A language for designing nursing training sessions

Student: Sergi Tortosa i Ortiz-Villajos

Advisors: Carlos Andújar Gran,
Antonio Chica Calaf, Marta Fairen Gonzalez

July 4th, 2019

MIRI - Computer Graphics and Virtual Reality

Facultat d'Informática de Barcelona

Polytechnic University of Catalonia - BarcelonaTech

INDEX

1. Abstract

Recent developments in Virtual and Augmented Reality have pushed further the benefits of computer-based training applications. These applications provide learners with a virtual environment where they can develop their skills without the real-world consequences of failing, which makes it especially appropriate in areas such as medicine and flight/vehicle simulations.

The design of training sessions usually requires domain knowledge and thus these sessions should be designed by domain experts. However, computer programming poses a high-entry barrier to non-programmers, and experts may not be able to program or translate these sessions to a computer environment, thus hindering the authoring of training sessions. This master thesis aims to create a system which allows non-programmers to design and reproduce nurse training exercises taking advantage of end user development using visual programming with an associated context.

2. Introduction

Training sessions are widely used on both professional and academic worlds in different scenarios and fields. With training sessions, we have the possibility to transfer or acquire skills that later can be used on real situations. Training sessions can be performed on real scenarios or simulations as real as possible depending on the risk and the cost of this scenario.

Training sessions where a failure can have real serious consequences, such as performing an operation or flying an airplane, cannot be performed on real scenarios and we should find ways to recreate the scenario as realistically as possible but removing the consequences of failure. For instance, on the **pharmacy field**, training a new pharmacist on analyzing compositions of a certain drug has the cost of the laboratory material and necessary equipment, but the failure of the session has no consequences. Another example is **driving lessons** which are completed with a professional also controlling the car, who removes part of the risk of failure by controlling, supervising the session and taking control in case of risk of failure. Another way to transfer these skills without the consequences is by doing video training which, despite being less effective, removes the risk and reproduces real situations.

When simulating a training session on real life, besides the risks of failure, we have to switch the context for each session if this is necessary. When handling two different sessions on the same physical space we should do the setup for each session which is time and resources consuming. One example would be training for the preparation of a surgical table vs training for subdermal injection of some drug.

Computer simulations allow us to create and reproduce training sessions that are similar to real world scenarios removing the risk of failure which makes them suitable for training. With Virtual Reality also raising on last years and being more accessible by end users, we have the possibility to move these computer simulations into a VR environment where the training session can be more realistic. It is also important to mention that with computer simulations we are able to introduce simulated problems onto the training sessions that can make it more difficult to train on not so common situations.

Despite computer simulations being suitable for training sessions, their creation is usually carried out by a developer team which has to create all the environment and the define the behavior that the session involves. The creation of the sessions thus is usually an expensive and time-consuming task. Furthermore, any minimal modification to the session should be carried by the developer team which makes these modifications also costly and time consuming. In addition, as the session should be as realistic as possible, usually a domain expert designs and supervises the development of the session to achieve this realism which makes it even more difficult as the development team usually does not have specific knowledge over the real problem nor about how to proceed beyond the domain expert definition.

Nurse training is a field where the training sessions are important and should be as realistic as possible without the real consequences of failing. On nurse training usually domain experts are the ones who design the training session and they usually do not have the expertise nor knowledge on programming or development of computer applications. But they have general knowledge over computers and are familiar with program interfaces which make them suitable for End User Development.

We thus state that if nurse domain experts could directly design and configure the training sessions, there would be no information loss in the communication of the problem or procedure between experts and application developers. Domain experts will directly modify the session parameters or scenarios which may reduce the cost of creating new computer training sessions which will make them more common and suitable not only for nurse training but for more fields by slightly modifying the session authoring tool.

The rest of this document is organized as follows. Section 3 reviews related work on End User Development and Virtual Reality for training. Section 4 discusses the system requirements and restrictions which will lead us to the proposed solution, which is presented in Section 5. We show the results in Section 6, by authoring a training session with the proposed tool. Finally, we present concluding remarks and discuss future work in Section 7.

3. Related Work

3.1. End User Development

Nowadays people are familiar with the interface and basic functionality of computers and mobile phones. However, most of the users are not be able to develop new applications or modify existing ones as it requires expertise in programming that could not be expected on most users. Nevertheless, users still want to customize, modify or create new behaviors that suit better their necessities. We know that different users have different skills, knowledge, cultural background and cognitive or physiological abilities so taking this into account, we have one fundamental challenge for the upcoming years: developing environments that allow non programmer users to customize, develop or modify their own systems. It is also well known that the number of end users[1][2] is way higher than the number of expertise programmers and requiring the programmers to customize and create per user applications is not feasible, neither on time nor economically.

One of the major fields requiring users to actively define and modify existing behaviors is the Internet of Things. IoT allow users to control remotely devices, create routines for those devices, allow or revoke access to them among many other options. For instance, if a user has an intelligent coffee machine and wants the same coffee every day, when he wakes up the coffee machine serves it, then this users would be able to set the parameters that determine that he has woken up. This simple scenario could get even more complicated by adding the same thing but on weekends, e.g. the user wants the machine to make two coffees instead of one. For this kind of scenarios we need users to be able to program or develop these rules and modify them whenever they want to.

In order to allow users to modify and develop new systems we can offer End User Development which can be defined as “a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact” [2]. End-User Development is already in widespread use in commercial software with success: recording macros in word processors, setting up spreadsheets for calculations and defining email-filters. Not only do these applications show part of End User Development potential but also reveal many flaws and this means that there is work to do on this field.

When we refer to end-users, we are also including domain experts whose domain is not computer programming or similar. When we take into account those domain experts, we could make the most of them by allowing them to change the application behavior [3]. In this way, we will obtain a more economically stable product which fulfills the needs of the domain experts but also the continuously increasing amount of software embedded within consumer and professional products points us to the need of more EUD for a better use of these products.

However, if we look at today's systems the creation of content and the modification of them are difficult for non-professional programmers, despite that amateur programmers are way more empowered than were before. This ends in a well-defined division between the content creators and the consumers where there are more consumers than creators. EUD aims to counterbalance the difference between content creators and consumers. Note that the emerging research of end user development includes fields such as Human Computer Interaction (HCI), Software Engineering (SE), Computer Supported Cooperative Work (CSCW), and Artificial Intelligence (AI); note that each field focus on one part of EUD but all of them have common goals.

