A Visual Programming Tool to Design Mixed and Virtual Reality Interactions

Guillaume Loup [1], Sébastien George [1], Iza Marfisi [1] and Audrey Sema [2]

[1] Université Bretagne Loire, Le Mans Université, EA 4023, LIUM, 72085 Le Mans, France
[2] Université de Lyon, CNRS, INSA-Lyon, LIRIS, UMR 5205, F-69621, France

Abstract - **Mixed and Virtual Reality (MVR) devices are now more accessible. However, developing MVR applications is still complex for the majority of developers, because it requires specific expertise. For the past few years, several packaged solutions offered to assist developers who are non-MVR experts. These solutions rarely offer full freedom to create specific interactions adapted to the context. We therefore propose a new MVR tool named *MIREDGE* (*MIxed and virtual REality Development tool for Game Engine*). Its interface allows visual programming of MVR interactions. This solution aims at allowing developers to capitalize, reuse, share and associate interaction algorithms. It also takes into account software and hardware compatibility in order to compose new algorithms. The specific architecture of *MIREDGE* provides opportunities for MVR and non-MVR developers to collaborate to meet a common need: writing efficient MVR interaction algorithms. *MIREDGE Editor* was evaluated by 31 MVR and non-MVR developers. Results shows that *MIREDGE Editor* seems effective and efficient particularly for non-MVR developers.**

Index Terms - **Mixed and Virtual Reality, immersive interactions, script generator, sharing expertise, game engine, multiplatform.**

I. INTRODUCTION

The general public has lately become particularly interested in Mixed and Virtual Reality (MVR) interactions and several Head Mounted Displays (HMDs) are now available and affordable. However, even though the demand for new applications and games that can be played with these technologies is high, still very few developers have the specific MVR skills (Hafner, Hafner, and Ovtcharova 2013).

The implementation of a realistic virtual environment remains a complex process for developers. In recent years, the use of tools dedicated exclusively to virtual reality is limited and developers have decided to use mainly video game development environments (Trenholme and Smith 2008). These environments are reliable and easy to use, with good performances. They are inexpensive and allow users to create extensions that improve the interface themselves. In many cases, developers also share projects with documentation and source code so that other users can create new content. These game engines make low-cost virtual reality more accessible. Hilfert and Knig (2016) consider that these game engines are sufficient to allow non-developers creating immersive virtual environments. Despite this, there are still many prerequisites to create an immersive application with interactions adapted to specific user's needs. Even if some game engines allow to develop applications without knowing a programming language, mastering the logic of programming remains essential. Our approach is based on the assumption that a developer, without training in MVR but assisted by a tool based on the knowledge of MVR experts, is able to develop an immersive application adapted to the user's needs.

Although a developer and virtual reality developer may have different skills and various design approaches, their communities are influencing each other and they are beginning to use common tools such as Unity3D (Zyda 2005). So, it may be relevant to consider that the same tool could meet the expectations of both communities. One priority of the non-MVR developer community is to simplify the implementation of interactions. Ideally, this simplification makes development more accessible without restricting the range of possibilities. Concerning the community of MVR developers, their priority is to optimize their process of writing interaction algorithms. This optimization should make it possible to write their solution faster, while ensuring reliable and efficient results.

This paper aims at proposing an approach and evaluating a tool - named *MIREDGE Editor* - to design and code immersive interactions based on algorithms already considered to be efficient. Furthermore, this solution becomes more attractive if its tools are free software and its compatibility with peripherals is important.

In the next section, we will discuss the pros and cons of different existing MVR development solutions. Section 3 describes *MIREDGE* principles. The evaluation of *MIREDGE Editor* is detailed in section 4. Finally, the limitations of the tool and its future evaluations are discussed in sections 5 and 6.

Oculus Rift (https://www.oculus.com/rift/), HTC VIVE (https://www.vive.com/eu/) and Microsoft HoloLens (https://www.microsoft.com/fr-fr/hololens)

II. Related Work

Today, a large number of tools allow writing MVR interaction algorithms. Some of them are independent environments, while others are dedicated to particular game engines (Kreylos 2008). Moreover, some are intended for MVR developers and others for non-MVR developers. In order to understand the variety of existing tools, we proposed to classify them into four categories:

- Development platforms dedicated to MVR
- Game engines with MVR manufacturers' libraries
- Game engines with middleware
- Game engines with assistance tools

For each of these categories, we highlight their advantages and disadvantages regarding the following criteria (Ritter, Borst, and Chambers 2015):

- Usability: can non-MVR developers easily use the tool?
- Script capitalizing: is it possible to capitalize scripts written by MVR developers?
- Collaboration: does the tool allow MVR and non-MVR developers to collaborate in order to create scripts? (Sowe, Stamelos, and Angelis 2008)
- Script re-use: is it possible to re-use scripts written during previous projects?
- Devices compatibility: when will the tool be compatible with new devices?
- Cost.

2.1 Development Platforms Dedicated to Mixed and Virtual Reality

The oldest development platforms fully dedicated to MVR allow programming interactions and staging 3D scenes *via* a single interface as illustrated in Figure 1. Such platforms were developed 30 years ago, when virtual reality was mainly associated with military, entertainment or education needs.

Pros. The programming interface is often designed to accommodate different levels of expertise. For example, the most popular platforms, *Virtools* and *Eon studio*, offer visual programming (Yingyan, Jun-sheng, and Zhijun 2009). To give MVR developers more freedom, *Virtools* also allows them to script their own programming blocks.

Cons. These platforms have a high acquisition cost and are therefore intended for a limited number of specialized developers. Furthermore, it is often necessary to wait or even pay for the platforms to be compatible to new software and hardware devices.

2.2 Game Engines with MVR Manufacturers' libraries

Ten years ago, only a few game studios had the means to purchase game engines. Today, the new generation of game engines, such as *Unity3D* and *Unreal Engine*, are less expensive or even free (Eberly 2007). These real-time 3D engines have begun to satisfy the general public and have been used to create many popular games on smartphones and game consoles (Juul 2012). To facilitate the work of developers, famous manufacturers such as *Oculus* and *HTC Vive* offer libraries for these game engines.

