part of it) has been defined, the user can export the project to a XML file. This file may be used later to implement changes in the logic using the VSE, or can be imported in Blender to run the interactive application in the BGE (for previewing, e.g., during development, testing or deployment). The import operation is performed with an add-on, named Visual Scene Importer. This add-on automatically translates the Interaction Logic to the visual description based on sensors, controllers and actuators which is recognized by Blender's Logic Editor. When the import is completed and the corresponding bricks and connections are added, the user can further modify the 3D visualization or the Interface Logic through Blender's native windows (e.g., the 3D View and the Logic Editor). It is worth observing that, although at present the framework selected to run the interactive application is the BGE, other frameworks, like, e.g., Unity or Unreal Engine, could be also used in the future, without the need to make

any change to the VSE. In fact, in order to support new frameworks, only the script that is in charge of translating the XML file generated by VSE into the target application logic (currently based on BGE bricks) should be re-implemented.

The rendering of the interactive assets generated in real-time by the BGE can be visualized through the selected output devices. As said, currently a 2D display and a holographic case (more details are provided in Section 4) have been considered, though in principle any other visualization means supported by Blender (or other tools, in the future), including stereoscopic displays, augmented/virtual reality systems, etc., could be easily integrated. At present, users can interact with produced contents by using hand gestures or traditional interfaces (like mouse and keyboard). However, since the BGE can be fully scripted, support for other interaction methods that can be handled to the host computer (like body gestures, voice control, etc.) could be integrated as well.

It is worth observing that, in the high-level architecture (and usage workflow) depicted in Fig. 3, the VSE presented in this paper fully replaces the LE tool proposed in Sanna et al. (2016). All the BGE's scriptable functions that were implemented in the LE were considered also in VSE, thus making the new tool able to support the same tasks and to manage the same construction complexity that could be achieved with the reference one. The export format used by the VSE is more sophisticated than that of the LE; hence, a dedicated import add-on (not shown) had to be developed.

3.3. GUI's Design Steps

Considering the limitations of the LE reported in Section 3.1, a new tool was created to let unskilled users produce 3D interactive contents in an effective, efficient and satisfying way. A new GUI and a new graphics notation were defined through design and development steps that considered the feedback collected on the previous tool, as well as the outcomes of literature review and established usability principles (Nielsen (1995)).

The GUI of the VSE and its visual notation can be regarded as the outcomes of a process that involved continuous refinements and improvements. Starting from the initial mockup based on the Scratch's panel-based layout, different content organizations (e.g., position of objects in panels, links between them, etc.) and interaction methodologies (e.g., drag-and-drop, selection from a list, etc.) were experimented and modified when judged as not

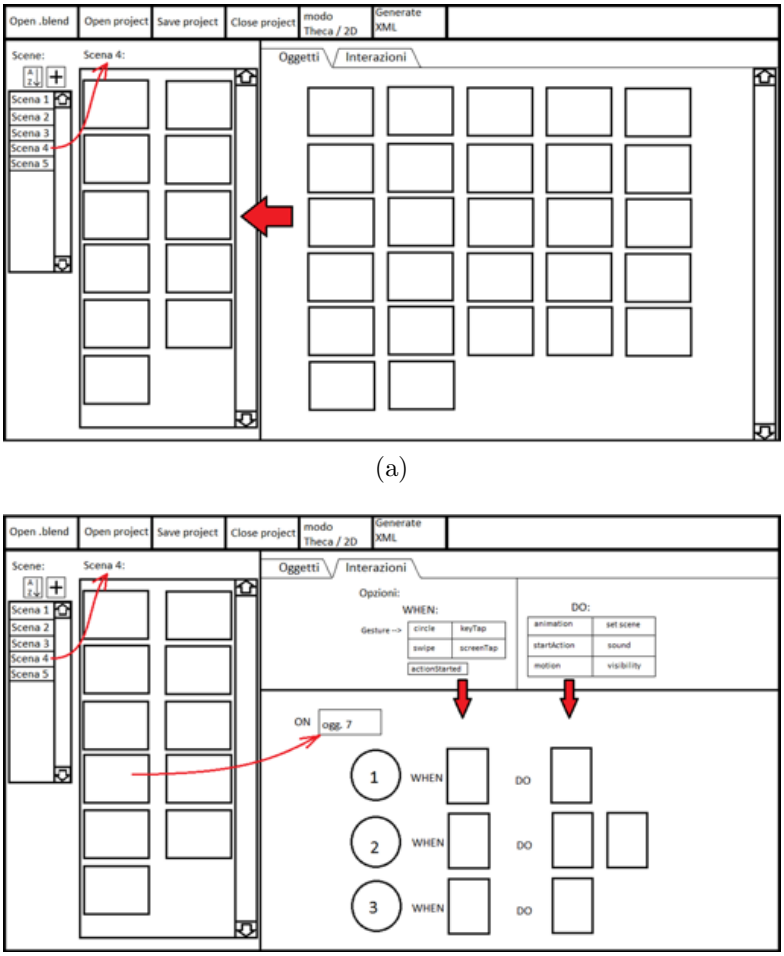


Figure 4: Initial mockup of the VSE's GUI: a) panels for managing the objects (right) in different scenes (left), and b) panels for defining the Interface Logic for the selected object.

appropriate (Fig. 4a). Since from the first mockup, the GUI was designed around the Kodu's "When-Do" paradigm (Fig. 4b).

However, this mockup proved not to be completely effective. One of the issues was associated with the difficulty of setting the various parameters required for the definition of the Interface Logic. The problem was the limited space to manage this operation allocated in the interface (bottom-right panel in Fig 4b). In the final implementation, a dedicated, adjustable panel was allocated to the definition of the behaviors for the selected object.

However, the most serious issue was related to the impossibility to visualize the interactions among different objects (created by wiring the "When" block of an object with the "Do" block of another object). It was additionally realized that this limitation, already present in Kodu's rule strips, was intensified by the message-passing approach used, e.g., in the BGE and the LE. In fact, this approach requires the users to specify (and remember) the actual message to pass to target objects, without actually seeing them at the time of defining the message.

As said, to deal with these issues, it was decided to exploit the link-and-wire paradigm in order to allow users define relations (later referred to as "Links") among "When-Do"-based blocks attached to different objects. Because of this choice, a different content organization letting the user see more than one object at the same time had to be implemented, which led to an organization based on the said scene-centric approach.

Other aspects that were considered to define the final implementation of the GUI are colors and indentation. Colors have been used to make it easier for the user to recognize a specific scene (and consequently the related links) and the When-Do buttons whereas indentation has been used to organize contents in the "When-Do" blocks.

3.4. GUI's Functionalities and Visual Notation

The final version of the main window of the VSE's GUI is shown in Fig. 5 (a high-resolution version of this figure is available for download[32]). In the following, the core components are described through dedicated sub-sections.

3.4.1. Library

Once the user has loaded a *.blend* file or a previous project, the interface presents, on the right side, the list of available objects which can be added to the scenes (1). Objects are represented by a small icon (showing a preview of the object's appearance when rendered in the 3D program) and a name. The list contains not only 3D geometries (like Cube, Sphere, etc., in Fig. 5) but also, e.g., lamps (like, LampA, LampB, etc.) which could be used in the scenes to implement a given lighting. Like in common modeling suites, if a given object is organized in a structured fashion (i.e., it has one or more child objects, which inherit transformations applied to their parents), then it

[32]High-resolution version of Fig. 5: http://tiny.cc/j4nqmz

is visualized in a tree within the library. Adding a parent object to the scene would automatically add all its children. A search bar lets the user filter objects based on their name. When an object is selected (like TorusDue, in the figure), a larger version of the preview is displayed on the bottom-right side of the interface.

3.4.2. Scenes

The panel labeled as (2) in Fig. 5 shows the scenes and the Interface Logic defined for them. A scene can be represented in either a collapsed or expanded way. Only one scene at a time can be expanded, allowing the user to edit it. The collapsed visualization (used for scenes Schema, Start and Scene_2 in the figure) reports only the name of the scene, the color assigned to it, and two buttons which allow the user to perform delete and edit operations. When the expanded visualization is chosen for a scene (like for scene Scene_3), all the collapsed scenes are automatically rearranged in the panel in order not to be covered by the selected scene.

On the top of the expanded scene (3), a toolbar provides the user with several configuration functionalities. For instance, the user can select the color for the scene (which will be used for drawing Links) and set its name; automatic coloring and naming are also implemented. A checkbox allows the user to set the current scene as the first to be launched when the BGE is started and the interactive application is executed. The starting scene is highlighted with the color assigned to it (like for the scene named Start in the figure). A button is available to let the user duplicate the current scene, creating a copy with the same objects and the same Interaction Logic.