As we mentioned before tasks such as configuring email filters are considered EUD which is way different than spreadsheets formulas. For this reason, we should distinguish two types of end-user activities:

- **Parameterization or Customization:** When there are certain options or predefined behaviors given to users that can be chosen.
- **Program Creation and Modification:** Activities that imply some modification, aiming at creating from scratch or modifying an existing software artifact.

When it comes to design a system that should support EUD, we have to take it into account and design the system to support it. For this reason, it is important to determine which EUD paradigm will be used before starting the design of the system. Once we have chosen a paradigm of EUD then we will need to adapt it to our specific case as research on the specific topic may not be available. When we want to apply EUD to authoring training sessions, we will have domain specific problems which most of the times cannot be expected to be solved by domain experts with programming, those should be solved not only by a domain expert but also by a professional programmer who may fulfill the domain expert needs. If the solution that the programmer brings up is too specific, it will be only valid for the designed training sessions and approach a new one will require to repeat the full process.

Instead of designing the session itself, it can also be defined as a set of basic functionalities that later on the domain expert could use to approach new training sessions. This leads us for a challenge in the programming side of designing a system open enough to provide new core functionalities which lets a domain expert define and configure new training scenarios without the later intervention of the programmer.

The task of authoring a training session can be split up into two parts:

- **Session context:** Includes the physical space and the scenario setup of the training session, including the objects trainees would interact with (e.g. surgical equipment).
- **Session flow:** Includes the steps needed to complete or fail the training session.

Note that splitting the process onto two parts allows us to define better which techniques we can use. For defining the session context is not clear that EUD could be useful as the

context may include many variables that include restrictions over the final solution. On the other hand, for defining the session flow EUD seems to fit perfectly as defining a flow constrained by a given context.

3.1.1. Context Aware systems

Context aware systems may be defined as systems can sense their physical environment and adapt their behavior accordingly. Context aware systems that have been evolving the past several years to produce better quality on chat bots or personal assistants like Alexa, Siri or Cortana among others. Users environments contain context information that can be sensed by an application, including location, identity, activity, and the state of nearby people. Context-aware systems involves sensing this context to implicitly provide appropriate information and services.

In order to attach correctly the user actions to their meaning, we could use a well-established context of the action before performing it. Context aware applications have been implemented successfully on other [4], [5]. Additionally, there have been implemented systems in which users describe situations and then attach action to them. [6].

On a smart home for example we may have a context like the one shown in Figure 1. This context is composed by the state of all the connected smart devices, the data of a set of sensors and the location of the devices. With this base we could add to the context the definition of rooms which will allow us to manage a group of devices or we could add the concept of bedtime which will be part of the context and will be defined by a series of conditions. Finally all these defined contexts will help us later on to let users program over them.

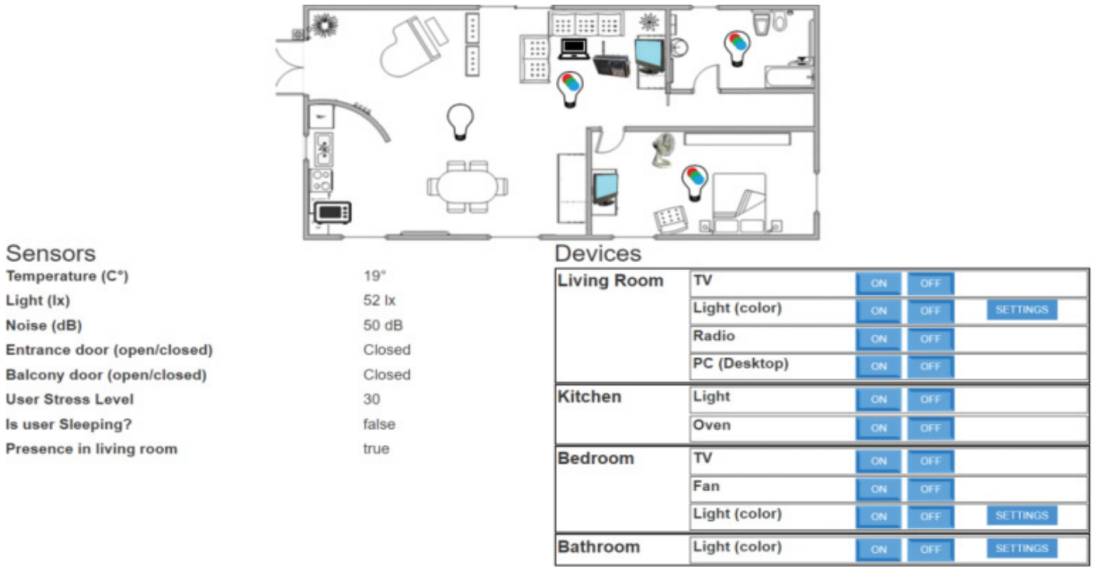

Figure 1 Automatized home context Source: "Personalization of Context-Dependent Applications Through Trigger-ActionRules" [7]

Context aware systems help us to adapt the EUD techniques for authoring training sessions. As mentioned in the Introduction we could split the task into two parts. The first part could be carried out, using a Programming by demonstration approach, and will define the context of the system. The second one will be a context aware system which will modify its behavior and help the final user on the programming task by taking into account the context.

3.1.2. Programming by demonstration

Programming by demonstration is based on an expert user showing how the task is performed and then the end user performing it in the same way. In vocational training this is essential to acquire practical skills. For many people it is much easier to show how it is done than to verbalize it (or even to describe it in a formal language). Programming by demonstration is very common in robotics. It has also been shown that knowing how to attach clear semantics to actions leads in to a valid result.

This approach is the most similar to how humans learn certain tasks. First, we watch how a task is done and then we try to reproduce it in the same context or similar contexts. For the users this is natural because they do the action as they would in real life, but the difficult part is that the system has to know what is important or what is not about the actions executed by the users. For example in an IOT environment, if we try to create a routine of turning all lights when we go out we will record all the lights turning off and the door closing, supposing that it has sensors. Now the problem is to know if the system should record also the hour of the day where it is recorded; if it has to be applied always or if it should be only performed on weekends as it was recorded on a Sunday. The system should know how to solve all these problems.