Pros. Most of the new libraries are free and allow direct access to the MVR hardware (Figure 2). The libraries allow interacting with the most popular MVR and common devices (e.g. *HoloLens*, *HTC Vive*, *Kinect*, joystick, mouse). Moreover, manufacturers provide access to low-level information coming from the hardware allowing developers to write custom algorithms.

Cons. The use of these libraries requires to understand specific and technical instructions that are not easy for non-MVR developers. For example, setting the helmets parallax or understanding how to find the arm orientation among the data provided by the Kinect (player id, articulation names, angle radiant).

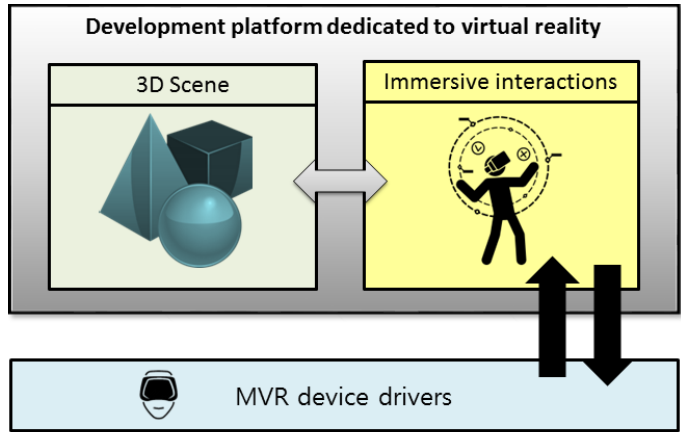

Figure 1: Structure of development platforms dedicated to virtual reality.

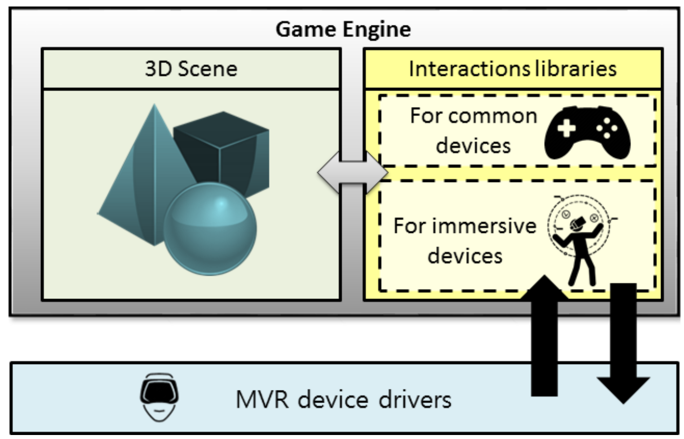

Figure 2: Structure of Game Engines with MVR Manufacturers' libraries.

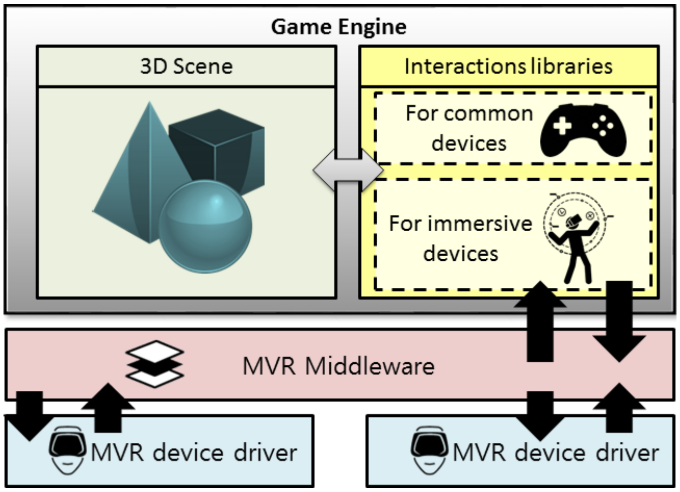

Figure 3: Structure of Game Engines with MVR Middleware.

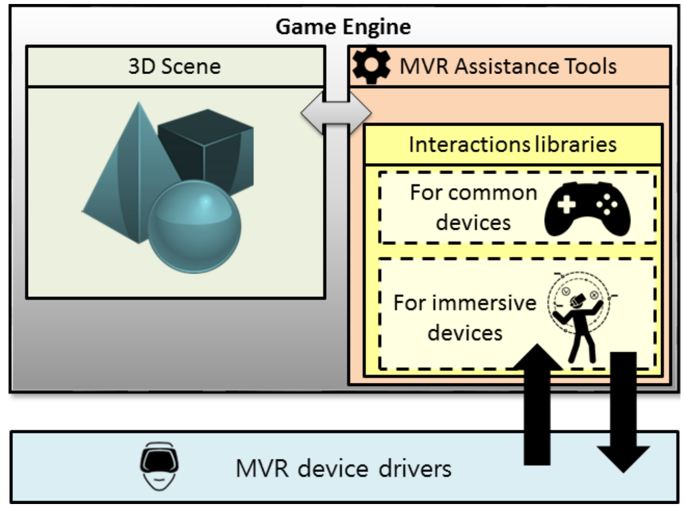

Figure 4: Programming interactions in Game Engines with MVR Assistance Tools.

2.3 Game Engines with MVR Middleware

MVR middlewares ensure the transfer of data between a wide range of devices and the final application.
Pros. The main advantage of middlewares is to unify the different exchange protocols required by each device. All the MVR interactions can be implemented uniformly, without direct communication with the drivers (Figure 3). The best example is *VRPN* (*Virtual Reality Peripheral Network*) (Ralph Stelzer and Steger 2014). This middleware offers a simple and efficient method of sharing device information with useful messages through a server client architecture. Other middleware, such as *CaveUDK*, also offer high level interface to facilitate the use of complex devices that are not managed by game engines (Lugrin et al. 2012). *MiddleVR* (Koenig et al. 2014), associated with *Unity3D*, also offers an accessible configuration interface. This solution has already been adopted by many MVR developers. Finally, it should be noted that, as independent applications, middleware offers a wide range of compatible devices.
Cons. Manipulation libraries require knowledge of innovative devices that is too complex for non-MVR developers. Furthermore, if developers wish to re-use their interaction algorithms, they have to duplicate them and manually transfer them from one project to another.