Under the toolbar, a large panel (right side) allows the user to manage objects appearing in the given scene (4) and to define their behavior using "When-Do" blocks (5). A smaller panel (left side) is meant to let the user define global behaviors affecting the whole scene: this functionality, which is accessed via a dedicated button, will be discussed later.

3.4.3. Objects

A new object can be added to a scene by dragging it from the library. As for the scenes, objects can be visualized either in collapsed or expanded way. However, in this case, many objects can be expanded at a given time. This way, it is possible for the user to define (and see) relations among objects.

In the collapsed visualization, only the name of the object and the buttons to delete and edit it (i.e., pass to the expanded visualization) are shown.

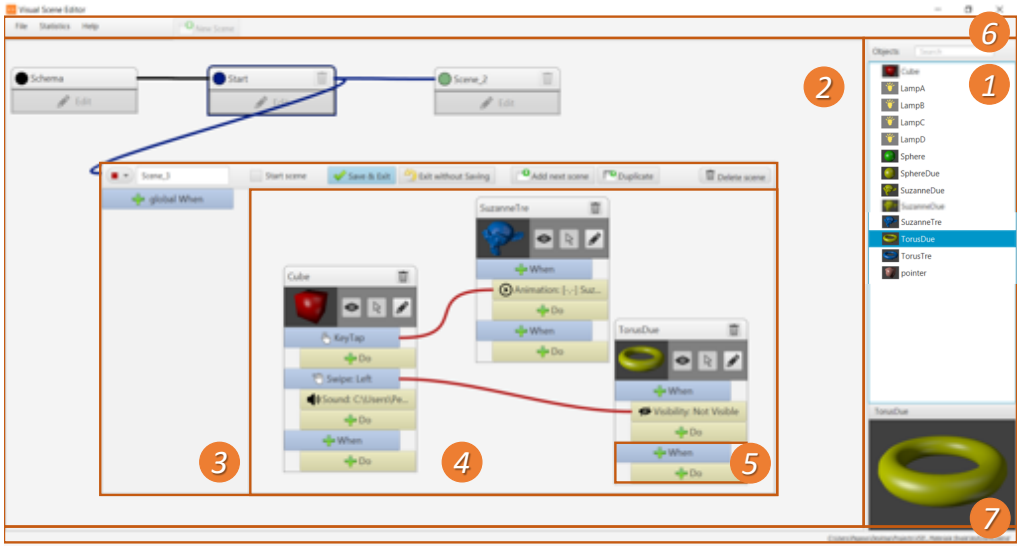


Figure 5: Final implementation of the VSE's GUI: 1) object library, 2) scenes, 3) selected scene, 4) scene's objects, 5) object's "When-Do" blocks, 6) menu bar, and 7) status bar.

When expanded, each object presents a title bar with the name and the delete button. Under the toolbar, a preview of the object is provided, together with three additional buttons. The first button is used to change object's visibility in the scene. The second button is used to make the object a *pointer* for the current scene. The pointer is the visual representation of the target of user's interaction. For instance, in the current implementation, when the mouse and the 2D display are selected as input and output devices, respectively, the movement of the mouse is transferred to the pointer object and the basic cursor is replaced with the representation of the pointer object itself. If the holographic case with hand tracking input is selected, cursor position is controlled by the 3D position of the user's hand in the tracked space. The last button activates the collapsed visualization for the object.

The bottom part of each object allows the user to define the Interaction Logic through "When-Do" blocks. By default, each object comes with an empty "When-Do" block. As the user starts modifying the default block by clicking the corresponding button, a new block is automatically added; moreover, if an action is specified for a "Do" block, a new "Do" block is automatically created under the same "When" block.

When the user has configured a "When-Do" block, an icon and a short description of the selected action/condition are added to the button (for

instance, the KeyTap event for the "When" block defined for the Cube object, or the Animation playback of the "When" block defined for the SuzanneTre object, in the figure).

The types of events recognized and actions supported will be presented in Section 3.4.6.

3.4.4. Links

The "When-Do" approach described above and adopted also in Kodu can be used when the conditions to be recognized and the actions to be executed concern the same object. As said, to overcome the inter-object communication method based on message-passing used in the BGE and the LE, the VSE introduced the concept of Link. Links are lines drawn between objects, which allow a condition detected by a given object to activate an action associated with another object, thus simplifying the creation of the logic.

A Link can be created by drawing a line (with a drag-and-drop operation) from the "When" block of an object to a "Do" block of another object. Links can be used also to change or reload a scene. In this case, the link has to be created between the "When" block of an object and a scene's toolbar.

The color of the Links is that of the scene that contains the object the connection originates from. As a matter of example, in Fig. 5 the Links between objects of Scene_3 are red since all the Links start from an object in Scene_3 (whose color is red), whereas the Link that has been defined to pass from the Start scene to Scene_3 is blue, since it originates from an object (not visible in the figure, due to the collapsed visualization) contained in the Start scene (which is assigned the blue color).

A click with the right mouse button over a "When-Do" block shows a menu that allows the user to delete the block and/or remove a specific outgoing Link.

3.4.5. Templates and Scene Behaviors

In Fig. 5, it is possible to notice in the left-top corner of the scene panel (2), a scene named Schema. The Schema scene is a sort of template that can be used to specify object/behaviors that are present/valid in/for all the scenes without the need to repeat the definition in each scene. It is generated automatically when a new project is created, and it is visually represented as a normal scene. With respect to the other scenes, however, it cannot be renamed, deleted or moved. Moreover, it is not possible to duplicate it, or

set it as the initial scene. A practical use of the Schema scene could be that of setting the lighting for all the scenes, or implementing the shared logic for managing the cursor. If the user specifies two different behaviors for the same object in the Schema scene and in a normal scene, the definition in the Schema scene is ignored.

As anticipated, for each scene it is also possible to define so called "GlobalWhen" blocks, which are responsible for handling events that pertain the whole scene and not just a particular object in it. For instance, the action of changing the current scene when a specific gesture is recognized during interaction with a given object is something that can be dealt with using a canonical "When-Do" block. However, changing the current scene independent of where the gesture is actually performed is something that should be delegated to the "GlobalWhen" block.

3.4.6. When-Do Blocks

The configuration of each "When-Do" block is accomplished through dedicated windows.

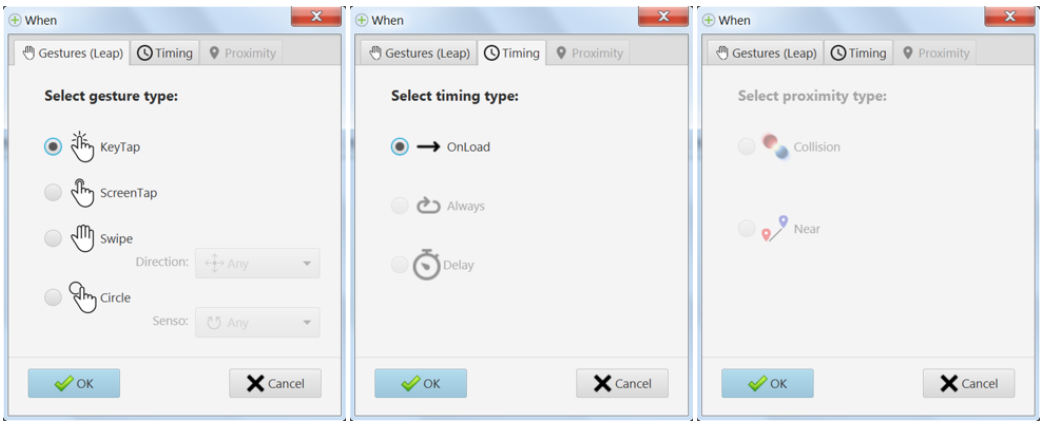
For "When" conditions, the interface is structured in three tabbed panels (Fig. 6), grouping semantically related events.

By means of the Gesture panel (Fig. 6a), the user can indicate the hand gesture to be recognized in order to trigger an event. As said, the device currently used for hand gestures recognition is the Leap Motion sensor. Hence, the panel shows only the four supported gestures, i.e., circle, swipe, key tap and screen tap. More details about these gestures can be found in the documentation of the Leap Motion SDK[33]. For sake of simplicity, mouse events are mapped onto recognized gestures.