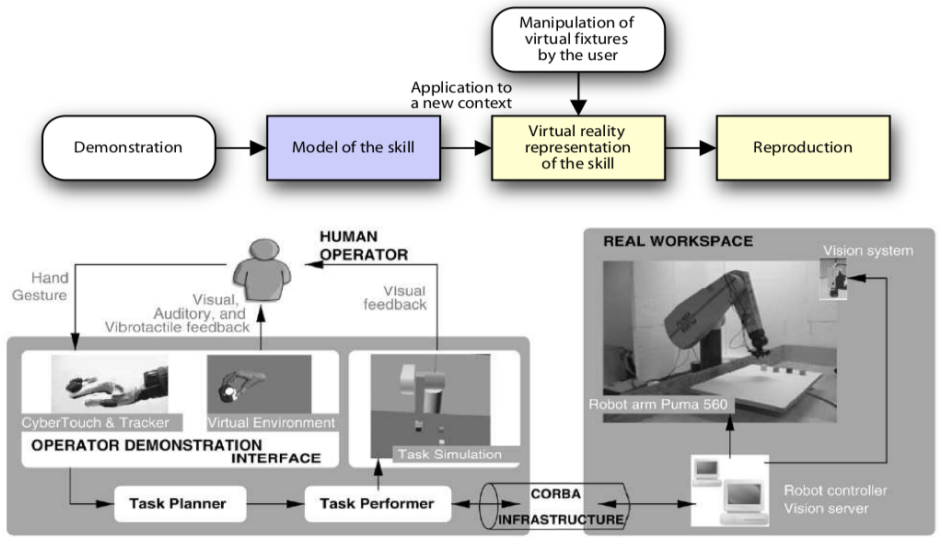

Figure 2 Programming by demonstration on robotics example. Source: "Handbook of Robotics Chapter 59 : Robot Programming by Demonstration" [8]

On the robotics field, programming by demonstration is used, for instance, to record the movements that a robot should do. One example would be the operator performing the action on a Virtual Reality environment and the actions would be recorded and later on reproduced into a real robot Figure 2. First the user will be equipped with a set of trackers to record the movements and then will be asked to perform the task in a virtual environment. Note that the virtual environment may not be an exact replica of the real world and only represent in a schematic way the elements that are needed to perform the task. Once in the virtual environment, the user will be asked to perform the task several times and the data will be recorded by the system. Finally the system will translate the recorded actions to the final task. Figure 2 shows an example on how a human operator will program a robotic arm using Cyber Touch and a Tracker to record user actions.

Another example of programming by demonstration on the field of robotics is try to perform grasp recognition. [9] In order to achieve this the authors setup a virtual environment with different objects which vary on size and form. They provide the users with gloves with sensors and ask them to grab different objects. During the process they record the contact points of the fingers and the objects and the positions and orientations of the hand. After recording this for all the different objects among several users they try to build a model on how objects with certain properties should be handled by a robotic hand. On Figure 3 we can see the final mapping between the user recorded hand position and the robotic hand. As it can be seen, when moving the positions and gestures of the user hand to the robotic one they should perform some corrections and transformations as they are morphologically different.

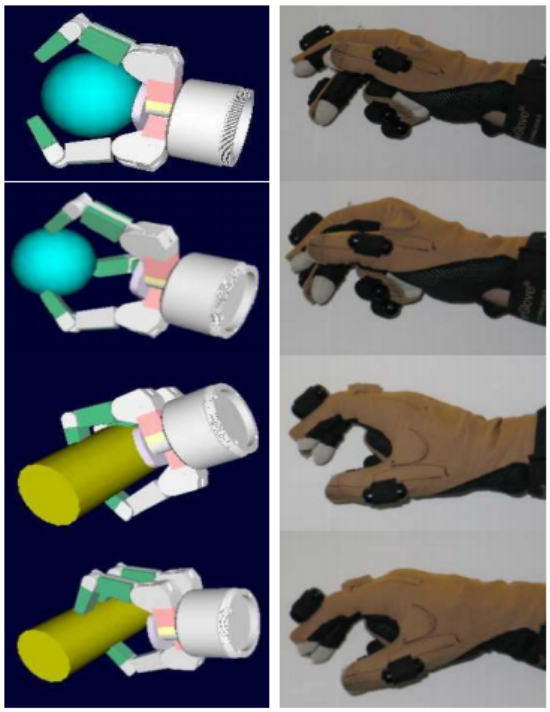

Figure 3 Grasp recognition. Source: "Grasp recognition in virtual reality for robot pregrasp planning by demonstration" [9]

As it can be derived from the two examples above, context plays an important role in programming by demonstration. In the first example, the context is defined by the environment setup and the initial state of the system. Whereas in the second example the context is given by the object that the user should grab as we want to act differently depending on which object is grabbed.

Programming by demonstration can be applied to authoring training sessions following a similar workflow that the one shown in the examples. For each session or set of sessions we should define a well-known context and then record the user performing the task. Defining the context, as we have seen in the previous section, ideally will be defined when we reach the EUD and therefore we could apply this technique by recording the domain expert actions over the defined context.

3.1.3. Trigger-Action programming

Trigger-Action programming is based on the user defining a set of triggers that when are triggered reproduce some action. The complete set of triggers and actions that define one use case are usually called recipes. End users are able to perform simple trigger-action rules which allows them to modify the state of the system [10]. This simple split of the problem onto triggers and actions is suitable for environments such as smart homes. In smart homes, we have a small set of triggers (door alarm, clock, smart lock...) and a small set of actions (turn on lights, turn off the lights, start computer,..) which can be combined to create these recipes. Note that in our case we could have a large amount of trigger and actions depending on the scenario. Figure 4 shows simple examples of simple trigger-action recipes and their semantic meaning.