2.4 Game Engines with MVR Assistance Tools

The term MVR assistance tool refers to all the libraries and other tools added to the development environment to simplify development of MVR interactions. Unlike the manufacturers libraries, these tools offer a higher level of design, closer to the interactions than to the hardware (Figure 4). The *Reality-based User Interface System* (*RUIS*) (Takala 2014) has been available for several years and has been the subject of several publications.
Pros. RUIS is cost-free and intended for non-MVR developers. It offers a set of components that are imported directly into the game engine such as *Unity3D*. These components can easily be added to a 3D scene to configure a set of predefined MVR interactions. It works with several types of devices: position trackers (e.g. *Kinect*, *Razer Hydra*, *PlayStation Move*) and display systems (e.g. *Oculus DK2*).
Cons. The tools only offer a set of predefined interactions for each device and are not intended to create new custom interactions.

Let us note that this classification encompasses the most common tools but not all of them. For example, it does not include *MASCARET*, a framework design for research (Chevaillier et al. 2012). *MASCARET* is original in the sense that it allows non-MVR experts to describe MVR interactions based on Unified Modeling Language (UML). The framework also has the specificity of perceiving the avatar of the player and all the virtual object of a scene (e.g. table, cup, pen, paper) as agents. The purpose of this architecture is to trace the users activity and not to offer powerful interactions with several devices.

We note that most of the tools require knowledge of MVR hardware and are therefore not easy to use for non-MVR developers. Only the MVR assistance tools offer the means to customize some basic MVR interactions. The existing tools also offer very little means of capitalizing and re-using scripts. Only the dedicated platforms offer the possibility of adding new blocks of scripts to a common library which facilitates the re-use of scripts within the community. Finally, all of the existing tools also have one major limitation. When a new MVR device placed on the market, the developers have to wait several months, and sometimes pay, for the tool to be upgraded to the next version or for the manufactures to release a library compatible with their game engine. Considering the high demand for MVR applications using the latest devices, this delay is a real concern for companies.

	Dev. platform dedicated to virtual reality	Game Engine with...		
		MVR Manufacturers' libraries	**MVR Middleware**	**MVR Assistance Tools**
Usability	Requires knowledge of MVR hardware	Requires knowledge of MVR hardware	Requires knowledge of MVR hardware	Easy for intended use, difficult for custom use
Script capitalizing	Requires packaging and placing it in an online store	Requires packaging and placing it in an online store	Only the authors of the tool can share	Only the tool authors can share
Collaboration	In many cases with visual programming	Difficult for non-MVR experts to re-use the work of MVR experts	Difficult for non-MVR experts to re-use the work of MVR experts	Possible but in a predefined boundary
Script re-use	A list of blocks is available	Search, duplicate and rewrite scripts	Search, duplicate and rewrite scripts	A list of components is available
Devices compatibility	Requires upgrading and sometimes paying	As fast as the manufacturer can	Requires upgrading	Requires upgrading
Cost	Usually expensive	More and more are free	Free or very expensive	Usually free

Table 1: Analysis for each category of tools.

2.5 Hybrid solution

In order to help developers who have no training in MVR, several tools have been designed. These tools are based on the concept of reusability and device abstraction for MVR frameworks (Steed 2008). The interfaces of these tools propose to connect graphical building blocks (Tramberend 1999). To be compatible with a large number of MVR devices, it is recommended to use a specific development system architecture (Blach et al. 1998) with entity-component (Fischbach, Wiebusch, and Latoschik 2017).
To combine all of the above features, Figueroa et al. (2008) propose the *Interaction Techniques Markup* Language (*InTml*) to facilitate the collaboration of two communities: VR graphic designers and VR developers. VR graphic designers are trained in User Experience and are usually not developers. *InTml* represents the interaction algorithm as a dataflow, allowing MVR designers to understand the logic of the chosen algorithm. This dataflow is then automatically transformed into C++ or Java scripts and allows VR developers to exploit it in their projects. To describe VR interactions, *InTml* is hardware-independent, component-based and uses formal models. New components can be created by MVR developer. This type of solution could facilitate fast prototyping (Knott et al. 2014).
The limit with *InTml*, is that its dataflow representation is intended for VR graphic designers, who are not developers. The representation is therefore very simplified because VR graphic designers have neither the ability nor the need to program the interaction. This tool is therefore not suited for developers, who need to compose their own interaction algorithm. To meet this need, new interfaces, such as Blueprints (Sewell 2015), let non-VR experts program interactions with dataflow. However, these interfaces are limited to a single development environment and to a small number of devices.

III. MIREDGE approach

As we have discussed above, none of the existing tools meet the needs of the non-MVR developers, such as usability, re-use, capitalization and cost-free. Therefore, we propose a hybrid solution including the advantages of each kind of tool.

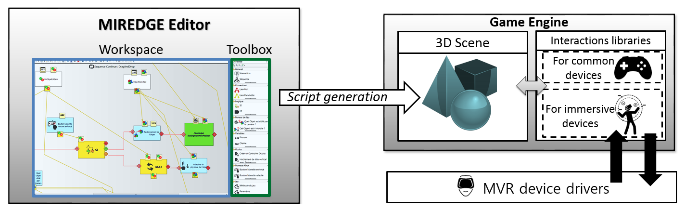

Figure 5: MIREDGE: visual programming and script generation, a hybrid solution.

3.1 System description

Based on the concept of interaction modeling in a virtual environment (Pellens et al. 2007) and on the above-mentioned solutions, we propose MIREDGE, a MIxed and virtual REality DEvelopment tool for Game Engines (Figure 5). The main goal of this tool is to allow MVR and non-MVR developers to re-use interaction algorithms available in a library, to visually program new ones, and generate the corresponding script for the chosen game engine.