The Timing panel (Fig. 6b) is used to specify conditions influenced by time. In particular, the OnLoad option is set to trigger an event when the scene is loaded (in the case the condition is specified in a GlobalWhen block) or when the object is added to the scene (if specified in the "When" block of an object). Option Always triggers events in a continuous way, whereas Delay fires the event after the specified time interval.

The Proximity panel (Fig. 6c) is meant to manage events associated with spatial conditions (two objects that collided or are close to each other at runtime).

[33]Leap Motion SDK documentation: https://bit.ly/2XOMXHP



(a) (b) (c)

Figure 6: Window for specifying parameters of a "When" block: a) Gesture panel, b) Timing panel, and c) Proximity panel.

For managing actions associated with a "Do" block, the window includes two tabbed panels (Fig. 7).

With the Action panel (Fig. 7a), the user can define three types of actions to be executed on the object (the playback of an animation, the reproduction of a sound and the change of the object's visibility) and the corresponding parameters (the animation or the sound to be played, the starting and end frame of the animation, whether the object is visible or hidden).

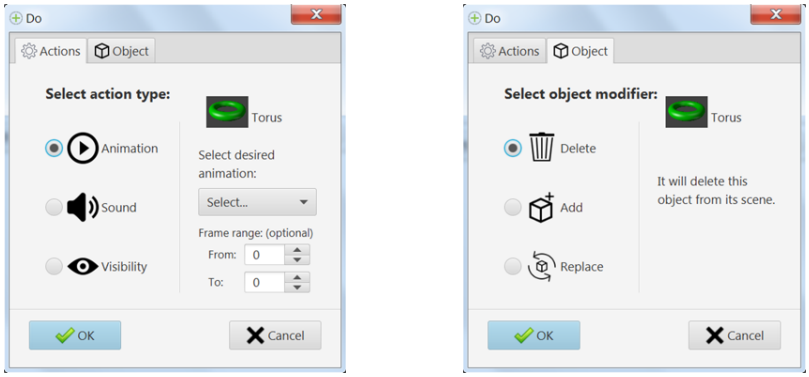
By means of the Objects panel (Fig. 7b), the user can manage actions that involve the creation and removal of objects in/from the scene. Intuitively, the Delete option removes the object, Add places a new object in the same position of the current object, whereas Replace deletes the current object and adds a new one (specified by the user) to the scene.

3.5. Software Modules

To implement the VSE tool, the JavaFX graphic library[34] was used. The software modules shown in Fig. 8 were organized according to the Model View Controller (MVC) software design pattern.

The View includes the modules listed below:

[34] JavaFX: https://www.oracle.com/technetwork/java/javafx/overview/



(a) (b)

Figure 7: Window for specifying parameters of a "Do" block: a) Action panel, and b) Object panel.

- Project Window: it manages the graphics components belonging to the main window;
- Scene Container: it is the visual container exploited by the user to assembly the current scene;
- Node: corresponds to one of the assets inserted into the current scene and configured by the user;
- When/Do Block: it allows the user to visualize and configure each of the "When" conditions and "Do" actions for a given asset.

The Model contains the following modules:

- Input Manager: it is the data structure representing the possible input device(s) to be managed by the interactive application;
- Blend Scene: it contains information concerning the library of objects that can be used to develop the application;
- Interaction Logic: it is the data structure holding the logic of the application being created;
- XML Project Wrapper: it stores the information required to create the XML file exporting VSE project data in a format that can be loaded in the BGE.

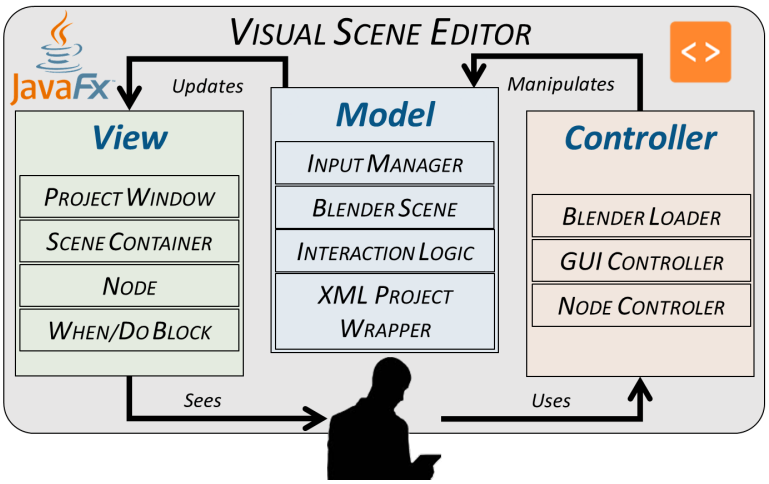


Figure 8: Architecture of the proposed VSE tool: software modules.

Finally, the Controller is based on the modules below:

- Blender Loader: it contains functions needed to import into the VSE assets made with Blender (or exported in a compatible format);
- GUI Controller: it supervises the behavior of the main window;
- Node Controller: it controls the asset configuration.

4. Use Case

In order to show how it could be exploited in a real usage scenario, the VSE was used to generate an interactive application for a possible virtual exhibition of a known artwork. An ancient Egyptian artifact, namely the bust of the Queen Nefertiti exposed at the Berlin's Neues Museum was considered. The assets used in the 3D program are those exploited in (Sanna et al. (2016)). The application was used in public exhibitions on virtual/augmented reality and other emerging technologies which were hosted at the authors' university.

The overall setup of the exhibit is illustrated in Fig. 9. The output device is represented by a custom-made holographic case: the device leverages the well-known Pepper's ghost effect to project 3D digital images on a pyramid

Figure 9: Overall setup of the system used for demonstrating the virtual exhibition of the bust of Queen Nefertiti.

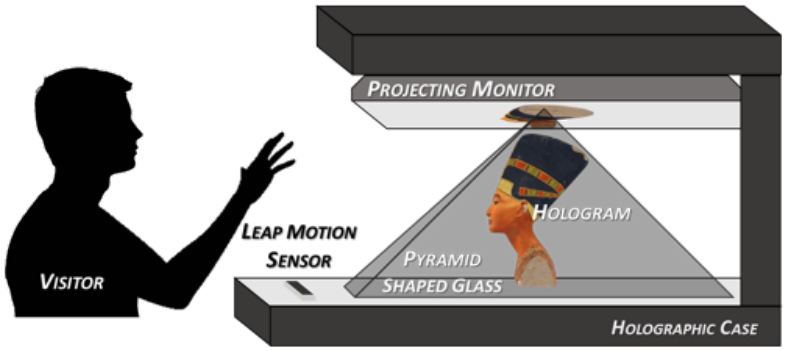


Figure 10: Holographic case.

shaped glass using a video source (the projecting monitor) which is hidden to the visitor in the top of the case (Fig. 10).

User interaction is based on hand tracking and hand gestures. A Leap Motion sensor mounted in the case is used to gather hand movements in the 3D space in front of it.

The application consists of nine scenes, which are illustrated below.

- Start (Fig. 11a): it provides a brief presentation of the gestures recognized by the system that allow the visitor to interact with the 3D contents; when the visitor performs a key tap gesture on the Start label displayed in the center of the holographic space, the application is launched and the next scene (Menu) is displayed.

- Menu (Fig. 11b): it allows the visitor to choose among three different scenes to visualize: Video, History and 3D Model; a key tap performed on one of the corresponding icons loads the new scene; a circle gesture can be used to return to the Menu scene.

- Video (Fig. 11c): it plays a video introducing the artwork.

- History (Fig. 11d): it contains a sequence of text descriptions, which present the history of Queen Nefertiti; descriptions can be scrolled using swipe gestures.