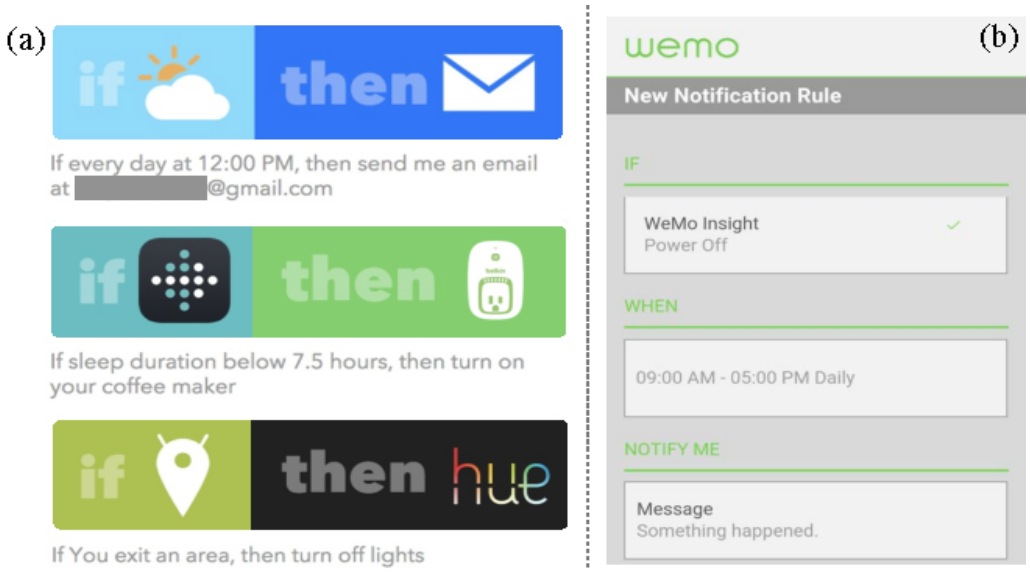

Figure 4 Trigger-action programming in existing products. (a) Example rules from IFTTT. (b) An example of programming a rule for a WeMo Insight Switch. Source: "Supporting mental model accuracy in trigger-action programming" [11]

Another example of simple trigger action case would be making one coffee when the person wakes up, for this example we would define clock which knows when an alarm has been triggered and which hour is it. When in the mornings the alarm goes on, we have to define a rule that when this action happens the machine has to make a cup of coffee, we will have a simple trigger action diagram (Figure 5).

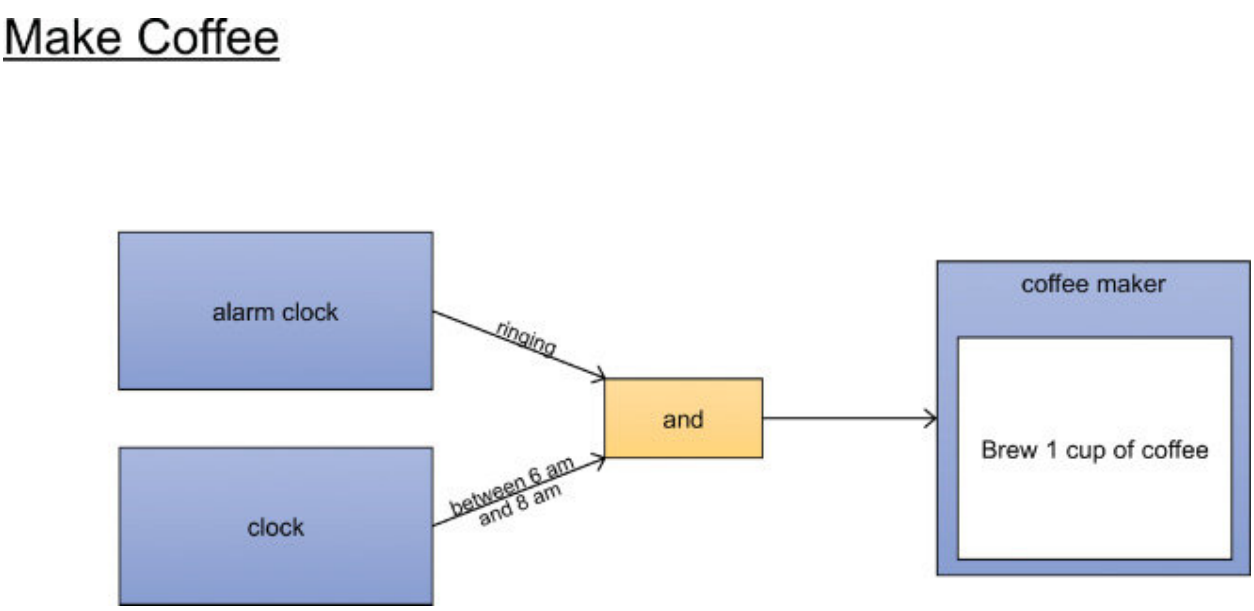

Figure 5 Coffee Maker Diagram Source: "Exploring End User Programming Needs in Home Automation" [12]

It has to take into account some contexts: what happens if the user delays the alarm? Should the coffee machine make one cup of coffee or should it wait until the user really stops it? Or even a worst scenario: what happens if the alarm rings twice on the same range hour, should the coffee maker do two coffees or only the first one? When we start adding this context aware to trigger action development we need the user to provide also context.

A more complex scenario is the amazon trigger system, where amazon provides to their cloud users the possibility of executing actions when something on one of the users cloud component is modified. This is a clear trigger-action programming case but where the triggers and the actions are specified by the amazon cloud services. Figure 6 shows a lambda trigger, a lambda on Amazon web service is a function programmed by the user and uploaded to the cloud where it is executed. The leftmost column of the figure shows the designer where are located all the triggers that can be used to define the recipe. On the next column we can see that the user has defined one trigger (Cloud Watch Event) that execute the lambda function (DDBBackup). On the right column there are the permissions that this lambda has access to. Despite amazon services are not addressed to end users with no programming languages they may take advantage of this simple trigger action system as they may not be experts on Amazon's services and therefore could not connect that easily their components.

