SimVR-Trei: A Framework for Developing VR-Enhanced Training

Peter Dam[1], Renato Prado[1], Daniel Radetic[1], Alberto Raposo[1], Ismael H. F. dos Santos[2]

1 – Tecgraf Institute/Pontifical Catholic University of Rio de Janeiro

{peter, rdprado, radetic, abraposo}@tecgraf.puc-rio.br

2 – CENPES/Petrobras

ismaelh@petrobras.com.br

ABSTRACT

There has been an increasing interest in using VR and serious games that mock real scenarios and allow professionals to perform operations that previously would only be possible in a real environment. In this kind of VR-enhanced training application, it is possible to identify aspects that are crucial to the quality of the system, which were taken into account in the framework we propose in this paper: SimVR-Trei. The most noticeable, to the end-user, are the user interface (UI) and the 3D navigation/interaction systems, which are key to establishing a valuable experience. Two other crucial aspects in such kind of application are the rules system, paired with the environment simulation, which are responsible for creating a veritable world and enhancing the sense of presence. The rules system is the aspect that relates to the training procedures, in other words, it defines the game logic. The environment simulation creates the behavior of the world in response to the users' actions, which, in some cases, may be provided by an external simulator. Finally, the immersion support system deals with the VR-related issues, such as support for multiple display setups. The challenges imposed by these aspects have been faced during the development of two industrial training applications, leading to the creation of SimVR-Trei framework, which seeks to facilitate the development and the reuse of solutions for common issues in different applications.

Keywords: Training, components, simulation, VR.

1 INTRODUCTION

The use of simulation and serious games as an alternative method to teach, train or reinforce skill acquisition has been subject of study for quite some time [1]. Simulation and games as learning tools have roots that date back to 5000 years ago [2]. Studies prove that computer-simulated environments can effectively help establish the link between theory and practice [3] [4], while providing a safe environment in which to acquire experience [5]. For these reasons, the industry has seen an increased interest in using serious games and simulated virtual environments for training purposes [6] [7] [8].

One key factor in the success of a simulation is what is called "situated learning" [9] [10], which is related to how engaged the user is able to become [11]. For this reason, many initiatives have turned to virtual reality as a potential way to enhance learning [12] [13]. However, introducing virtual reality in a training application and making use of its potential benefits are a challenge [14].

The recent rise of VR and natural user interface (NUI) technologies, from the Wii Remote [15], to Kinect [16] and Oculus Rift [17], has leveraged the use of VR in this kind of application. While in some cases it may be true that "full immersion" is not always necessary [18], it has been found that, for learning purposes, more immersive experiences lead to better results [19] [20]. The reason for this is that VR is able to divert attention away from the real world [21], increasing focus on the content in the virtual environment. This immersion effect is further enhanced by NUI, which may reduce the effort of sending commands to the system [22] [23].

This paper identifies and discusses aspects that are crucial to the quality of a training application and presents a framework to help to overcome common issues. In section 2, we identify the key aspects of VR-enhanced simulations, the role they play and their importance. In the third section, we present a high-level view of the framework's architecture, linking the components to the aspects identified in section 2. At last, we present the usage of the framework in two different industrial training applications.

2 VR-ENHANCED SIMULATIONS

Five aspects have been identified as key to developing a high quality experience in a VR-enhanced training system: immersion support, navigation & interaction, environment, rules, and user interface (UI).

2.1 Immersion Support

In a process that involves learning, a person's capacity to retain new information and knowledge depends on their ability to maintain focused. Studies show that multitasking while learning causes new information to be stored in a different part of the brain [24]. Self-repeating activities carried along with learning leads to storing information in a region specialized in procedures instead of the hippocampus, which is more organized and is easier to retrieve from [25].

One way to help retain focus is increasing the sense of presence. Although a completely immersive virtual environment may not be strictly necessary for the training application at matter, researches show that the more immersive the environment, the more likely for a user to achieve better results [20].

For this reason, immersion support is an important feature to consider in a training application. Immersion support is both the

support for multiple input devices and multiple output devices. We consider that it is important to be able to easily plug in new devices, in order to keep the software up to date with newer technologies.

2.2 Navigation & Interaction

Navigation and interaction are two of the main actions a user can perform in a virtual environment, where the latter can be broken down into selection and manipulation [26]. Studies have shown the importance of developing these techniques [27], which are constantly being improved. In some cases, new technologies may introduce new possibilities, making it crucial that these techniques are able to be seamlessly implemented within the framework.

The way users translate their intentions into commands is another factor that may reduce focus on the task itself. For this reason, the quality of the input system is also an important factor for the success of the learning process. In other words, frequently used commands and tools demanded for fast-response situations should be easier to activate than accessing a setting option that will rarely be modified.