3.2 Capitalizing, Sharing and Evolution: a Community based Tool

A new tool providing the source code is more attractive for a developer community. A virtual reality platform, named OSVR, demonstrated that open source could enable collaboration between communities as diverse as academia and industry (Boger, Pavlik, and Taylor 2015). Consequently, MIREDGE is open-source to offer transparency and freedom. One of the main limitations of the existing MVR tools, described in the first part of this paper, is the fact that they are only compatible with a limited number of MVR devices. As shown in Figure 6. MIREDGE offers the possibility for MVR developers to create new components for the algorithms as soon as new devices are released with

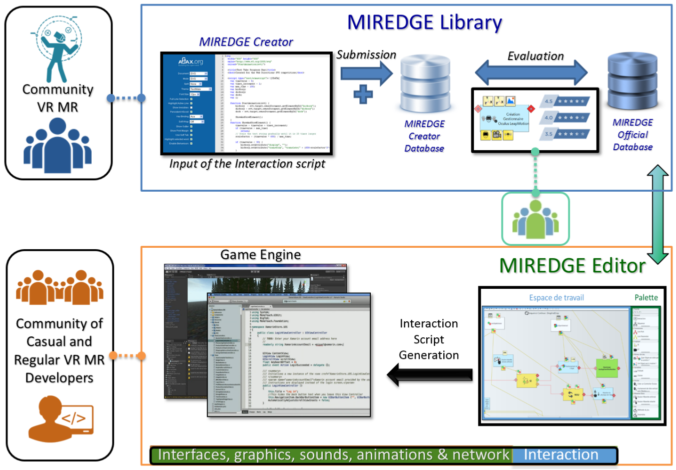

Figure 6: Process for community development of immersive interaction in MIREDGE.

the MIREDGE Creator tool. These new components can be rated by other developers to offer a guarantee of reliability (Plonka et al. 2015), and integrated to the other blocks in the MIREDGE Library. Manufacturers of new technologies can also create new blocks to facilitate the use of their devices.

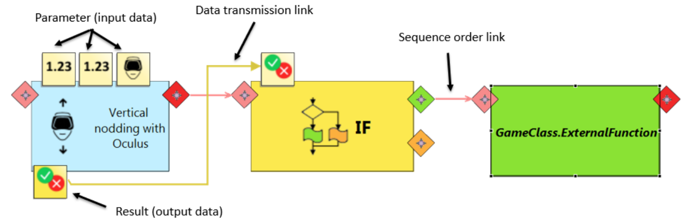

Figure 7: Process for community development of immersive interaction in MIREDGE.

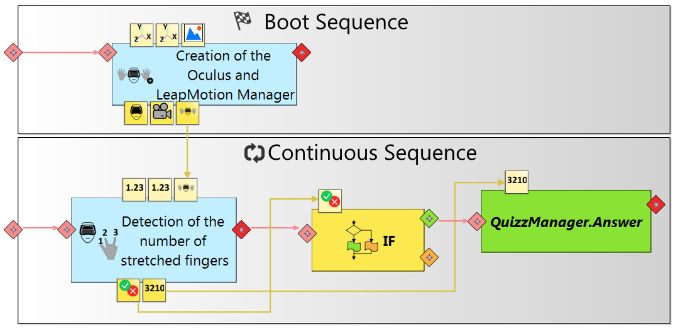

Figure 8: Example of an algorithm in MIREDGE.

3.3 Ease of use: visual programming

MIREDGE Editor allows non-MVR experts to write MVR algorithms by creating visual flowcharts of blocks (Myers 1990) (Figure 7). Despite the simplistic aspect of these blocks, this kind of interface is adapted to developers logic and allows writing a large number of algorithms (Resnick et al. 2009). Each method is represented by a block containing a text and an icon. There are three block categories:

- The first category of blocks (blue blocks) are specific methods for interacting with MVR devices. These methods are used to transmit execution commands to devices and to collect information about their properties, such as the vertical nodding Oculus detection block.
- The second category (yellow blocks) are logical elements. These allow developers to add conditions as well as repetitions, such as IF or WHILE blocks. This category is essential to allow the creation of custom algorithms while remaining simple for non-MVR developers.
- The third category (green blocks) allows developers to refer to variables, classes and methods that already exist in the scripts of the project.

The developers can connect these blocks with two types of links. The first type of links (pink arrows) defines the order in which the block will be executed. The second type of link (yellow arrows) allows to transmit variables from the output of one block to the input of another block.

For example, in Figure 8, the algorithm defines that, when fingers are detected by the LeapMotion device, the Answer method of QuizzManager script within the game engine project will be called. To manage the LeapMotion device, a blue block had to be placed in the initialization sequence. To manage the detection, a blue block is placed in the continuous sequence. This block has two output parameters, one boolean determining if the detection is correct, another indicating the number of fingers detected. Thus the link with the yellow condition block allows to filter according to the quality of the detection status. Finally, the green block represents a specific method from one of the existing scripts of the game engine project.