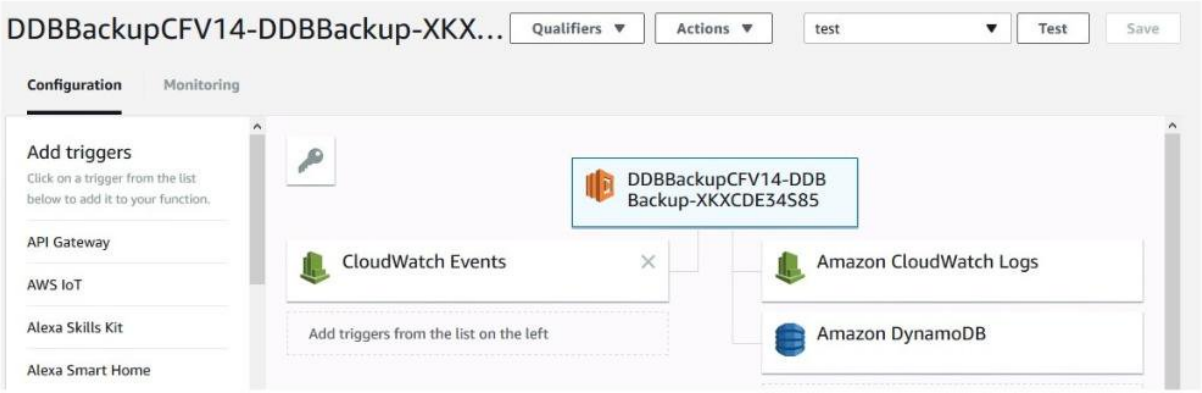

Figure 6 Amazon lambda trigger Source: Amazon Web Service

Finally we should choose a significant group of triggers and actions, as users will combine them by their needs [12]. Providing the right set of triggers and actions may not be straightforward and will require several iterations with domain experts in order to select the relevant set of triggers and actions. Also note that this set of triggers and actions can be modified by the developer of the system which will require more than just the EUD but will bring up new functionalities without breaking the users knowledge over the system.

Knowing which actions and triggers are more useful for the users can be easily visualized plotting them as a heat map where on the y axis we have triggers and on the x axis we have actions Figure 7. This heat map ideally will show in darker colors the combinations of action trigger that the users use the most. Nevertheless, a low occurrence of an action or a trigger may not indicate that we should remove it as maybe the low occurrence is caused by not having a matching action or trigger. For instance, let us define a set of triggers {house alarm triggered, clock, door unlocked, door locked} and a set of actions {turn on lights, turn off lights, make coffee} and we see on the use heat map that the action "house alarm triggered" is never used. This is actually never used not because it is not useful itself; it is never used as there is no action that makes sense to execute from the action set when the alarm rings. Adding the action "call the police" in this case will make more sense than removing the action.

Applying a trigger-action approach for training session is quite straightforward. First, we should define with the domain experts a relevant set of triggers and actions. Once the two sets are defined, the system should be able to reproduce all the user combination of these actions, and be flexible enough to introduce new ones. Note that if we will need branching all the conditions of the branch should be replicated as we cannot define "else" statements. Also for our approach we will need that actions modify the status of the system in some way that it creates a trigger to be able to concatenate different trigger actions. One potential problem of this solution will be addressing scalability as when approaching large scenarios we could reach unmanageable trigger-action chains.

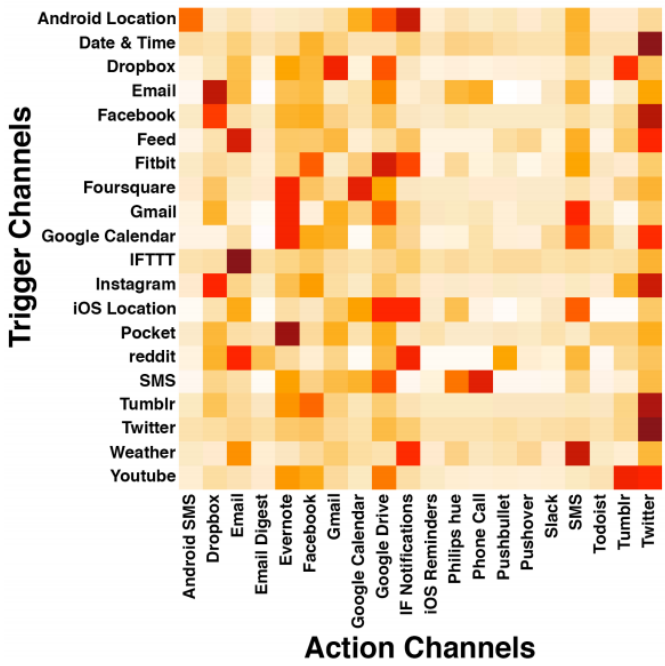

Figure 7 Triggers-Actions heat map Source: "Trigger-Action Programming in the Wild: An Analysis of 200, 000 IFTTT Recipes"

3.1.4. Visual Programming

Visual programming languages (VPL) can be defined as any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. Giving users the possibility of modifying programs visually and giving explicit feedback makes the enter barrier lower. This lowering of the entering barrier among the restrictions that the visual language introduces makes it suitable for EUD.

Examples of visual programming are Scratch [13], a platform of Massachusetts Institute of Technology, designed to teach the basics of programming to kids. Unreal Engine 4 [14] uses "Blueprints" to program video games; VisSim allows users to make complex mathematical models in smarter and faster ways while executing them in real-time; and CiMPLE is used for teaching automation through robotics.

Visual programming has been widely implemented among several domains as we have seen on with the examples mentioned above but one of the domains with more importance is the system/simulations ones which is closer to authoring training sessions Figure 8. Also education plays an important role on VPLS which may indicate to us that VPLS are suitable for users with low knowledge or no knowledge on programming but that is used on today's systems and interfaces.

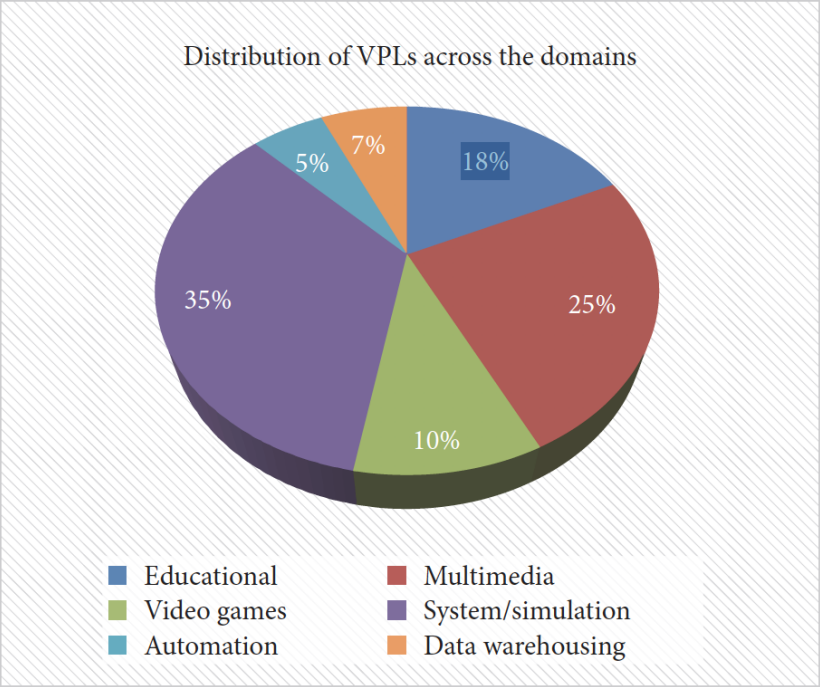

Figure 8 Distribution of VPLs Source:" A survey on visual programming languages in internet of things"