2.3 Environment

Information can be augmented for the user if it is closely related to and tightly coupled with the virtual environment [28]. Furthermore, the virtual environment should provide the user with challenges that require and further develop specific skills while building retainable knowledge to keep after the experience. In addition to this, trainee behavior and attitude towards the given scenarios can be improved aiming for safety and efficiency concerns. Besides, it is also possible to use virtual training to test the ability to solve rare, complex or dangerous situations that could happen in reality.

The environment is not limited to the 3D virtual reality in which training takes place. In this sense, the environment includes the feedback system and user interaction criteria, acting as an agent, as the user itself or the instructor.

A feedback system in this case refers to the data flow in the training environment, which returns human-friendly information to the user. In other words, it does not include the GUI system or peripheral devices operation, but the message treatment to make the environment communicate with the user without ambiguity. Designing such a system demands, but is not limited to, understanding the user's profile and expected behavior, as well as the tasks and procedures being simulated and any technical language required to accurately communicate the steps to conclude the procedure.

As for the interaction criteria, as far as the environment is concerned, it defines which elements can interact with the trainee and under what circumstances. Many objects may exist in a virtual workplace only to help with immersion, making it more real, or to divert attention of those training from their given tasks. For instance a noisy equipment that is along the path of a procedure and can be interacted with, but the procedure requires no actions to be performed on that equipment. To produce a rich and efficient virtual environment, it is recommended to have more possible interactions available than those strictly needed. Care must be taken to ensure mandatory interactions are not harder to engage than they would be in reality.

2.4 Rules

For behavioral training and educational purposes, the rules may reflect only a sparse subset of reality which concerns the content being taught. However, for other trainings, a veritable, close to reality, simulation must be applied in order to demand from its users the knowledge and skills expected in the real life procedures they are learning.

Considering the objectives of the training application, rules must be chosen to enforce the desired level of realism to the simulated procedures. The framework should allow a designer to decide upon the strictness of these rules. For instance, in any given application, it may be desirable that users can be able to skip uninteresting steps of a procedure, in which case the rules will be designed to purposely leave out this portion.

Another case of varying rule set is the way these rules are applied. A critical path of actions that define a task should be required, whilst those that define the quality of the work should remain partially or totally optional. The application can allow the user to complete the whole operation, even if she/he makes a mistake somewhere and inform of this mistake in the end, as long as it isn't critical to the procedure to excel in that specific task. On the other hand, it might be interesting to not allow progress until every requirement for a mandatory step has been met, persisting the relevance of the related actions in the trainee's memory and ensuring the learning and practice of the same.

2.5 User Interface

We consider UI to be a combination of 3D objects in world-space and 2D objects on screen-space. Each one plays an important role in communication with the user. Studies have shown the importance of the UI in virtual environments [29] [30], and the use of 3D objects in what is called ecological interface has been proven [31] to have a positive effect on target acquisition time, which means how long a user takes to find the relevant information. However not all information can be spatial, rendering 2D screen-space elements is useful as well.

The framework should be capable of simultaneously communicating with both types of UI. Different output devices might require different layouts, especially regarding 2D screen-space elements, since the screen-space may be radically different. For this reason the framework should also be able to identify different target outputs and choose the appropriate 2D UI, if provided in the application.

Audio messages and alerts may also be used to reinforce communication. Although it is important to note that it should not be the single source of information to any interface message if audio is optional in the application.

3 Architecture

We based our architecture on the Model-View-Controller design pattern. Variations of it have been around for some time but ours is similar to Cocoa's MVC [32] where the controller acts as a mediator between the model and view layers as shown on Figure 1. The model and view layers are independent and can only communicate back with the controller blindly, which means by delegates and notifications. This way we can increase the objects' reuse and have easily extensible programs.

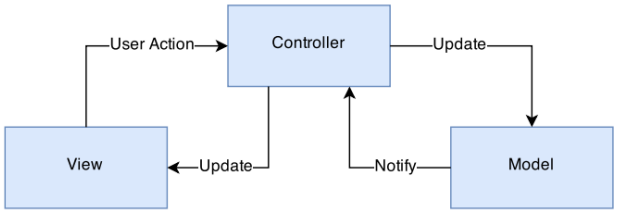

Figure 1: Standard MVC

3.1 MVC in SimVR-Trei

We split views into two groups, the ones related to controlling the game through the GUI (buttons, menus, dialogs, labels and a game inventory or a mini-map, for instance), and the ones that are part of the actual game (waypoints, the player and game elements that need to be updated depending on the game logic or even GUI changes). For this reason, we also have two types of view controllers: the game elements' controller and the GUI controller. Figure 2 presents a high-level view of that architecture. When one of these controllers change the game logic, the other receives a notification about the changes and can update its views.