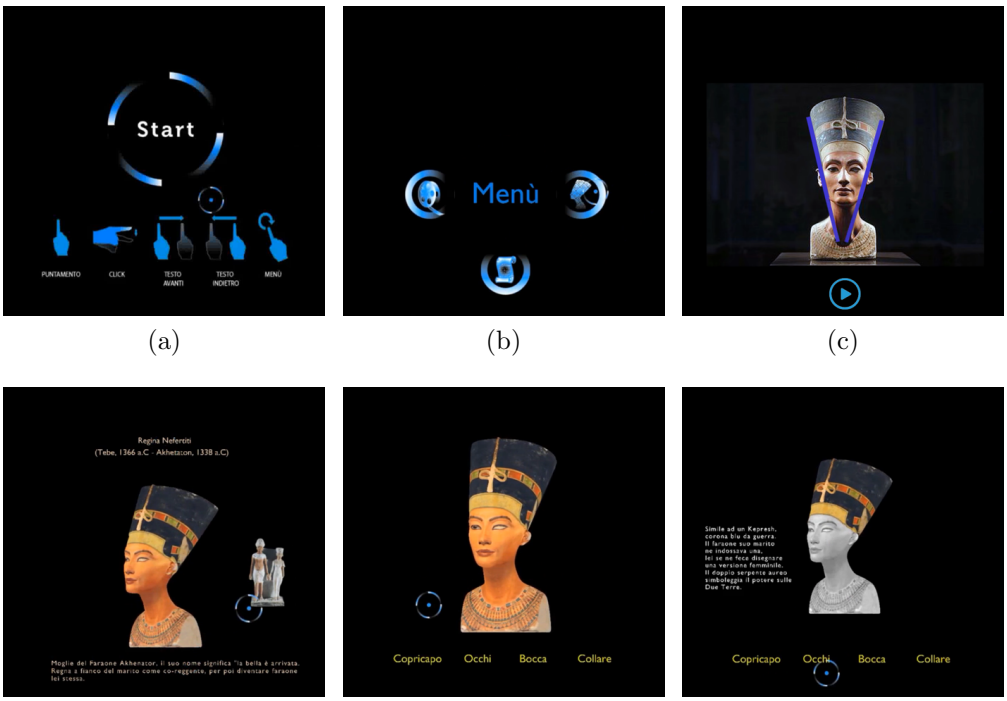
- 3D Model (Fig. 11e): it shows a 3D model of the Queen Nefertiti's bust; an animation that rotates the model clockwise is automatically played; when the visitor performs a tap gesture on the labels indicating the collar, eyes, mouth and head to the bottom of the holographic projection, a new scene is loaded where the selected part is highlighted on a grayed model using colors (see below).

- Collar, Eyes, Mouth and Head (Fig. 11f): the part of the artwork selected in the previous scene is highlighted, and a text description appears providing further information of the bust's detail. If the visitor performs a circle gesture, the current scene is closed and 3D Model scene is reloaded.

This use case was selected because it is an example of how scenes can be arranged to present different types of assets, namely, graphics widgets (like buttons, panels, labels, etc.), 3D models (the model of the Queen Nefertiti's bust and its details), animations (e.g., to rotate the bust in the 3D Model scene), and videos (as in the Video scene).

A video showing the generated interactive application is available for download[35]. Three videos showing the realization of the application using the BGE, the LE and the VSE tools are also available, which allow to qualitatively compare the different levels of complexity of both the creation process and the resulting logic[36].

[35] Queen Nefertiti, video of the resulting application: http://tiny.cc/bydtbz
[36] Queen Nefertiti, video of the creation process: http://tiny.cc/axdtbz



(a) (b) (c)

(d) (e) (f)

Figure 11: Some of the scenes crated for the Queen Nefertiti use case: a) Start, b) Menu, c) Video, d) History, e) 3D Model, and f) Head (Collar, Eyes and Mouth are similar).

5. Experimental Evaluation

With the aim to evaluate the performance of VSE with respect to other tools, two experiments were carried out by involving subjects with different levels of expertise in (interactive) application development and design of public exhibitions. In the following, user groups composition for the two experiments is first introduced. Tasks to be performed and experimental methodology are then described. Finally, criteria considered in the evaluation are discussed.

5.1. Experiments and Participants

The first experiment represented a preliminary study (later referred to as S1) aimed to set the baseline for the evaluation of the tools' performance. Hence, it involved three users skilled in computer programming, precisely, 3D

game/application developers with years of experience with various languages and suites, including Blender and the BGE.

The second experiment was designed as a user study (later referred to as S2) aimed to investigate tools' effectiveness and usability with potential end-users. Hence, the user group consisted of 14 volunteers aged between 22 and 29 years ($\mu = 25.93$, $\delta = 1.94$), selected among students enrolled in the B.Sc. degree on Design and Visual Communication at Politecnico di Torino [37]. They were expected to possess skills in the fields of visual communication and product design, with extremely basic programming experience and no knowledge of the considered tools. Based on available statistics, during the degree program (e.g., in internships, as well as in course and thesis projects) as well after graduation, many of them will be working in the field targeted by the present work.

5.2. Tasks

The interactive application to be developed in both the studies was a simplified version of the planner tool on the IKEA website[38]. The original tool allows IKEA's customers to create/customize, e.g., a sofa by assembling several components, like loungers, poufs, etc. and choosing sizes and locations for them. Once completed the assembly, customers can visualize an animation for the created product.

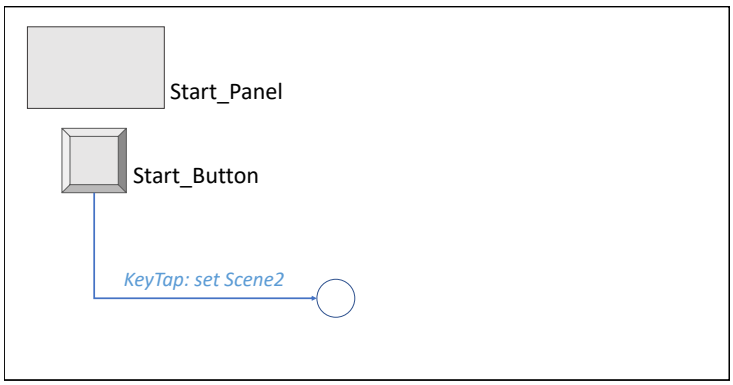
Participants were requested to create three scenes (labeled from 1 to 3), each containing several interactive 3D objects. In each scene, they were requested to manage various conditions (corresponding to different user interactions) and activate corresponding actions (i.e., changing the visibility of an asset, playing an animation, etc.).

A high-level representation of the expected application's workflow is presented in Fig. 12, whereas the actual aspect of the resulting application when imported and visualized in the BGE is illustrated in Fig. 13.

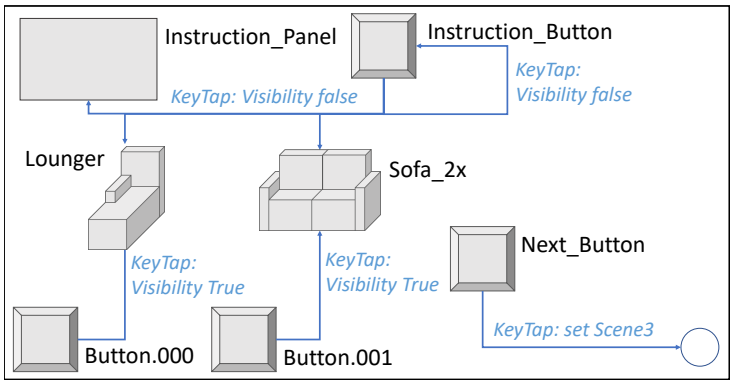
Scene1 (Fig.13a) is the start scene, and represents an introduction to the actual application. The scene contains only two objects: Start_Panel and Start_Button. The interaction to be implemented in this scene is a key tap gesture on the Start_Button, which triggers a transition to Scene2.

Scene2 is the scene in which the customer is provided with the instructions for using the application (Fig. 13b), and can configure the sofa by choosing

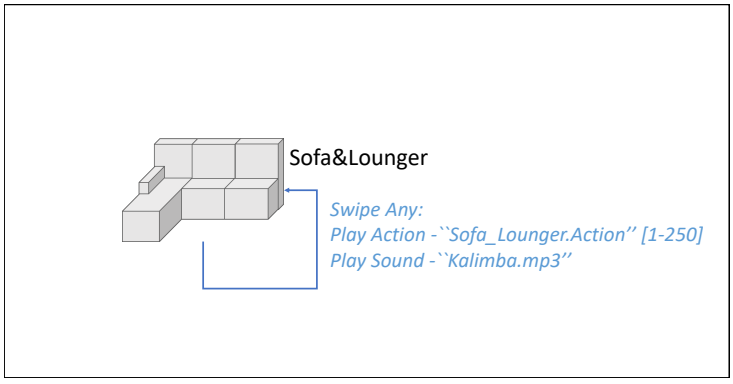
[37] Design and Visual Communication B.Sc.: https://didattica.polito.it/laurea/design/en
[38] IKEA's planner: https://bit.ly/2OsIG6P



(a)



(b)



(c)

Figure 12: Overview of the scenes to be created: a) Scene1, b) Scene2, and c) Scene3.

components from a list (Fig. 13c). In order to make the task not repetitive and reduce the completion time of the overall experiment, it was decided to partially implement the logic of this scene, limiting the number of components to be managed and the interactions to be implemented. Thus, participants were requested to insert into the scene and configure only the following assets: Instruction_Panel, Instruction_Button, Button.000, Button.001, Sofa_2x, Lounger, Next_Button. The interactions to be managed in Scene2 are listed below:

1. a key tap gesture on Instruction_Button changes the visibility to false for the Instruction_Button itself, as well as for the Instruction_Panel, the Lounger, and the Sofa;
2. a key tap gesture on Button.000 and Button.001 sets the visibility to true for the Lounger and the Sofa_2x, respectively;
3. a key tap gesture on Next_Button triggers a transition to Scene3.