One example of domain experts using VPLS is for scientific visualization as a node based language [15], [16]. For example on Voreen domain experts can define a set of nodes which define how ray casts are done, materials, thresholds and more by combining nodes. Each node has one or more inputs and yields one or more outputs, both have a type that is color coded so users can connect the outputs to the inputs with the same color. Also the system prevents users from connecting things that are not compatible and executes in real time the user changes so they see which immediate effect the changes have over the final result. The figure shows an example of the voreen user interface where on the pop-up window is the visualization and on the main window a canvas with a set of nodes that compose this visualization. Note that on the left there are the set of nodes that are available; on the right there is a contextual menu to configure per-node parameters. (Figure 9)

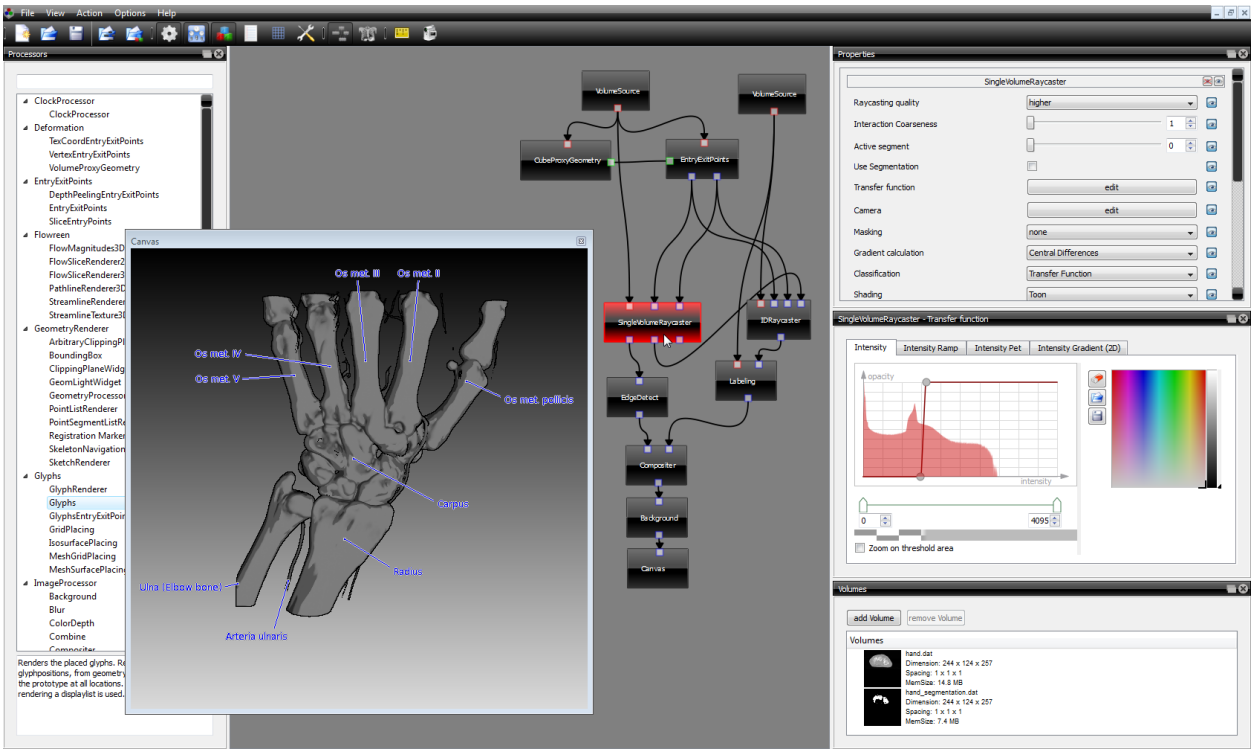

Figure 9 Voreen example Source: "Voreen: A rapid-prototyping environment for ray-casting-based volumevisualizations"

Also another popular example is Scratch, which directly translates programming concepts to a visual environment. Here the users have an entry point and blocks which represent programming concepts such as if-else, loops and variables among others. This is closer than the Voreen node approximation to programming as it gives more flexibility to the users but also introduces new concepts to them. As we can see in the hello world example Figure 10 all these concepts are color coded to give the end user a better understanding of what each block is on a top level semantic. For example control loops are coded with yellowish color whereas visual effects are coded with a purple one. In this case the user could start the action by clicking the flag on the top of the preview window.