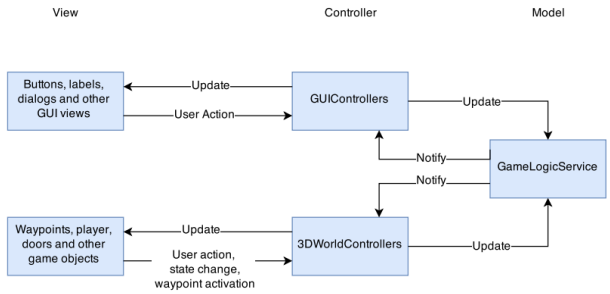

Figure 2: MVC adapted to SimVR

Most of what happens in the game is reported to the logic by a controller. For instance, if the user enters a specific room and an instruction containing what she/he needs to do next needs to be presented on the screen, a 3DWorldController will treat the event generated by a waypoint, which represents the user entering that room, and will inform the logic about it. The action is going to be processed and a notification is going to be sent to the game logic's observers. Then, the GUI controller, which is an observer, will be able to show the instruction text on screen. Another example is the score – if the game has a score system, the logic classes will calculate and update it depending on the user actions during the game.

This allows the system to build multiple presentations of the same information, such as multiple instances of the GUI, which allows us to target different display setups using the same application.

With the chosen Model-View-Controller architecture, the game could run without the 3D part. Although the 3D part is a fundamental aspect of our application, the game could execute in console mode for testing purposes for example. An interesting aspect of this MVC architecture, applied to games, is that one could program the entire logic independently of the chosen game engine and without the need of an advanced GUI, or 3D game objects. Since every user action coming from the GUI or from any in-game actions are reported to the logic, the logic classes can be invoked directly by a test script and then a GUI controller is able inform the results on a console, for instance.

Depending on the content of the training game, the boxes in the diagram in Figure 2 should be broken into several classes. For example, in one project we have one controller for a mini map, one for a thermal camera GUI and one for a report view, which is a view for the user to submit data as well as to show the results at the end of a training session. A main GUI controller manages them and is the only one that has access to the game logic. The same goes for the game logic service: we have an interface called IGameLogic and a GameLogic class that implements it. The GameLogic class delegates its responsibilities to specialized classes.

With the proposed architecture, it is simple to add components such as an external simulator. One of our projects uses an external simulator SDK to measure a solar power plant's performance. The user can change the input parameters from a simulation GUI. Even during the training session, she/he can change simulation parameters from the result of some action started in the 3D world. For instance, the player can travel to a solar panel control box, which is a three dimensional object in the virtual environment, and change the angle of a group of solar panels, which affects the performance of the solar power plant. In Figure 3, it is possible to see how an external simulator can be integrated in the architecture. The simulator is accessed through a simulation service API, preferably an interface.

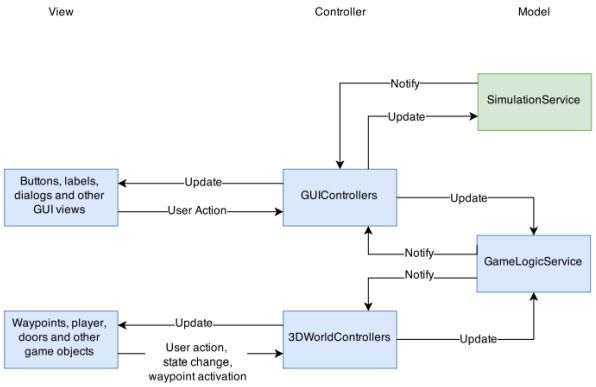

Figure 3: Simulator integration

3.2 Waypoint System

A waypoint is a reference point used for navigation purposes by in-game characters. A basic type of waypoint is the movement waypoint. Game objects assigned to these waypoints will move towards them in sequential order, until they reach the final waypoint. In training modes of serious games a waypoint system might be an important resource to guide the player and the NPCs' – non-player characters, or characters that are not controlled by a player –, movement in a 3D scenario.

It is very common to have those NPCs in a game or a in a serious game. For instance, in one of our applications of a solar power plant, there are workers, who are not controlled by a player, walking around following arbitrary paths determined by waypoints. This is useful for simulating what real workers would be doing in a plant. The NPC waypoints add realism to the scenario that the game represents.

The waypoint system may be fundamental for the game logic since the logic may need to be informed all the time about where the player (or even an NPC) is in the 3D world – if she/he left an important room, reached a checkpoint, or if she/he chose an incorrect path. This way the programmer can deal with the player movement generating responses for the waypoint events.

Another important feature is to be able to limit the user's movement along pre-defined paths, since sometimes it may not be desirable that she/he can explore the entire scene. With the waypoint movement system, it is possible to let the user make decisions about which ways to go, but she/he will only be able to follow the paths previously created with waypoints. There is also the possibility of controlling the characters' speed when they are following a waypoint path.

A waypoint system that meets these needs – guiding and limiting the player and the NPCS's movement in the 3D world – should have at least three types of waypoints. The "choose path" type, which makes the player stop and lets him choose between different paths to follow. The "turn around" type, which makes the player or the NPCs turn around and go back when they reach that kind of waypoint. Finally, the standard type, which forces the player or the NPCs to follow arbitrary paths determined by the application developer.