3.4 A generic script translator

Once the developers have designed their MVR interactions with blocks in *MIREDGE Editor*, they are translated into source code (Biernacki et al. 2008). For example, if the developers are using *Unity3D*, *MIREDGE* converts the dataflow to *Javascript* and *C#* code. The generated scripts are automatically sent to the project chosen by the developer and linked with others resources. The developer can then modify these scripts in their development environment. This feature also has a pedagogical value. Indeed, it gives the non-MVR experts the possibility to read the generated code thanks to the comments.
MVR is an area in which devices, languages and game engines appear every year. It therefore offers a solution that can adapt to this constant renewal of technology. In order to transform the graphical entities into lines of code for different game engines, we consider the algorithms produced under *MIREDGE* as models. Moreover, all these models are derived from the meta-model presented in Figure 9. The main rule is that each graphic block is mapped to different

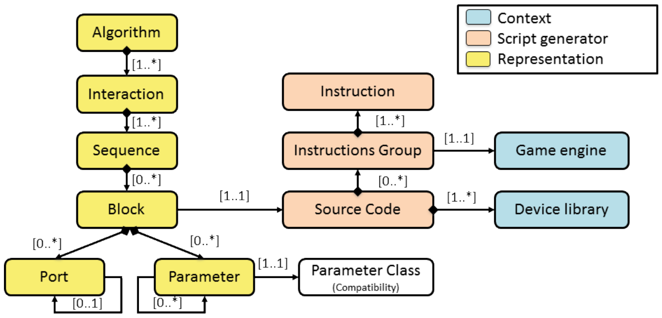

Figure 9: Overview of MIREDGE Immersive Interaction Meta-model.

sets of code lines. These lines allow users to declare or use libraries, methods or variables. They can be attached to one or more devices. There may be variations of the same block in different languages and for different engines.

Evaluation of MIREDGE editor: an exploratory study

4.1 Objective

The study described here aims at evaluating the effectiveness and efficiency of *MIREDGE Editor* (i.e. the part of *MIREDGE* the most used by the two communities), both for MVR and non-MVR expert developers. We wanted to verify that non-MVR experts would be able to use *MIREDGE Editor* easily and develop the required interactions in the allotted time. We also wanted to ensure that MVR experts would not be inconvenienced by *MIREDGE Editor* and would be inclined to contribute to the community-based features that we wish to promote.

4.2 Participants

We asked a panel of developers to add several MVR interactions in an existing project with *MIREDGE Editor*. The first group of fifteen participants (aged from 21 to 27, mean=22, SD=1.5), qualified as the non-MVR expert group, trained to develop serious games but without MVR interactions. The second group consisted of sixteen other participants (aged from 21 to 27, mean=24, SD=1.8) trained to develop MVR applications, qualified as the MVR expert group. The participants came from two backgrounds: the 15 non-MVR participants were in bachelor of serious games, and the 16 MVR participants were in master of virtual reality. In each background, students were selected by the pedagogical supervisor based on their programming grades (i.e. students with the highest grades participated in the study).
Participants of both groups were evenly distributed in the two following experimental conditions:

- With *MIREDGE Editor*: the developers used *MIREDGE Editor* to implement the required interactions and export them to Unity3D (eight non-MVR experts and eight MVR experts)

Figure 10: Capture of the interface related to task 1.

- Without *MIREDGE Editor*: the developers used the *Unity3D* game engine and existing libraries to implement the required interactions (seven non-MVR experts and eight MVR experts). We choose this game engine because it is currently the most used in the game industry and it is the tool that the MVR experts where trained on.

4.3 Equipment and measures

Each participant had a PC with *Unity v5* and access to Internet. Two MVR devices were available: the head mounted display *Oculus Rift DK2*, and the *Leap Motion* controller. This combination offers very specific interactions (Hilfert and König 2016). The device drivers were already installed, and the SDK included in the Unity project.
We measured efficiency and effectiveness using indicators from digital tracks. After completing the task, participants had to list positive feedback and negative feedback (i.e. areas for improvement) regarding *MIREDGE Editor*.

The participants had one hour to implement three MVR interactions for an existing project. We explicitly asked them to implement each interaction one after the other because the tasks were increasingly difficult. The existing project consists of a scene where the player is surrounded by four avatars. Each avatar has a series of questions to ask the player (Figure 10). We asked the developers to implement three MVR interactions for the player to interact with these avatars:

- The first interaction to implement is answering yes or no by shaking the head. This interaction uses the gyroscope of the head mounted display and parameters such as the duration and magnitude of the movements.
- The second interaction to implement allows the players to choose one proposition out of four by holding up the corresponding number of fingers. This interaction uses the Leap Motion controller and parameters related to the duration and the inclination of the fingers.
- The last interaction to implement allows the players to change interlocutor by simply facing another avatar and pointing it with their finger. This interaction uses a combination of data coming from the HMD and the Leap Motion.

Developers without MIREDGE Editor had a library containing methods for detecting head and finger movements. Their final production was a script directly written in *C#* in *MonoDevelop* editor. Likewise, developers using *MIDEGE Editor* had access to blocks corresponding to each of the methods of this library. In the *MIREDGE Editor* workspace,

Group	0 Task	1 Task	2 Tasks	3 Tasks	Total num. of participants
MVR experts without MIREDGE	1	1	3	3	8
MVR experts with MIREDGE	4	2	1	1	8
non-MVR experts without MIREDGE	2	4	1	0	7
non-MVR experts with MIREDGE	2	2	4	0	8
Total number	9	9	9	4	31

Table 2: Number of users who completed tasks

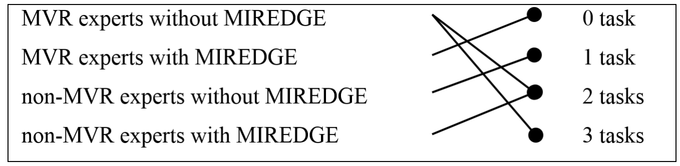

Figure 11: Main attractions based on RD values between the experimental groups and the number of tasks performed by participants.

they had to connect the available blocks to define sequences. Finally, they had to launch the script generation procedure that directly sent the result to Unity. All the participants could test their script as soon as they wanted in order to make the necessary modifications.

4.5 Data analysis

This exploratory study included only 31 participants, and involved appropriate statistics. For categorical variables (e.g. effectiveness measured by successful/unsuccessful task completion (Loup-Escande et al. 2016)), we used Cramers V2 to estimate the magnitude of the association between two categorical variables (Cramer 1999). For numerical variables (e.g. efficiency measured by duration), we used descriptive statistics (i.e. mean, SD, min, max).