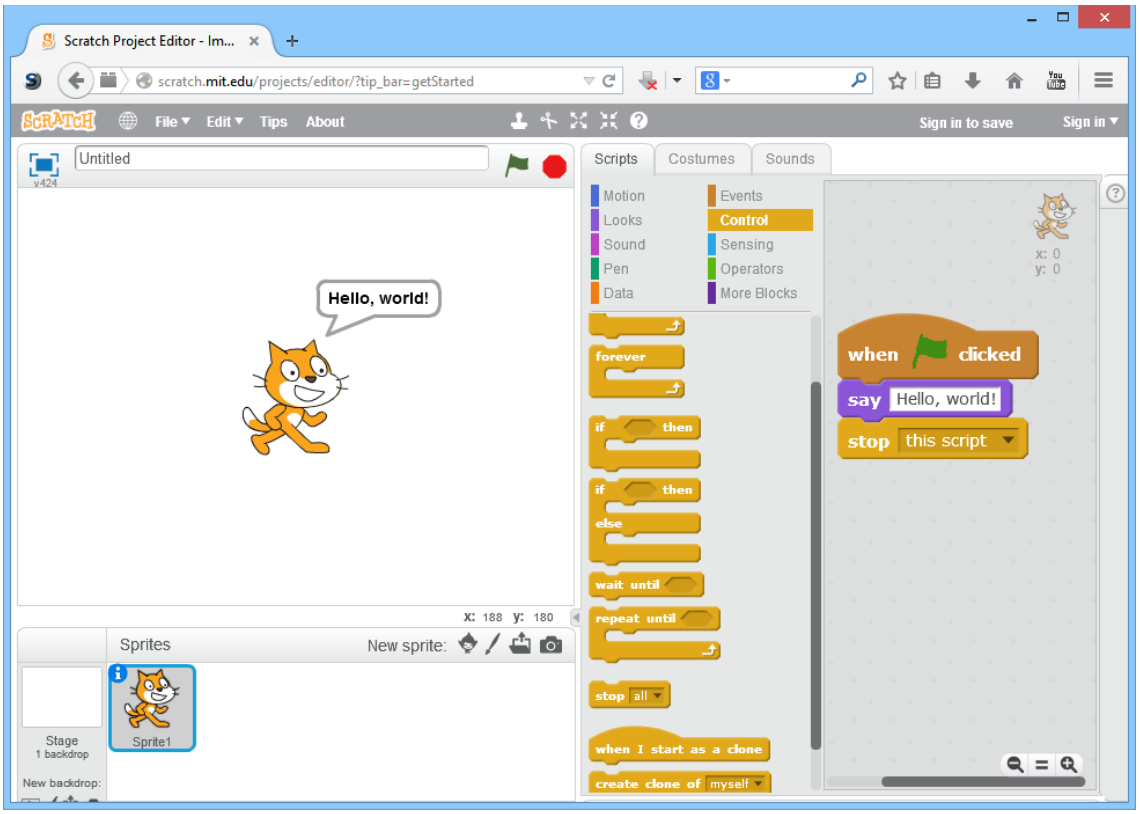

Figure 10 Scratch Hello World Source: "Scratch: Programming for All"

If we want to adapt this approach to authoring training sessions we may define a series of nodes as Voreen does with extra control blocks as in Scratch to accomplish a good scenario for training sessions. We have to take into account that this method also may suffer from scalability and reusability as we can end up with large defined graphs or non-reusable ones (which would result in repeating the same nodes several times).

3.2. VR for training

In this section we review the literature that refers to how virtual reality can be suitable for training sessions. Also provides some context on the reason why we decided to allow the tool to create sessions targeting VR Devices.

One alternative to computer simulated training sessions is video training (VT). On this field we know that Virtual Reality [17] training can supplement standard laparoscopic surgical training of apprenticeship and it is at least as effective as video training. Which suggests that the system that we are aiming to build can be at least as effective as the most common solution used nowadays.

Furthermore, we also know that if we divide the learning progress, we obtain the following results [18]:

- VR training group showed improvement of 54% on the VR posttest, as compared with 55% improvement by the VT group.
- The VR training group improved more on the VT posttest tasks (36%) than the VT training group improved on the VR posttest tasks (17%).
- Operative performance improved only in the VR training group.
- Psychomotor skills improve after training on both VR and VT, and skills may be transferable.

With these results we see that there are skills that actually can improve using VR over VT, although the authors state that further research on methodological quality and patient-relevant outcomes are needed.

Despite the fact that there are no differences in baseline assessments between VT and VR groups it is shown that as Virtual reality involves performing the real actions, the users will show improvements. On the specific case of laparoscopic cholecystectomy is shown that [19] [20] :

- Gallbladder dissection was 29% faster for VR-trained residents.
- Non-VR-trained residents were nine times more likely to transiently fail to make progress.
- Non-VR-trained were five times more likely to injure the gallbladder or burn non target tissue.
- Mean errors were six times less likely to occur in the VR-trained group.

All of this makes it worth it to try replacing standard video training by VR training sessions and explore how they can be improved to provide even better results.

Another alternative is to perform directly on the real scenarios under the supervision of an expert. Despite this one being more effective than VR by obvious reasons it cannot give us the security and the tolerance to failure that VR gives us. This extra security and tolerance to failure can give VR an advantage on some fields were failure can have a high cost.

Finally, we know that we can transfer successfully [21] skills from computer environments to the real world situations which makes it suitable for the task.

4. Requirement Analysis

As we aim to design a flexible system that allows end users to create new sessions by introducing the minimum economic and time impact on the process, we should set several requirements to the systems that allow us to approach the problem in the right way.

First, we should take into account, as its mentioned earlier, that target users will be domain experts that do not know about computer programming nor development but with the user knowledge to use modern interfaces and programs. Taking into account this information, we need to know which system fits better to this kind of users. Furthermore, the requirements and sessions are in constant evolution and the designed system may not fulfill all the possible combinations needed to design all the training sessions at the start. When there are new requirements to design new training sessions the system has to be able to handle them.

As virtual reality devices are constantly evolving and the access to modern ones are not always possible the system should be able to generate sessions suiting this restrictions.

In order to effectively answer such questions, we identify the following requirements:

R1: Designed for End Users

The final system should be used by end users that will not have any programming knowledge nor experience using similar tools. They should be able to use this new system, to author and reproduce training sessions with only a little training.

Although we have to introduce new concepts for the programming sessions, the system and interfaces should be as similar as possible to existing configuration screens or other applications widely used because users are used to change parameters on simple interfaces (volume, toggles, buttons), simple drag and drop (copy files, drop files on explorers) and simple navigation on 3D scenarios (video games).

R2: Extensible

As medical knowledge doubles every 6-8 years [22] with new medical procedures emerging every day, the system should be extensible by other developers as the requirements to design training sessions can change and the system should be updated to match the reality as close as possible. Furthermore, the extensibility of the system should not break previous sessions already built with it.

Finally the addition of new features to the system should be less costly in time and economically that designing new exercises from scratch as otherwise the system would not be usable.

R3: Self-Contained

We aim that the system doesn't need extra software to be installed, as this would hinder its usage. Also embedding it onto, for instance, a real time 3D engine editor will introduce an unnecessary overhead and we will need to make sure the compatibility of our system with previous and new versions of it. As long as the system is self-contained it will be easy for end users to install and use it without any technical assistance.

R4: Multiplatform VR

Since the tool aims to author training sessions that later on can be reproduced on Virtual Reality we should try to support as many devices as possible, including new and old ones.

Owing to the different interactions between devices and in order to specify more requirements, we will split them in two groups:

- **For 3 DOF devices:** We will use the controller to teleport the user around the scenario when it points and holds. (e.g. Daydream, Oculus Gear with a linked controller, Oculus GO)

- **For 6 DOF devices:** We will use native tracking of the device to position the user and we will add the possibility of teleport with the controller for devices with small movement area. (e.g. HTC Vive, Oculus Rift, Microsoft MXR)

On both platforms the interaction with session items will be performed with controllers.