One other important property of waypoints are the looping type. This defines what happens once an object reaches the last waypoint in a given path. The different behaviors can be "once", "loop", and "ping pong". The first will simply leave the object in place, causing it to become static. The second moves the object back to the first waypoint in the path, starting over from the beginning. The last will cause the object to turn around and move along the path back from the end to the beginning.

We developed our own waypoint system but some engines may already have one available. We created three classes: Waypoint, WaypointMover, and WaypointsHolder. The first one (Waypoint), is a script that contains waypoint visualization options and associated actions – we want to be able to visually place the waypoints when creating the scene. This class stores not only the waypoint's color and size, but also the character's activation distance to it and its type (regular waypoint, "choose path" or "turn around") and looping type. The WaypointMover is the most important script; it controls the object's movement along a waypoint path. It has a WaypointHolder, which contains all the waypoints and renders them as they were connected, as shown in Figure 4.

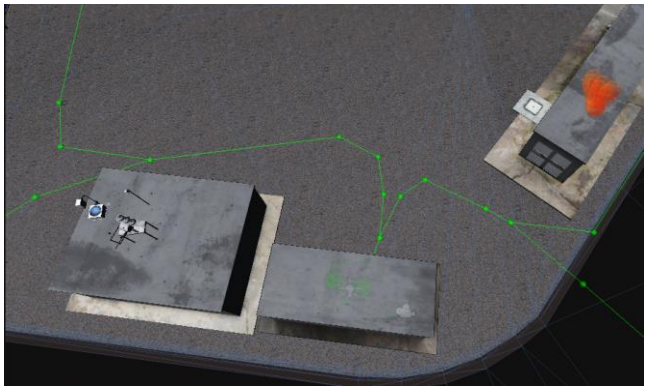

Figure 4: Waypoint paths

3.3 SimVR-Trei and Unity3D

We have successfully integrated this architecture with the Unity 3D game engine [33]. In Unity 3D, every object in the game is called a *GameObject* and each of these game objects can have many different components. The *GameObjects* do not do anything on their own; they are containers for scripts that add different behaviors to them. Scripts that add behavior to a game object are called components. An example of a component would be a script that adds physics behavior to an object. Even though Unity 3D has this component architecture, we were still able to blend our MVC with it and take advantages of both worlds.

To illustrate that, consider our thermal camera view, which is shown in Figure 5. During a training session with one of our applications, the user must make use of the thermal camera to photograph failures on a solar panel. The thermal camera is a game object, which has several components, such as a *Transform* and a *RenderTexture*. The *Transform* component determines the location of the game object in the 3D world; and we use a second camera in the scene, which renders its image (with a thermal effect), to the thermal camera texture. This is a simple way to implement the thermal camera and it is possible thanks to the component architecture. However, the thermal camera view needs more than that: it has buttons for switching from thermal to regular camera mode and a button to take a picture. When the regular camera mode button is pressed, the image effect must change. Furthermore, all of these actions must be reported to the game logic in order to decrease the player's score when he photographs a failure with the wrong camera mode, for example. In other words, the view needs a controller that can receive events and deal with them properly.

Figure 5: Thermal Camera

The solution is adding one more component to the thermal camera with a view controller role, in this case the *ThermalCameraViewController*. In addition to this, it is necessary to connect the game objects that fire events, such as buttons, to the view controller script – which is possible with Unity's GUI framework. This way the Model-View-Controller can be effectively used within the Unity engine.

3.4 Character Control System

In this architecture, we also built a character control system (Figure 6), which is responsible for allowing the user to move around the virtual environment, but also allow non-player characters to do the same. This is achieved by having a core component, the Character Motor, which receives input in a standard format. Using LVRL [34], we read information from devices and transform them, via an

implementation of an Input Manipulator, into the format expected by the Character Motor. This allows a single implementation of the motor to be used by any device or combination of devices. This also enables the creation of different techniques or ways to translate input data into commands, such as different approaches for mapping Kinect gestures to actions in the virtual environment [35].

One such case is the use of Oculus Rift paired with a game pad. While the Rift provides orientation, there is no travel associated with this specific piece of device. By linking a character to both a Rift manipulator and a game pad manipulator, we receive orientation data from Rift and translation data from the game pad, and can easily switch what kind of data and where it is obtained by simply enabling different manipulators. For instance, if the system detects a Kinect, it is possible to switch between game pad and Kinect on the fly.

Furthermore, the state of any given character can be easily obtained from the motor, which is how character animations are dealt with. The Character Animator continuously reads the character's state and updates with the according animation.

One special case is the AI manipulator, which works as any common input manipulator, but instead of reading information from a device a specific behaviour is programmed, which is responsible for feeding information as though it were a device.