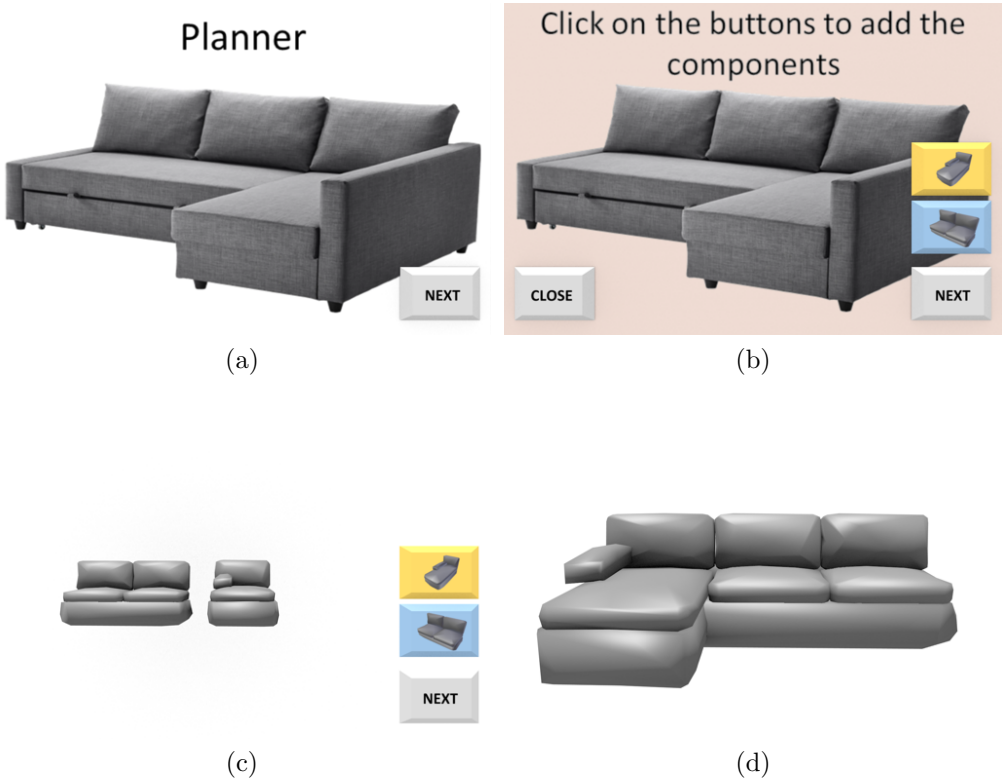
Scene3 (Fig. 13d) is the scene in which the customer can visualize the assembled sofa. It includes only the Sofa&Lounger object. The interaction to be created activates the reproduction of a sound and the playback of an animation when a swipe gesture is performed on the Sofa&Lounger.

5.3. Methodology

Because of the expertise of the involved subjects, in study S1 it was possible to test the creation of the above scenes with the VSE and with two other tools, i.e., the LE and the BGE. Based on results achieved in this study (Section 6.1), in study S2 volunteers were only asked to operate with the VSE and the LE.

All the participants were first introduced to the experiments. The way to work with the tools was then explained (when needed), by letting them familiarize with the interfaces. When they felt ready to start the experiment, participants were asked to first create the three scenes with one of the tools. Afterwards, they were invited to do the same with the other tool(s). The order was continuously switched, in order to limit the impact of learning effects in the evaluation.

The creation of the application was split in two steps. In the first step, instructions describing scene composition and interactions to be implemented were presented to the participants by showing them the workflow in Fig. 12. Afterwards, only for study S2, participants were suggested to take some notes about the application to be created, by filling in a table-based template



(a) (b)

(c) (d)

Figure 13: Scenes of the application created by participants of the user study as rendered by the BGE: a) Scene1, b) Scene2 (instructions visualization), c) Scene2 (sofa assembling), and d) Scene3. For sake of readability, colors in this figure have been altered w.r.t. the Blender's native ones which were seen by the participants during the experiment.

with their annotations (templates for the VSE and the LE are reported in Table 1 and Table 2, respectively). This step simulated a design process that could let users organize contents by following the working schema of the specific tool used to implement the application. Before taking notes, users were given the possibility to adjust the structure of the template (e.g., adding/removing columns, etc.), in order to let them make the annotations more aligned with their mental structures. In the second step, participants were asked to develop the three scenes (possibly using the notes). Progress was supervised. At the end, participants were informed about the possible presence of errors, and were asked to fix them in order to develop a fully working set of scenes.

When	On	Do	On
KeyTap	*Object*1	*Set Visibility False*	*Object*2

Table 1: Table template to use with the VSE: interaction "on key tap gesture on Object1, change the visibility of Object2 to false" is represented.

Sender	Msg Subject	Msg Text	Receiver	Action
*Object*1	*OnKeyTap*	*Set Visibility False*	*Object*2	*Change the visibility*

Table 2: Table template to use with the LE: interaction "on key tap gesture on Object1, send a message to Object2 for setting its visibility to false" is represented.

Participants could have a preview of the application being created by generating the XML file (with the export function of the VSE and the LE) and importing it in the BGE. The presence of critical errors like, e.g., forgetting to set a cursor or defining the start scene, could prevent the generation of the XML file and, consequently, the visualization of the preview.

Three videos showing the execution of the above tasks with the BGE, the LE and the VSE are available for download[39].

5.4. Metrics

Because of the differences between studies S1 and S2, the evaluation required the definition of two sets of metrics.

In particular, for study S1, two objective measures were collected, namely, the amount of time and the number of visual components (bricks/blocks and connections) employed by the skilled users to create the interactive application with the three tools. Two different times were measured during the experiment: the first one is the time needed by the participant to obtain his or her best results, whereas the second one is the time to fix possible errors identified by the supervisor. These times were then cumulated to determine the overall completion time for creating the application. Moreover, at the end of the experiments, skilled users were requested to provide feedback on their experience in an interview.

For study S2, objective metrics encompassed completion time and number of errors made. In addition to the two definitions of completion time given

[39]Videos of the experiments with the three tools: http://tiny.cc/8biqkz

above, a third definition was added to take into account the time needed by participants to fill in the table template describing the application to be implemented. Subjective aspects were assessed by means of a post-test questionnaire organized in four sections[40]:

- Q1: general users' information, familiarity with 3D and programming languages/tools (visual or not);
- Q2: evaluation of system usability based on the SUS questionnaire (Brooke (1996));
- Q3: evaluation of the task load based on NASA Task Load Index, or NASA-TLX (Hart and Staveland (1988));
- Q4: users' preferences.

Concerning Q4, users had to express their preference for the two tools on a 1–5 scale (with 1 meaning high appreciation for the VSE, 3 neutral, 5 high appreciation for the LE) by considering four different aspects, i.e., scene, object and interaction/relation management, and overall preference. Possible comments or reasons for their choice were also collected. Users were invited to fill in the questionnaire after having completed the tasks with both the tools.

6. Results

In this section, results for the two studies are presented.

6.1. Study S1

Fig. 14 shows objective results obtained in the preliminary study. Fig. 14a reports the overall completion time, including both the time to finish the tasks (to the best of participants' understanding) and the time possibly required to fix errors (identified by the supervisor). Fig. 14b and Fig. 14c show the number of visual elements (blocks and connections, or Links, respectively) required to assemble the three scenes.