4.6 Results

Before the experimentation, an inspection of the tool was carried out by an ergonomic expert. It consisted in ”reviewing *MIREDGE Editor*s interface to verify that it meets a set of ergonomic criteria” (Nogier 2008) (Baccino, Bellino, and Colombi 2005). The recommendations helped us improve *MIREDGE Editor* before the main experimentation. To comply with this protocol and adapt to the availability of users and equipment, it was necessary to conduct all the experiments during only one week. Despite these constraints, no problems have been encountered and all the data generated by the participants could be taken into account in this results of effectiveness and efficiency.

Effectiveness Our main indicator for measuring effectiveness is the percentage of successfully accomplished tasks (i.e. implementation of three MVR interactions). Table 2 shows the number of users who were able to complete zero, one, two or all three of the tasks for each group. We observe that none of the tasks were either too easy or too difficult because they were all be completed by at least one person and never by everyone for each experimental group (MVR experts/non-experts combined with/without *MIREDGE Editor*).

In order to determine if the use of *MIREDGE Editor* significantly helped the developers to accomplish their tasks, we used the Cramers V2 rate . The calculations give a score of 0.14, which shows an intermediate association between the experimental group and the number of tasks performed by participants. In addition, we calculated the Relative Deviation (RD), that measures the association between modalities of two variables (e.g. MVR experts without *MIREDGE Editor* for 1 performed task) (Knott et al. 2014). As depicted in Figure 11, it reveals positive attractions between:

- MVR experts without MIREDGE Editor and 2 or 3 finished tasks (resp. RD=0.29 and RD=0.91)
- MVR experts without MIREDGE Editor and 2 or 3 finished tasks (resp. RD=0.29 and RD=0.91)
- non-MVR without MIREDGE Editor and 1 finished task (RD=0.97)
- non-MVR experts with MIREDGE Editor and 2 finished tasks (RD=0.72)

First of all, this data shows that non-MVR experts managed to accomplish one task without *MIREDGE Editor* whereas they accomplished two with the tool. This show that *MIREDGE Editor* somewhat helped them. In addition, two tasks seem like a good score, considering that MVR experts managed to accomplish 2 to 3 task in the same given time with their usual tools (Unity3D).
The data also shows that MVR experts had difficulty completing the first task with *MIREDGE Editor*. This could be explained by the difference between their practice of this tool and their automatisms acquired on the tool usually used. Changing habits over such a short duration is very difficult. Also, we would like to emphasize the fact that we strictly followed the experimentation protocol and therefore did not intervene during the experimentation, even when the developers seemed to be struggling with the tool. Helping them would certainly have increased their rate of success but we wanted to reproduce real conditions.

Efficiency Completing a task successfully is interesting but, completing it quickly is just as important. We therefore measured the time necessary to develop each MVR interaction. The start time corresponds to the entry of the first

Cramers V2 estimates the magnitude of the association between two categorical variables [30]. It is calculated by dividing phi2 by phi2 max. Phi2 is the average deviation in the table, while Phi2 max is the smallest dimension in the table minus 1. Cramers V2 lies between 0 and 1. The association is conventionally considered as strong when V2 >0.16, as weak when V2 <0.04, and as intermediate between the two scores (Wolf and Corroyer 2004).

Relative Deviation (RD) is calculated on the basis of a comparison between observed and expected frequencies (i.e. those that would have been obtained if there was no association between the two variables), according to the following formula: RD = (observed data - theoretical data) / theoretical data. There is attraction when RD is positive, and repulsion when it is negative. By convention, we retain only RD with absolute terms >0.25.

Group	Mean	Min	Max	Standard deviation
MVR experts without MIREDGE	22min 07s	12min 34s	33min 07s	09min 12s
MVR experts with MIREDGE	30min 18s	17min 39s	42min 56s	17min 53s
non-MVR experts without MIREDGE	36min 35s	28min 32s	43min 49s	07min 40s
non-MVR experts with MIREDGE	25min 10s	20min 03s	33min 13s	05min 51s

Table 3: Duration in seconds to perform task 1

Group	Mean	Min	Max	Standard deviation
MVR experts without MIREDGE	10min 35s	04min 45s	23min 51s	06min 59s
MVR experts with MIREDGE	04min 44s	02min 26s	07min 47s	02min 45s
non-MVR experts without MIREDGE	06min 16s	06min 16s	06min 16s	
non-MVR experts with MIREDGE	03min 06s	01min 55s	04min 26s	01min 13s

Table 4: Duration in seconds to perform task 1

instruction or the placement of the first block. The end time corresponds to the last action necessary for the algorithm to be functional for this task. Tables 3 and 4 show the data to compare tasks 1 and 2. The data in the tables correspond to the completion time of these tasks in minutes and seconds.

We observe that the MVR experts complete the first task faster without *MIREDGE Editor*, whereas the non-MVR experts complete it faster with *MIREDGE Editor*. In short, the non-MVR experts are more efficient with the tool, whereas the MVR developers are more efficient without the tool.

For the second task, non-MVR experts are still more efficient with *MIREDGE Editor*. More interesting: MVR experts, who have gotten used to *MIREDGE Editor*s interface, are also more efficient with *MIREDGE Editor*, than those using Unity.

Feedback For users, *MIREDGE Editor* should have a functionality to identify algorithmic errors. Indeed, a block can be created without any connection to other blocks. Although this will not cause errors in the generated script, it may cause confusion. A connection checker could be to better assist the user. Concerning the positive points, the flow-based programming representation was appreciated by the MVR experts. Here are some of the comments we collected during the experimentation: *the block programming is simpler to visualize than code*, *I linked the simplicity of the terms in the tool and the explanation of the input and output variables*. Non-MVR experts, on the other hand, emphasized the simplicity of the interface: *the interface was easy to understand and the blocks easy to manipulate*, *the tool is easy to use*, *the tools facilitate the creation of interactions*.

V. Discussion

In order for the results to be relevant for a time limited to an hour, we had to provide all participants with equivalent resources and an identical goal. Indeed, in the experimentation without *MIREDGE Editor*, the developers were provided directly with the libraries containing high-level functions adapted to the context. However, in reality, developers have to conduct considerable research to find these functions in previous projects, forums or documentation. Concerning experiments with *MIREDGE Editor*, we asked developers to program the entire algorithm. Normally, *MIREDGE Editor* also allows sharing and thus re-use of existing algorithms. We can therefore assume that the task could have been carried out far more efficiently with access to an algorithm database.