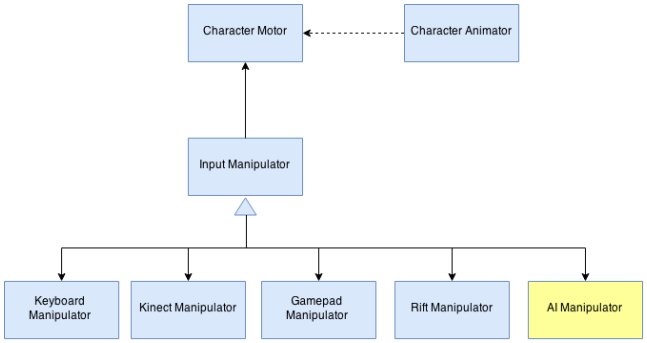

Figure 6: Character control system

3.5 SimVR-Trei Server

An additional feature present in this framework is the possibility to have multiple users in a single training instance. For this to be possible, however, not only must each user be able to see one another, their actions in the environment must be visible to all other users. Every relevant interaction is automatically reported to a server, which contains the state of all variables. The server is responsible for pushing these updates to all clients. The basic layout for the server can be seen in Figure 7.

In this architecture, the executor is responsible for ensuring a proper interaction between the different modules as well as the correct execution of the game rules. It receives and executes commands coming from the training modules and client applications, and guarantees that the variables do not become in an invalid state. It also is responsible for executing the Game Machine module, pushing and retrieving data from any simulators that may be connected.

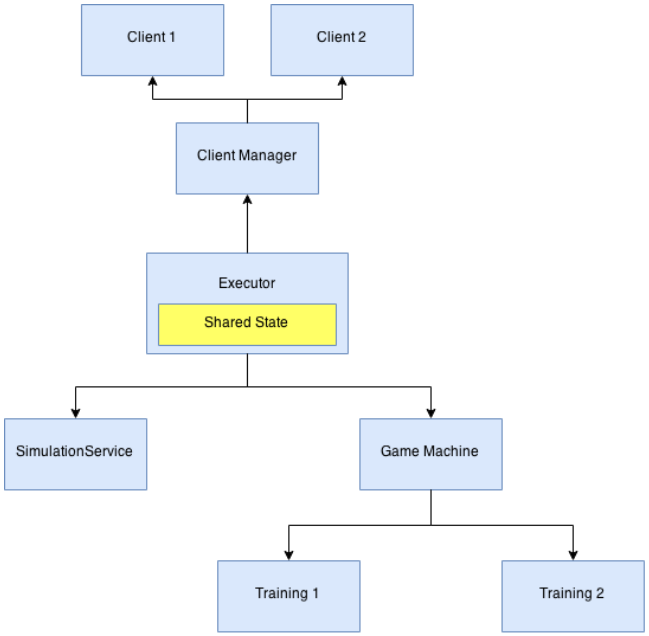

Figure 7: Server architecture

The executor also contains a memory space that can be accessed by the clients and modules, called "shared state". The shared state contains variables that can be created and updated by the training modules and clients, which automatically become available to all other components. These variables are indexed by a key and can hold values as long as they either are basic types or are previously serialized.

These variables hold information of all parts of the system, such as crucial information for representing the state of the simulation, determining the state of the game or storing the location of users in the virtual world.

The client manager controls the communication between server and clients. When a client connects to the server, it must send a "client registration" message. There are two types of permissions: *administrator* and *student*. The *administrator* client can initiate training sessions and the *student* client actively participates in those sessions. This also allows different types of clients to communicate with a single server, sharing the same abstract virtual world among different views. One can connect using a three dimensional application while another can connect using a top-down map-like view, such as a spectator view. The client is responsible for both defining what actions the user may perform in the world and showing the information available in the shared state. Furthermore, there is no hard requirement for how the client is built, since all communication is made either via TCP/IP or through a web service, the same world can be displayed in a web browser, on a desktop application, on a smartphone and so on.

The simulation service allows external simulators to be connected to the system. This service is responsible for feeding information to the simulator and retrieving it, updating the appropriate variables so that all the other components can read it and update themselves accordingly.

The game machine is the component that is responsible for the execution and application of the game rules. Each operation has its own set of rules, which reflect the step-by-step actions the user must perform, as well as the responses to any possible mistakes the user might make.

4 Case Studies

The SimVR-Trei framework is a by-product of the development of two VR-enhanced training applications, where it was employed and continuously improved during the development of both.

4.1 AmbSim

AmbSim (Figure 8) is an application for training operations on an oil rig. The operations strictly follow a manual, as well as employing general safety measures. All actions on the virtual environment are sent to a simulator, which is responsible for providing the behavior of the systems and equipment on the oil rig.

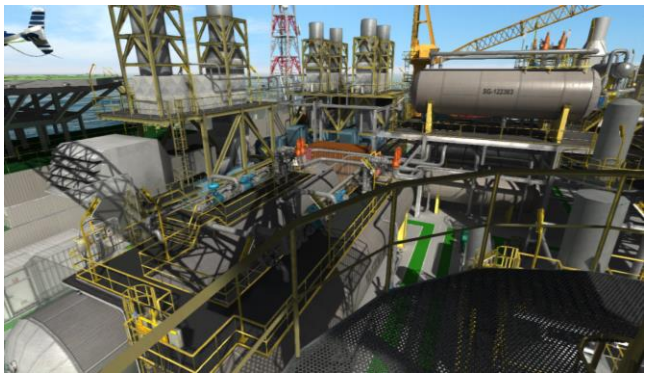

Figure 8: AmbSim screenshot

During the development of this application, the major aspects that were developed were input and output systems, allowing it to be effortlessly built for a large range of physical setups. The environment and rules system were also seen as key factors to the success of the application. While a system was built internally for the rules system, we realized we could use an external simulator for the environment, which is when we built the possibility of connecting simulators to provide the behavior of the environment.