[40] Questionnaire: http://tiny.cc/w9nqkz

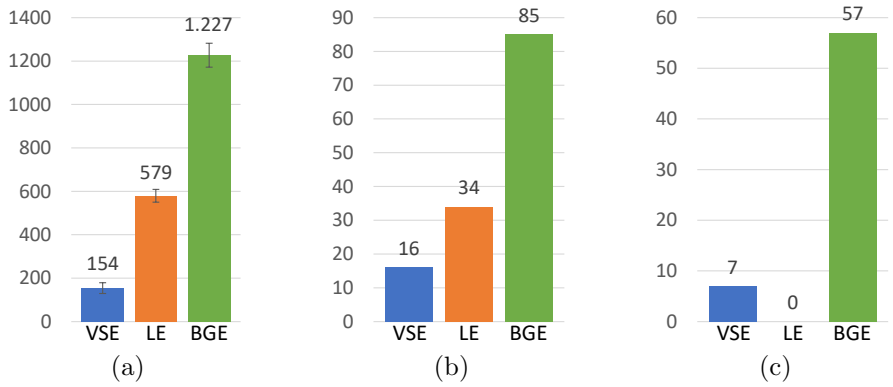


Figure 14: Objective results for study S1 (skilled users): a) average completion time for developing the application (seconds), b) number of blocks and c) number of connections or Links used.

Given the limited number of participants, results were not analyzed using statistical tools, but they were rather studied in an explorative way by considering means and variances.

With respect to completion time, observing the large differences among the average values and the small variances, it appears that VSE allowed participants to be much faster than the other tools. Moreover, in absolute terms, VSE also required participants to assemble/join a lower number of elements (blocks and connections). Connections are not accounted for the LE, since they are created automatically by the tool when the related blocks are selected. Due to the relatively high number of blocks (twice those of the VSE), selection required a considerable amount of time, slowing down the logic definition process. The VSE outperformed the BGE, being roughly seven time faster and requiring a considerably lower number of both blocks and connections. This fact is related to the need to define Message sensor-actuator pairs (and related connections) in place of a single Link connecting the "When" and "Do" blocks when two different objects had to communicate.

Comments gathered at the end of the experiments can be summarized as follow:

- the VSE could simplify repetitive operations that characterize both the BGE and the LE, like, e.g., the activation of an action on an asset upon recognition of a condition on another asset; simplification comes from the fact that the VSE is able to automate tasks that need to be

performed manually with the latter tools (e.g., setting up the message subject and content for both the sender and the receiver);

- although the LE was found to be similar to the BGE (a software that participants already knew), the VSE could be more easy to use by those who are not familiar with message-based communications and/or with programming paradigms;

- differently than both the BGE and the LE, the VSE provides users with a high-level view of the overall application being created, as well as of the interactions among assets; this aspect of VSE could help to design and manage the workflow of the application better than with the other two tools.

6.2. Study S2

Objective results for the second study in terms of completion time and number of errors are reported in Fig. 15. Statistical significance was analyzed using paired Student's t-tests ($p = 0.05$).

It can be easily seen that the VSE allowed participants to complete the assigned tasks in almost half of the time than with the LE. In fact, participants were, on average, 43.29% faster with the VSE than with the LE to fill in the table template ($p = 0.0034$), and 51.58% faster with the VSE than with the LE to create the application ($p = 0.0001$). Moreover, with the VSE they made, on average, roughly 11% of the errors made with the LE ($p = 0.0001$). The possibility to visualize all the connections among objects at the same time greatly lowered the time spent in adjusting the visualization. Furthermore, since in the LE the concept of message used in the underlying BGE was not abstracted, participants were still forced to manually set all the message parameters through a process that required a considerable amount of time compared to the mechanism used in the VSE (drawing a Link between the "When" and "Do" blocks). It is worth observing that, when operating with the VSE, most of the participants took advantage of the Schema scene to define components shared among all the scenes (cursor and lamps); with the LE, they would have been forced to specify the presence of these objects in each scene.

Regarding subjective results, according to section Q1 of the questionnaire, the majority of the participants stated that they had familiarity with graphics suites like Blender and Maya as well as with video and photo editing

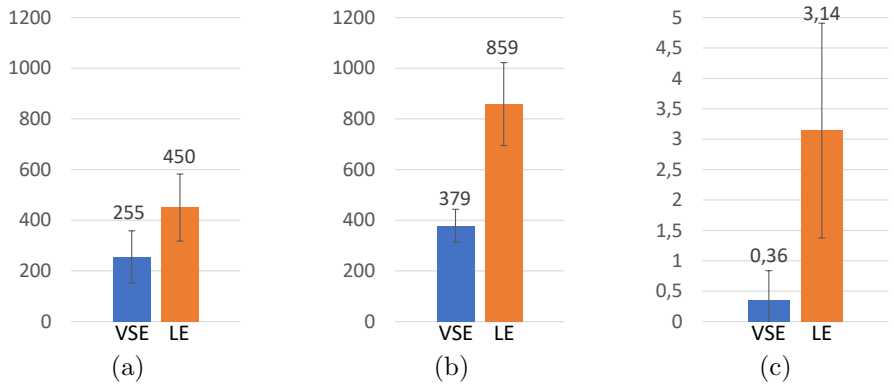


Figure 15: Objective results for study S2 (unskilled users): a) average completion time for representing the application's workflow (seconds), b) completion time for developing the application (seconds), and c) number of errors.

tools like Adobe Photoshop[41] and Adobe Premiere[42] (Fig. 16a); these kinds of software are regularly used in the classes of the considered B.Sc. program, where students are taught how to manage (visual) product design and communication. A small percentage of them said to have little or no experience with node-based programming tools and VPLs (Fig. 16b and Fig. 16c). Finally, roughly 70% of participants stated to have little experience with tools and languages for developing video-games and/or interactive applications (Fig. 16d).

Regarding results of section Q2, participants found the VSE as characterized by a higher usability compared to the LE. In particular, the VSE obtained a SUS score equals to 79.10 compared to the 53.57 achieved by the LE ($p = 0.0001$). According to Brooke (1996), the VSE score corresponds to the C grade in the SUS scale, which classifies usability as "Acceptable", whereas the score of LE corresponds to the F grade, that is,"Marginally acceptable". According to provided feedback, the capability of the VSE to automatically manage some key operations (like creating inter-object communications) reduced the number of operations to perform. This aspect made the participants perceive the VSE as easier to use and more satisfying than the LE. Furthermore, the possibility to take advantage of an overall view to

[41] Adobe Photoshop: https://www.adobe.com/products/photoshop.html
[42] Adobe Premiere: https://www.adobe.com/products/premiere.html

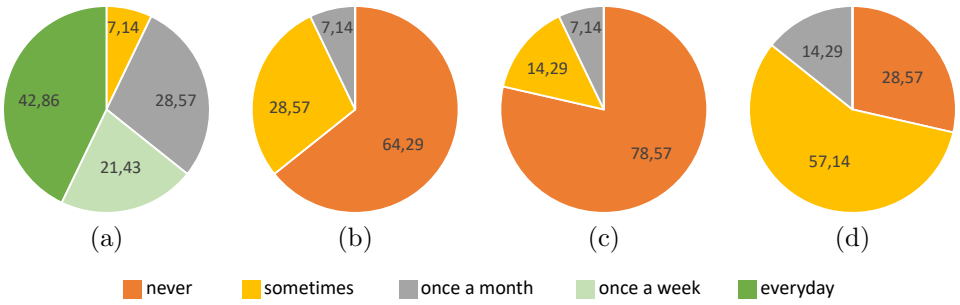


Figure 16: Results of section Q1 of the questionnaire concerning users' experience with: a) graphics suites, b) node-based programming tools, c) VPLs, and d) tools for video-game and/or interactive application development.

visualize all the interactions at the same time allowed participants to learn the VSE more quickly than the other tool.

These conclusions are also confirmed by the NASA-TLX results (section Q3). In particular, participants perceived the VSE as characterized by a lower workload than the LE (33.48 for VSE, 59.78 for LE, $p = 0.0001$), letting them achieve better results (performance) with less effort (mental demand) as well as with less errors to be dealt with at the end of the experience (frustration).

Results of section Q4 appear to confirm the above findings (Fig. 17). On average, roughly 85.5% of the participants preferred the VSE to the LE in all the aspects considered. Only slightly less than 10% of the participants preferred the LE, stating that, although the message mechanism was difficult to manage, the GUI was cleaner and easier to understand.

7. Conclusions

In this paper, a new tool supporting the generation of 3D applications for interactive exhibitions targeted to users with limited to no programming skills has been presented. In the development of the new tool, named VSE, EUD principles as well as pros and cons of related VPL-based environments like Scratch, Kodu, BGE and LE, among others, have been considered.