Since the *MIREDGE Editor* training sessions were short and only theoretical, we can assume that longer term experiments could demonstrate greater efficiency and effectiveness of this tool. That is why improving the form and content of application training remains one of our main objectives.

Thus, the concept of reengineering is a process used by developers. As InTml (Figueroa et al. 2008), if a developer modifies a script generated by *MIREDGE Editor*, this will not affect its graphical representation of the algorithm. Consequently, each time a script is generated, the changes made in the text editor will be overwritten. Assuming that these modifications are usually simple customizations, we suppose that MIREDGE Editor can avoid being confronted with this situation.

About debugging, this is necessary whenever errors can be made. Using visual programming, thanks to the logic of the meta-model and various restrictions rules of the interface, the goal is to prevent errors. These limits are partially linked to the fact that *MIREDGE* is an independent application. Therefore, considering *MIREDGE* as a development environment plugin is an option that needs to be studied. In addition, the current block library only contains a limited number of blocks. When the MVR experts will add blocks it will be necessary to implement filters to help developers quickly find the right. These filters could be based on the type of action, such as moving a 3D object, selecting an area or changing the view. We can also take into account the hardware and software that the developers want to use such as the game engine, the input device (e.g. mouse, *Kinect, Leap Motion*) and the output device (e.g. TV screen, augmented reality glasses, the head mounted display).

Finally, even if the experiment was carried out on a very limited number of interactions, the meta-model must theoretically allow a wide range of interactions to be coded. It would thus be interesting to study a larger variety of interaction algorithms such as selection, navigation and manipulation (Weidig et al. 2014) (LaViola Jr et al. 2017). Another question worth asking is whether visual programming will always be as efficient for more advanced interactions (Wonner et al. 2013) or for specific devices such as *Brain Computer Interfaces* (Mercier-Ganady et al. 2013).

VI. Conclusion

Today, developers who are experts in MVR and those who are not experts in MVR often work with the same programming tools but using different methods. Yet, these communities could greatly benefit from each other in order to rapidly produce powerful MVR applications. Indeed, MVR experts have the knowledge to create specific MVR

algorithms for new devices and non-MVR experts can produce large quantities of applications that re-use and combine these algorithms. Our approach therefore consists in proposing *MIREDGE* that responds to this goal by allowing these two communities to collaborate.
MIREDGE has two main advantages. First of all, it allows non-MVR experts to create MVR interactions, without any specific MVR knowledge, by combining blocks with graphical programming. These blocks contain scripts written by MVR experts and allow to interact with MVR devices. *MIREDGE* offers a library of blocks that can be enriched by experts when new devices appear. The second advantage of *MIREDGE* is the fact that it is interoperable with any game engine. Indeed, the tool is based on a meta-model that allows to convert the programming blocks into fully editable script for any game engine.
To validate the contributions of this tool, experiments have showed a gain in efficiency and effectiveness especially for non-MVR developers. New experiments will be conducted on a larger scale to confirm compatibility with maximum interactions, game engines and devices. Also, a study will be done to evaluate all the conditions required so that MVR experts are ready to share their knowledge and so that non-MVR experts are ready to integrate new MVR devices with an assistant such as *MIREDGE*.

ACKNOWLEDGMENTS

This work was funded by the French Research Agency (ANR-13-APPR-0001, JEN.lab project). The authors want to thank all the participants for their key contributions.

REFERENCES

[1] Baccino, T.; Bellino, C.; and Colombi, T. 2005. *Mesure de l'utilisabilité des interfaces*. Herms Sci. - Lavoisier.

[2] Biernacki, D.; Colaço, J.-L.; Hamon, G.; and Pouzet, M. 2008. Clock-directed modular code generation for synchronous data-flow languages. In *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES '08, 121–130. New York, NY, USA: ACM.

[3] Blach, R.; Landauer, J.; Rosch, A.; and Simon, A. 1998. A highly flexible virtual reality system. *Future Generation Computer Systems* 14(3):167 – 178. Virtual Reality in Industy and Research.

[4] Boger, Y. S.; Pavlik, R. A.; and Taylor, R. M. 2015. Osvr: An open-source virtual reality platform for both industry and academia. In *2015 IEEE Virtual Reality (VR)*, 383–384.

[5] Chevaillier, P.; Trinh, T. H.; Barange, M.; Loor, P. D.; Devillers, F.; Soler, J.; and Querrec, R. 2012. Semantic modeling of virtual environments using mascaret. In *2012 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 1–8.

[6] Cramer, H. 1999. *Mathematical Methods of Statistics (PMS-9)*. Princeton University Press.

[7] Eberly, D. H. 2007. *3D game engine design: a practical approach to real-time computer graphics*. CRC Press.

[8] Figueroa, P.; Bischof, W. F.; Boulanger, P.; Hoover, H. J.; and Taylor, R. 2008. *InTml: A Dataflow Oriented Development System for Virtual Reality Applications*. 492–511.

[9] Fischbach, M.; Wiebusch, D.; and Latoschik, M. E. 2017. Semantic entity-component state management techniques to enhance software quality for multimodal vr-systems. *IEEE Transactions on Visualization and Computer Graphics* 23(4):1342–1351.

[10] Hafner, P.; Hafner, V.; and Ovtcharova, J. 2013. Teaching methodology for virtual reality practical course in engineering education. *Procedia Computer Science* 25:251 – 260. 2013 International Conference on Virtual and Augmented Reality in Education.

[11] Hilfert, T., and König, M. 2016. Low-cost virtual reality environment for engineering and construction. *Visualization in Engineering* 4(1):2.

[12] Juul, J. 2012. *A Casual Revolution: Reinventing Video Games and Their Players*. MIT Press. MIT Press.