The AmbSim application has two major targets: Kinect setup and Oculus Rift setup. The Kinect setup (Figure 9) uses one large screen and focuses on performing a sequence of actions to successfully complete an operation on the oil rig. The Oculus Rift setup (Figure 10) is the same application, just using different input and output systems. While the Kinect setup is focused on the operation, the Rift setup is focused on exploring and allowing the user to become familiarized with the environment, which was built precisely based on blueprints. Due to this, the Rift setup allows the user to move freely around the environment and, when she/he approaches a certain equipment, she/he can hear technical information about that equipment.

As mentioned in section 3.5, the way the server was built, the same data for a virtual world can be accessed on different devices, and can perform different roles in the training system. In Figure 11 is an example of a web client that is accessible on a smartphone. The role of this client is to allow the user to submit a report to the server. This report is linked to the same session the same user is performing on the desktop or VR client. While in the latter the user should inspect the virtual location for potential failures, in the former the user will report which failures he has encountered. This data is sent to the server and will be processed, sending feedback to the desktop or VR client that the same user is using. This feedback is also available for other clients, such as a supervisor client.

Figure 9: AmbSim using Kinect

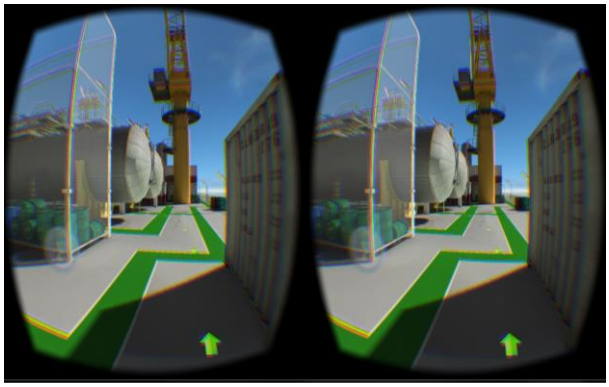

Figure 10: AmbSim using Oculus Rift

Figure 11: Web client for smartphone access

4.2 Solar

Solar (Figure 12) is an application for training routines in a solar power plant. These require the user to have basic knowledge of the plant layout and what types of failures might occur. The application teaches and tests response to potential situations.

Figure 12: Solar Screenshot

In this application we improved the input system, allowing multiple input systems to be built into the same application, and the system will detect which is active to receive proper input from, as well as displaying the appropriate visual cues. The visual cues may be input sensitive, which means that a different graphic may be displayed depending on which input system is active.

Two different images are shown in Figure 13. The left image represents the expected gesture when the input system is using Kinect. The right image represents the expected button input when the input system is using the keyboard. Both perform the same action in the world (move forward).

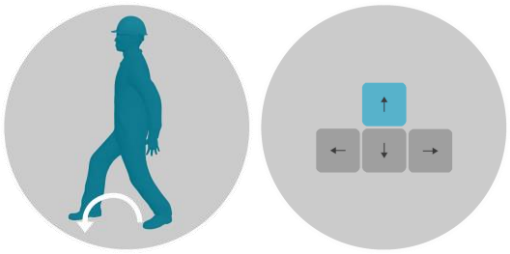

Figure 13: Input-sensitive visual cues

One interesting aspect is that this architecture can be used not only for training applications. An alternative application was built for the Solar project (Figure 14) that allows users to fly and control the date and time to visualize the interaction between the sun and the power plant.

5 Conclusion

This paper presented SimVR-Trei, a framework for developing VR-enhanced training applications. Training applications and VR applications require a series of features, which are present in this framework.

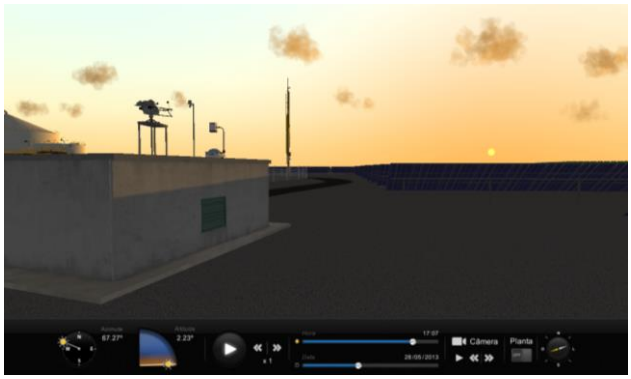

Figure 14: Alternative Solar application

The five key aspects for VR-enhanced training applications we identified in section 2 are covered in the framework as follows.