Differently than some of the previous works, the proposed tool adopts a scene-centric approach for managing asset connections, which allows users to better visualize the elements composing the various scenes and their interac-

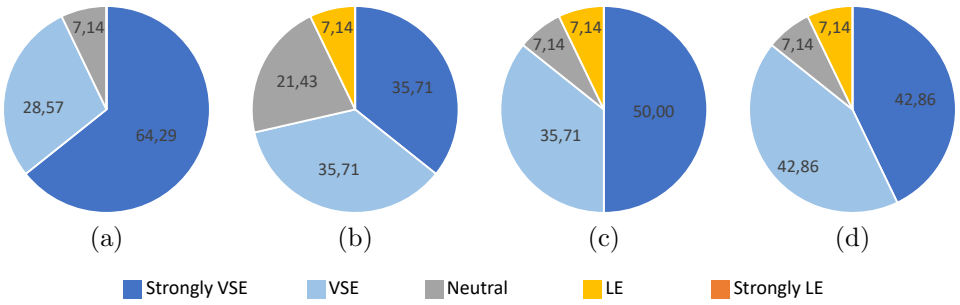


Figure 17: Results of section Q4 of the questionnaire about users' preference regarding: a) scene, b) object, and c) interaction/relation management, and d) the tool, overall.

tions. However, from other previous works it inherits features like, e.g., the "if this happens, do that" paradigm, which helped to make the definition of object's behaviors more intuitive.

Experiments have been performed to evaluate the effectiveness of the VSE with respect to both the BGE and the LE. Results obtained by involving users with both 3D graphics and programming skills showed that the VSE allows them to create the intended application in a shorter time and using a lower number of blocks. Regarding target end-users (i.e., users with little to no programming experience but with a background in design and visual communication), with the VSE they were able to complete the assigned tasks saving time, as well as making a lower number of errors. The superiority of VSE was also confirmed by subjective results, which indicated a superiority of the proposed tool for all the usability dimensions considered. Overall, the VSE was largely preferred to the LE.

As said, the domain of public exhibitions was tackled in this paper. In particular, the flexibility of the VSE was shown by exploiting it for the creation of two different applications: the first one (the use case) focused on a cultural heritage interactive exhibit, the second one (used in the experiments) targeted to customers of either physical or online stores. Notwithstanding, the VSE could be used also to create 3D interactive applications required in other contexts.

Based on encouraging results obtained with the current study, future works will be devoted to expand the number of possible interactions supported by the interface, considering, for example, the integration of new input devices (like smartphones or smartwatches, with their embedded gy-

roscopes, accelerometers, microphones, etc.), other sensors (to manage, e.g., body gestures and/or voice commands, etc.), or richer hand gestures (like gestures performed with both the hands). The possibility to manage a wider set of conditions and actions (like the change of the object's size and shape, etc.) will be also considered. Based on the feedback received, a new software component will be implemented in order to let users obtain a preview of the interactive application being created directly within the VSE, without the need to export the logic being created to the graphics engine for visualizing it. Efforts will be put in developing custom import scripts able to translate the XML file generated by the tool into data required for reconstructing scenes into other real-time 3D graphics engines, like Unity or Unreal Engine. Finally, since in this paper the focus was put more or development than on creative aspects, a more in-depth investigation of the tool in the latter dimension will be required.

References

Aghaee, S., Pautasso, C., 2014. End-user development of mashups with naturalmash. Journal of Visual Languages & Computing 25, 414–432.

Alexander, J., Barton, J., Goeser, C., 2013. Transforming the art museum experience: Gallery One, in: Proceedings of Museums and the Web.

Andreoli, R., Corolla, A., Faggiano, A., Malandrino, D., Pirozzi, D., Ranaldi, M., Santangelo, G., Scarano, V., 2018. A framework to design, develop, and evaluate immersive and collaborative serious games in cultural heritage. Journal on Computing and Cultural Heritage 11, 4.

Ardito, C., Buono, P., Desolda, G., Matera, M., 2018. From smart objects to smart experiences: An end-user development approach. International Journal of Human-Computer Studies 114, 51–68.

Ardito, C., Costabile, M.F., Desolda, G., Matera, M., Piccinno, A., Picozzi, M., 2012. Composition of situational interactive spaces by end users: A case for cultural heritage, in: Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design, pp. 79–88.

Barricelli, B.R., Cassano, F., Fogli, D., Piccinno, A., 2019. End-user development, end-user programming and end-user software engineering: A systematic mapping study. Journal of Systems and Software 149, 101–137.

Bekele, M.K., Pierdicca, R., Frontoni, E., Malinverni, E.S., Gain, J., 2018. A survey of augmented, virtual, and mixed reality for cultural heritage. Journal on Computing and Cultural Heritage 11, 7.

Billard, A., Calinon, S., Dillmann, R., Schaal, S., 2008. Robot Programming by Demonstration. Berlin, Heidelberg. pp. 1371–1394.

Broll, B., Lédeczi, A., Volgyesi, P., Sallai, J., Maroti, M., Carrillo, A., Weeden-Wright, S.L., Vanags, C., Swartz, J.D., Lu, M., 2017. A visual programming environment for learning distributed programming, in: Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, pp. 81–86.

Brooke, J., 1996. SUS - A quick and dirty usability scale. Usability Evaluation in Industry 189, 4–7.

Burnett, M.M., Scaffidi, C., 2014. The Encyclopedia of Human-Computer Interaction. The Interaction Design Foundation, Aarhus, Denmark.

Bustillo, A., Alaguero, M., Miguel, I., Saiz, J.M., Iglesias, L.S., 2015. A flexible platform for the creation of 3D semi-immersive environments to teach cultural heritage. Digital Applications in Archaeology and Cultural Heritage 2, 248–259.

Cassidy, B., Sim, G., Robinson, D.W., Gandy, D., 2019. A virtual reality platform for analyzing remote archaeological sites. Interacting with Computers 31, 167–176.

Christou, C., Angus, C., Loscos, C., Dettori, A., Roussou, M., 2006. A versatile large-scale multimodal VR system for cultural heritage visualization, in: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, pp. 133–140.

Danado, J., Paternò, F., 2014. Puzzle: A mobile application development environment using a jigsaw metaphor. Journal of Visual Languages & Computing 25, 297–315.

De Pace, F., Manuri, F., Sanna, A., Zappia, D., 2019. A comparison between two different approaches for a collaborative mixed-virtual environment in industrial maintenance. Frontiers in Robotics and AI 6, 18.

Desolda, G., Ardito, C., Matera, M., 2017. Empowering end users to customize their smart environments: model, composition paradigms, and domain-specific tools. ACM Transactions on Computer-Human Interaction 24, 1–52.

Ellis, T.O., Heafner, J.F., Sibley, W., 1969. The GRAIL language and operations. Technical Report. RAND Corp. Santa Monica, CA.

Fowler, A., Fristce, T., MacLauren, M., 2012. Kodu game lab: A programming environment. The Computer Games Journal 1, 17–28.

Francese, R., Risi, M., Tortora, G., 2017. Iconic languages: Towards end-user programming of mobile applications. Journal of Visual Languages & Computing 38, 1–8.

Gault, P., Masthoff, J., Johnson, G., 2015. Dicer: A distributed consumer experience research method for use in public spaces. International Journal of Human-Computer Studies 81, 49–71.

Ghiani, G., Manca, M., Paternò, F., Santoro, C., 2017. Personalization of context-dependent applications through trigger-action rules. ACM Transactions on Computer-Human Interaction 24, 1–33.

Ghiani, G., Paternò, F., Spano, L.D., 2009. Cicero designer: An environment for end-user development of multi-device museum guides, in: International Symposium on End User Development, pp. 265–274.

Hart, S.G., Staveland, L.E., 1988. Development of nasa-tlx (task load index): Results of empirical and theoretical research, in: Advances in Psychology. volume 52, pp. 139–183.

Herrmann, H., Pastorelli, E., 2014. Virtual reality visualization for photogrammetric 3D reconstructions of cultural heritage, in: International Conference on Augmented and Virtual Reality, pp. 283–295.

Ibrahim, N., Ali, N.M., 2018. A conceptual framework for designing virtual heritage environment for cultural learning. Journal on Computing and Cultural Heritage 11, 11.

Ioannidou, A., Repenning, A., Webb, D.C., 2009. Agentcubes: Incremental 3D end-user development. Journal of Visual Languages & Computing 20, 236–251.

Jeon, M., Fiebrink, R., Edmonds, E.A., Herath, D., 2019. From rituals to magic: Interactive art and HCI of the past, present, and future. International Journal of Human-Computer Studies .

Jo, D., Kim, G.J., 2019. Iot+ ar: pervasive and augmented environments for digi-log shopping experience. Human-centric Computing and Information Sciences 9, 1.

Jost, B., Ketterl, M., Budde, R., Leimbach, T., 2014. Graphical programming environments for educational robots: Open Roberta-yet another one?, in: Proceedings of the IEEE International Symposium on Multimedia, pp. 381–386.