[13] Knott, T.; Weyers, B.; Hentschel, B.; and Kuhlen, T. 2014. Data-flow oriented software framework for the development of haptic-enabled physics simulations. In *2014 IEEE 7th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, 65–72.

[14] Koenig, S.; Ardanza, A.; Cortes, C.; De Mauro, A.; and Lange, B. 2014. *Introduction to Low-Cost Motion-Tracking for Virtual Rehabilitation*. Berlin, Heidelberg: Springer Berlin Heidelberg. 287–303.

[15] Kreylos, O. 2008. Environment-independent vr development. In *International Symposium on Visual Computing*, 901–912. Springer.

[16] LaViola Jr, J.; Kruijff, E.; McMahan, R.; Bowman, D.; and Poupyrev, I. 2017. *3D User Interfaces: Theory and Practice*. Addison-Wesley Professional.

[17] Loup-Escande, E.; Jamet, E.; Ragot, M.; Erhel, S.; and Michinov, N. 2016. Effects of Stereoscopic Display on Learning and User Experience in an Educational Virtual Environment. *International Journal of Human-Computer Interaction* 1 – 8.

[18] Lugrin, J.-L.; Charles, F.; Cavazza, M.; Le Renard, M.; Freeman, J.; and Lessiter, J. 2012. Caveudk: a vr game engine middleware. In *Proceedings of the 18th ACM symposium on Virtual reality software and technology*, 137–144. ACM.

[19] Mercier-Ganady, J.; Loup-Escande, E.; George, L.; Busson, C.; Marchal, M.; and Lécuyer, A. 2013. Can we use a brain-computer interface and manipulate a mouse at the same time? In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, 69–72. ACM.

[20] Myers, B. A. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1(1):97 – 123.

[21] Nogier, J.-F. 2008. *Ergonomie du logiciel et design web - 4e ed.: Le manuel des interfaces utilisateur*. Dunod.

[22] Pellens, B.; Troyer, O. D.; Kleinermann, F.; and Bille, W. 2007. Conceptual modelling of behaviour in a virtual environment. *International Journal of Product Development* 4(6):626–645.

[23] Plonka, L.; Sharp, H.; van der Linden, J.; and Dittrich, Y. 2015. Knowledge transfer in pair programming: An in-depth analysis. *International Journal of Human-Computer Studies* 73:66 – 78.

[24] Ralph Stelzer, Erik Steindecker, S. A., and Steger, W. 2014. In *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 1B.

[25] Resnick, M.; Maloney, J.; Monroy-Hernández, A.; Rusk, N.; Eastmond, E.; Brennan, K.; Millner, A.; Rosenbaum, E.; Silver, J.; Silverman, B.; and Kafai, Y. 2009. Scratch: Programming for all. *Commun. ACM* 52(11):60–67.

[26] Ritter, K.; Borst, C.; and Chambers, T. 2015. Overview and assessment of unity toolkits for rapid development of an educational vr application. *International Journal for Innovation Education and Research. 3*.

[27] Sewell, B. 2015. *Blueprints Visual Scripting for Unreal Engine*. Blueprints Visual Scripting for Unreal Engine.

[28] Sowe, S. K.; Stamelos, I.; and Angelis, L. 2008. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software* 81(3):431 – 446.

[29] Steed, A. 2008. Some useful abstractions for re-usable virtual environment platform. In *In Proceedings of the IEEE VR SEARIS Workshop*.

[30] Takala, T. M. 2014. Ruis: A toolkit for developing virtual reality applications with spatial interaction. In *Proceedings of the 2Nd ACM Symposium on Spatial User Interaction*, SUI '14, 94–103. New York, NY, USA: ACM.

[31] Tramberend, H. 1999. Avocado: a distributed virtual reality framework. In *Proceedings IEEE Virtual Reality (Cat. No. 99CB36316)*, 14–21.

[32] Trenholme, D., and Smith, S. P. 2008. Computer game engines for developing first-person virtual environments. *Virtual Reality* 12(3):181–187.

[33] Weidig, C.; Mestre, D. R.; Israel, J. H.; Noel, F.; Perrot, V.; and Aurich, J. C. 2014. Classification of VR Interaction Techniques, Based on User Intention. In *EuroVR 2014 - Conference and Exhibition of the European Association of Virtual and Augmented Reality*. The Eurographics Association.

[34] Wolf, M., and Corroyer, D. 2004. *Analyse Statistique des Données en Psychologie (Statistical data analysis in psychology - translated by authors)*. Armand Colin, Paris.

[35] Wonner, J.; Grosjean, J.; Capobianco, A.; and Bechmann, D. 2013. Bubble bee, an alternative to arrow for pointing out directions. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, 97–100. New York, NY, USA: ACM.

[36] Yingyan, T.; Jun-sheng, Z.; and Zhijun, G. 2009. Design and implementation of interactive 3d scenes based on virtools. In *2009 International Forum on Computer Science-Technology and Applications*, volume 2, 87–89.

[37] Zyda, M. 2005. From visual simulation to virtual reality to games. *Computer* 38(9):25–32.

Short bio

Guillaume Loup obtained his PhD in Computer Sciences from Le Mans University. He is interested in the implementation of 2D/3D interactions in real-time, multiplatform implementation and the integration of devices for interacting with virtual environments. His area of expertise is the design of virtual reality-based games.

Short bio

Sébastien George is a Full Professor in Computer Science from Le Mans University. His research deals with the field of Technology Enhanced Learning. He is interested in interactions and communications with context as a central issue: learning context, context of learners or tutors, context for knowledge and skills building.

Short bio

Iza Marfisi is an Associate Professor in Computer Science from Le Mans University. Her research interests are in the domain of Technology Enhanced Learning and include the exploration of innovative educational techniques such as Serious Games for education.

Short bio

Audrey Serna is an Associate Professor in Computer Science at INSA Lyon. Her research area lies at the confluence of computer science (HCI and Usability/Ergonomics) and cognitive science (Cognitive Modeling/User Modeling), to design interactive systems adapted and adaptable to users characteristics.