For *immersion support*, we use the LVRL library [34] to provide communication to input and output devices, paired with a basic system to detect which target setup is active and switch the application in to the proper state.

As for *navigation & interaction*, the framework has a character motor, which allows the use of any techniques and combination of techniques to send data via an *Input Controller* to perform travel, selection and manipulation actions. In addition, a waypoint system was built to allow creation of specific paths, either for the user or for AI-controlled characters. The waypoint system uses the same character controller, making it easy to opt between placing characters on a path or not without having to change any input configuration.

Regarding the *environment*, the framework allows connecting to external simulators, which, via the *Simulator Service*, communicates with the application, receiving and sending data in response to the user's actions. Environment behavior, however, can be programmed directly in the application and should use the same communication infrastructure. This allows a highly customizable and veritable scene to be designed, contributing to an immersive experience.

The *rules* are an important feature for training applications. These are implemented as a module in the *Game Logic*, which is in constant communication with the other modules of the system, evaluating every action performed in the virtual environment. The strictness of these rules can be defined, where a critical path can be created, which are the mandatory parts of a training procedure, while the secondary rules allow quality evaluation. The user might not follow some of the rules and still be allowed to complete the procedure, however the system will track these mistakes, enabling them to be reported at the end of the session.

Finally, the MVC architecture proposes the use of view classes to handle all user communication with the system. All system logic and information is accessible through controllers, while the view is responsible for the layout and display of the information. This is because the *user interface* is very specific to each application. By using this approach the framework allows the same underlying virtual environment, with its rules and behavior, to be displayed in context-sensitive situations, either due to different physical setups or due to different execution modes, for instance, a simple visualization versus a systematic guide for a specific procedure.

The applications in section 4 were built in parallel with SimVR-Trei, where their requirements lead to the creation of new features for the framework. The usefulness of this framework was noticed

when we were able to revisit the applications, retroactively enhancing them with some of the newly available features. Furthermore, with the release of new technologies, we have been able to easily extend both the framework and the applications. The client-server architecture also allows enhancing the application by enabling the creation of remote assisted operations [36].

Even though the framework provides these facilities, it is important to note that the final quality of a VR-enhanced training application ultimately lies in the hands of those responsible for designing them. The proper use of the rules system, the creation of a veritable environment, and an adequate communication with the user are the responsibility of the developers using the framework. It is important to consider aspects of gamification, although it might not be indicated for all cases, it has its benefits and can be successfully used to increase user engagement and improve the learning process.

REFERENCES

[1] G. Andlinger, "Business Games - Play One," *Harvard Business Review,* vol. 36, no. 2, 1958.

[2] J. Wolfe and D. Crookall, "Developing a Scientific Knowledge of Simulation/Gaming," *Simulation & Gaming,* vol. 29, no. 1, pp. 7-19, 1998.

[3] D. Michael and S. Chen, Serious Games: Games That Educate, Train, and Inform, Muska & Lipman/Premier-Trade, 2005.

[4] S. Tobias, J. Fletcher and A. Wind, "Game-Based Learning," in *Handbook of Research on Educational Communications and Technology*, New York, Springer, 2014, pp. 485-503.

[5] H. Haapasalo and J. Hyvönen, "Simulatin Business and Operations Management - A Learning Environment for the Electronics Industry," *International Journal of Production Economics,* vol. 73, no. 3, pp. 261-272, 2001.

[6] VStep, "VSTEP," [Online]. Available: http://vstepsimulation.com/. [Accessed January 2015].

[7] Skills2Learn, "Virtual Reality Case Studies," [Online]. Available: http://www.skills2learn.com/virtual-reality-case-studies.html. [Accessed January 2015].

[8] Siemens, "COMOS Walkinside ITS," [Online]. Available: http://w3.siemens.com/mcms/plant-engineering-software/en/comos-lifecycle/comos-walkinside/walkinside-its/pages/default.aspx. [Accessed January 2015].

[9] J. Lave and E. Wenger, Situated Learning: Legitimate Peripheral Participation, Cambridge University Press, 1991.

[10] L. Lunce, "Simulations: Bringing the Benefits of Situated Learning to the Traditional Classroom," *Journal of Applied Educational Technology,* vol. 3, no. 1, pp. 37-45, 2006.

[11] C. Dede, "Immersive Interfaces for Engagement and Learning," *Science,* vol. 323, no. 5910, pp. 66-69, 2009.

[12] M. Jou and J. Wang, "Investigation of Effects of Virtual Reality Environments on Learning Performance of Technical Skills," *Computers in Human Behavior,* vol. 29, no. 2, pp. 433-438, 2013.

[13] S. Borsci, G. Lawsone and S. Broome, "Empirical Evidence, Evaluation Criteria and Challenges for the Effectiveness of Virtual and Mixed Reality Tools for Training Operators of Car Service Maintenance," *Computers in Industry,* vol. 67, pp. 17-26, 2015.