Kahn, K., 1995. Metaphor design. Case study of an animated programming environment, in: Proceedings of the 1995 Computer Game Developer Conference.

Kato, S., Tominaga, H., 2009. A practical lesson of introductory programming exercise with Lego robot control and game project, Proceedings of the Association for the Advancement of Computing in Education. pp. 1747–1752.

Kim, H., 2010. Effective organization of design guidelines reflecting designer's design strategies. International Journal of Industrial Ergonomics 40, 669–688.

Kim, H., Yoon, W.C., 2005. Supporting the cognitive process of user interface design with reusable design cases. International Journal of Human-Computer Studies 62, 457–486.

Kumar, R., Talton, J.O., Ahmad, S., Klemmer, S.R., 2011. Bricolage: example-based retargeting for web design, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2197–2206.

Lamberti, F., Cannavò, A., Montuschi, P., in press. Is immersive virtual reality the ultimate interface for 3D animators? IEEE Computer Magazine , 1–7.

Lee, B., Srivastava, S., Kumar, R., Brafman, R., Klemmer, S.R., 2010. Designing with interactive example galleries, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2257–2266.

Leonardi, N., Manca, M., Paternò, F., Santoro, C., 2019. Trigger-action programming for personalising humanoid robot behaviour, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, pp. 1–13.

Lind, M., Skavhaug, A., 2012. Using the Blender Game Engine for real-time emulation of production devices. International Journal of Production Research 50, 6219–6235.

Liu, C.L., 2014. A study of detecting and combating cybersickness with fuzzy control for the elderly within 3D virtual stores. International Journal of Human-Computer Studies 72, 796–804.

Lizcano, D., López, G., Soriano, J., Lloret, J., 2016. Implementation of end-user development success factors in mashup development environments. Computer Standards & Interfaces 47, 1–18.

Loke, L., Robertson, T., 2009. Design representations of moving bodies for interactive, motion-sensing spaces. International Journal of Human-Computer Studies 67, 394–410.

Maceli, M.G., 2017. Tools of the trade: A survey of technologies in end-user development literature, in: International Symposium on End User Development, pp. 49–65.

MacLaurin, M.B., 2011. The design of Kodu: A tiny visual programming language for children on the Xbox 360, in: ACM Sigplan Notices, pp. 241–246.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E., 2010. The Scratch programming language and environment. ACM Transactions on Computing Education 10, 16.

McDermott, F., Maye, L., Avram, G., 2014. Co-designing a collaborative platform with cultural heritage professionals, in: Proceedings of the 8th Irish Human–Computer Interaction Conference, pp. 18–24.

Menestrina, Z., De Angeli, A., 2017. End-user development for serious games, in: New Perspectives in End-User Development. Springer, pp. 359–383.

Nakanishi, H., 2004. Freewalk: A social interaction platform for group behaviour in a virtual space. International Journal of Human-Computer Studies 60, 421–454.

Nielsen, J., 1995. 10 usability heuristics for user interface design. Nielsen Norman Group .

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., Lahmine, S., 2015. Learning basic programming concepts by creating games with Scratch programming environment. Procedia Social and Behavioral Sciences 191, 1479–1482.

Paternò, F., 2013. End user development: Survey of an emerging field for empowering people. ISRN Software Engineering 2013.

Pinto-Llorente, A.M., Casillas-Martín, S., Cabezas-González, M., García-Peñalvo, F.J., 2018. Building, coding and programming 3D models via a visual programming environment. Quality & Quantity 52, 2455–2468.

Polvi, J., Taketomi, T., Moteki, A., Yoshitake, T., Fukuoka, T., Yamamoto, G., Sandor, C., Kato, H., 2018. Handheld guides in inspection tasks: Augmented reality versus picture. IEEE Transactions on Visualization and Computer Graphics 24, 2118–2128.

Protopsaltis, A., Auneau, L., Dunwell, I., de Freitas, S., Petridis, P., Arnab, S., Scarle, S., Hendrix, M., 2011. Scenario-based serious games repurposing, in: Proceedings of the 29th ACM international conference on Design of communication, pp. 37–44.

Rubino, I., Barberis, C., Xhembulla, J., Malnati, G., 2015. Integrating a location-based mobile game in the museum visit: Evaluating visitors' behaviour and learning. Journal on Computing and Cultural Heritage 8, 15.

Sáez-López, J.M., Román-González, M., Vázquez-Cano, E., 2016. Visual programming languages integrated across the curriculum in elementary school: A two year case study using Scratch in five schools. Computers & Education 97, 129–141.

Samsel, F., Klaassen, S., Rogers, D.H., 2018. Colormoves: Real-time interactive colormap construction for scientific visualization. IEEE Computer Graphics and Applications 38, 20–29.

Sanna, A., Lamberti, F., Bazzano, F., Maggio, L., 2016. Developing touchless interfaces to interact with 3D contents in public exhibitions, in: Proceedings of the International Conference on Augmented Reality, Virtual Reality and Computer Graphics, pp. 293–303.

Schmalstieg, D., Stork, A., 2019. Unified patterns for realtime interactive simulation in games and digital storytelling. IEEE Computer Graphics and Applications 39, 100–106.

Serim, B., Ahmed, I., Ylirisku, S., 2015. Extreme co-design: Prototyping with and by the user for appropriation of web-connected tags, in: 5th International Symposium on End-User Development, p. 109.

Song, M., Elias, T., Martinovic, I., Mueller-Wittig, W., Chan, T.K., 2004. Digital heritage application as an edutainment tool, in: Proceedings of the ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry, pp. 163–167.

Spahn, M., Wulf, V., 2009. End-user development of enterprise widgets, in: International Symposium on End User Development, pp. 106–125.

Stolee, K.T., Fristoe, T., 2011. Expressing computer science concepts through Kodu game lab, in: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, pp. 99–104.

Stratton, A., Bates, C., Dearden, A., 2017. Quando: Enabling museum and art gallery practitioners to develop interactive digital exhibits, in: International Symposium on End User Development, pp. 100–107.

Sutherland, I.E., 1964. Sketchpad - A man-machine graphical communication system. Simulation 2, R–3.

Swearngin, A., Dontcheva, M., Li, W., Brandt, J., Dixon, M., Ko, A.J., 2018. Rewire: Interface design assistance from examples, in: Proceedings of th e 2018 CHI Conference on Human Factors in Computing Systems, pp. 1–12.

Tetteroo, D., Markopoulos, P., Valtolina, S., Paternò, F., Pipek, V., Burnett, M., 2015. End-user development in the Internet of Things era, in: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, pp. 2405–2408.

Valderas, P., Pelechano, V., Pastor, O., 2006. Towards an end-user development approach for web engineering methods, in: International Conference on Advanced Information Systems Engineering, pp. 528–543.

Ventura, M., Ventura, J., Baker, C., Viklund, G., Roth, R., Broughman, J., 2015. Development of a video game that teaches the fundamentals of computer programming, in: Proceedings of the Annual IEEE Region 3 Technical, Professional, and Student Conference, pp. 1–5.

Vi, C.T., Ablart, D., Gatti, E., Velasco, C., Obrist, M., 2017. Not just seeing, but also feeling art: Mid-air haptic experiences integrated in a multisensory art exhibition. International Journal of Human-Computer Studies 108, 1–14.

Waldon, S.M., Thompson, P.M., Hahn, P.J., Taylor II, R.M., 2014. Sketchbio: A scientist's 3D interface for molecular modeling and animation. BMC Bioinformatics 15.

Weintrop, D., Afzal, A., Salac, J., Francis, P., Li, B., Shepherd, D.C., Franklin, D., 2018. Evaluating coblox: A comparative study of robotics programming environments for adult novices, in: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, pp. 1–12.

Wojciechowski, R., Walczak, K., White, M., Cellary, W., 2004. Building virtual and augmented reality museum exhibitions, in: Proceedings of the 9th International Conference on 3D Web technology, pp. 135–144.

Yoon, S.Y., Choi, Y.J., Oh, H., 2015. User attributes in processing 3D VR-enabled showroom: Gender, visual cognitive styles, and the sense of presence. International Journal of Human-Computer Studies 82, 1–10.

Zhan, Y., Hsiao, M., 2018. A natural language programming application for Lego Mindstorms EV3, in: 2018 IEEE International Conference on Artificial Intelligence and Virtual Reality, pp. 191–192.

Zyda, M., 2005. From visual simulation to virtual reality to games. Computer 38, 25–32.