[14] T. Moher, A. Johnson, S. Ohlsson and M. Gillingham, "Bridging Strategies for VR-based Learning," in *SIGCHI Conference on Human Factors in Computing Systems*, 1999.

[15] Nintendo, "Wii Remote Plus Controller," Nintendo, [Online]. Available: http://www.nintendo.com/consumer/downloads/Wii_Remote_Plus_En.pdf. [Accessed January 2015].

[16] Microsoft, "Kinect for Xbox 360," Microsoft, [Online]. Available: http://www.xbox.com/en-US/xbox-360/accessories/kinect/KinectForXbox360. [Accessed January 2015].

[17] Oculus, "Oculus Rift - Virtual Reality Headset for Immersive 3D Gaming," Oculus, [Online]. Available: https://www.oculus.com/rift/. [Accessed January 2015].

[18] D. Bowman and R. McMahan, "Virtual Reality: How Much Immersion Is Enough?," *Computer,* vol. 40, no. 7, pp. 36-43, 2007.

[19] J. Jacobson, "Digital Dome Versus Desktop Display in an Educational Game: Gates of Horus," *International Journal of Gaming and Computer-Mediated Simulations,* vol. 3, no. 1, pp. 13-32, 2011.

[20] W. Winn, M. Windschitl, R. Fruland and Y. Lee, "When Does Immersion in a Virtual Environment Help Students Construct Understanding?," in *International Conference of the Learning Sciences*, 2002.

[21] J. Lin, H. Duh, D. Parker, H. Abi-Rached and T. Furness, "Effects of Field of View on Presence, Enjoyment, Memory, and Simulator Sickness in a Virtual Environment," in *Virtual Reality*, 2002.

[22] D. Wigdor and D. Wixon, Brave NUI World: Designing Natural User Interfaces for Touch and Gesture, Elsevier, 2011.

[23] R. Francese, I. Passero and G. Tortora, "Wiimote and Kinect: Gestural User Interfaces Add a Natural Third Dimension to HCI," in *International Working Conference on Advanced Visual Interfaces*, 2012.

[24] K. Jimura, F. Cazalis, E. Stover and R. Poldrack, "The Neural Basis of Task Switching Changes With Skill Acquisition," *Frontiers in Human Neuroscience,* vol. 8, 2014.

[25] D. Levitin, The Organized Mind: Thinking Straight in the Age of Information Overload, Penguin, 2014.

[26] D. Bowman, E. Kruijff, J. LaViola Jr and I. Poupyrev, 3D User Interfaces: Theory and Practice, Addison-Wesley, 2004.

[27] D. Bowman and L. Hodges, "Formalizing the Design, Evaluation, and Application of Interaction Techniques for Immersive Virtual Environments," *Journal of Visual Languages & Computing,* vol. 10, no. 1, pp. 37-53, 1999.

[28] D. Bowman, L. Hodges and J. Bolter, "The Virtual Venue: User-Computer Interaction in Information-Rich Virtual Environments," *Presence: Teleoperators and Virtual Environments,* vol. 7, no. 5, pp. 478-493, 1998.

[29] G. Leach, G. Al-Qaimari, M. Grieve, N. Jinks and C. McKay, "Elements of a Three-Dimensional Graphical User Interface," in *Human-Computer Interaction*, 1997.

[30] D. Bowman, E. Kruijff, J. LaViola Jr and I. Poupyrev, "An Introduction to 3D User Interface Design," *Presence: Teleoperators and Virtual Environments,* vol. 10, no. 1, pp. 96-108, 2001.

[31] W. Ark, D. Dryer, T. Selker and S. Zhai, Representation Matters: the Effect of 3D Objects and a Spatial Metaphor in a Graphical User Interface, Springer London, 1998.

[32] Apple, "Cocoa Fundamentals," [Online]. Available: https://developer.apple.com/legacy/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf. [Accessed January 2015].

[33] Unity Technologies, "Unity - Game Engine," Unity Technologies, [Online]. Available: http://www.unity3d.com. [Accessed January 2015].

[34] D. Trindade, L. Teixeira, M. Loaiza, F. Carvalho, A. Raposo and I. Santos, "LVRL: Reducing the Gap Between Immersive VR and Desktop Graphical Applications," *International Journal of Virtual Reality,* vol. 12, no. 1, pp. 3-14, 2013.

[35] P. Dam, P. Braz and A. Raposo, "A Study of Navigation and Selection Techniques in Virtual Environments Using Microsoft Kinect®," in *15th International Conference on Human-Computer Interaction*, Las Vegas, 2013.

[36] D. Medeiros, E. Ribeiro, P. Dam, R. Pinheiro, T. Motta, M. Loaiza and A. Raposo, "A Case Study on the Implementation of the 3C Collaboration Model in Virtual Environments," in *XIV Symposium on Virtual and Augmented Reality*, 2012.