We also decided to do not support platforms in VR which do not have a controller, the most known ones that do not satisfy this constraint are Google Cardboard and Oculus Gear without a controller.

5. Proposed Solution

So as to approach the problem of authoring nurse training sessions and later on reproduce this sessions we decided to split the problem into two parts:

- **Authoring training sessions.** For the first part of the problem we propose a visual programming paradigm based on nodes associated to a certain context (e.g. a surgical room prepared for laparoscopic appendectomy).

- **Reproducing training sessions.** For the second part of the problem we propose an automatic tool that reproduces in VR the designed session, allowing trainees to interact with the objects in the context to perform the required task (e.g. handling surgical instruments to the surgeon).

We decided to build our system using Unity as it gives us rendering features and a solid base to build on top of it. For the deployment of the system we compile the project for the desired platforms. Session management can be done with a simple menu that allow users to create, edit, reproduce or delete sessions. (**R1**)

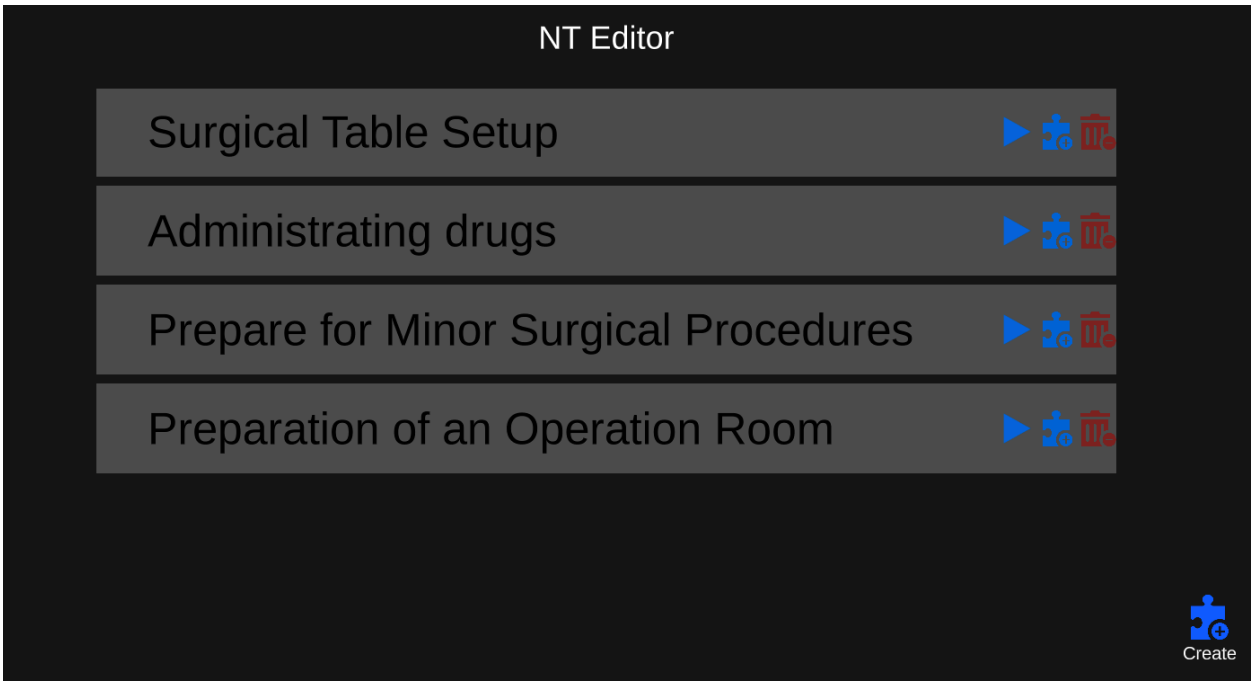

Figure 11 Session Selection Menu

Note that in the current state of the system the export of the session should be done manually by copying the designed session folder to the player folder. Furthermore mobile versions can only have the session player as the editor is not optimized for mobile devices. On desktop builds the tool automatically switches to VR mode when entering into play mode.

5.1. Authoring Sessions

In order to author training sessions we propose the following steps:

- Create a context by adding objects to scene to compose the environment where the session will be carried on. Objects are selected from a library that includes typical furniture and equipment in hospital rooms and surgery rooms.
- Modify object properties on the inspector.
- Create the behavior of the session.
- Create the behavior of the individual items.

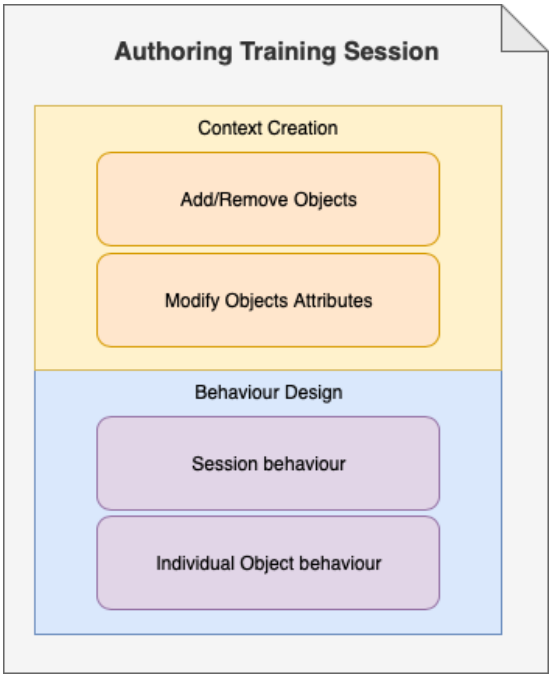

Figure 12 Steps required to design a training session.

Note that the items have the expected default behavior and give the user callbacks on their respective graphs for the modification of this default behavior (e.g. when administrating a drug to a patient it can have unexpected consequences). In addition this steps can be performed out of order with the only premise of having an object on the scene to build its graph and tween its properties.

The Scene Graph is where the session events and the generic behavior of the session can be designed, each training session always has one which provides special callbacks that let user control session flow.

5.1.1. Creating the session context

In order to create the context of the session, first we should set up the environment where the session is going to be developed. After that, we have the possibility to configure the attributes of the objects. For instance, indicating the remaining battery of an electronic device and if it is charging or not.

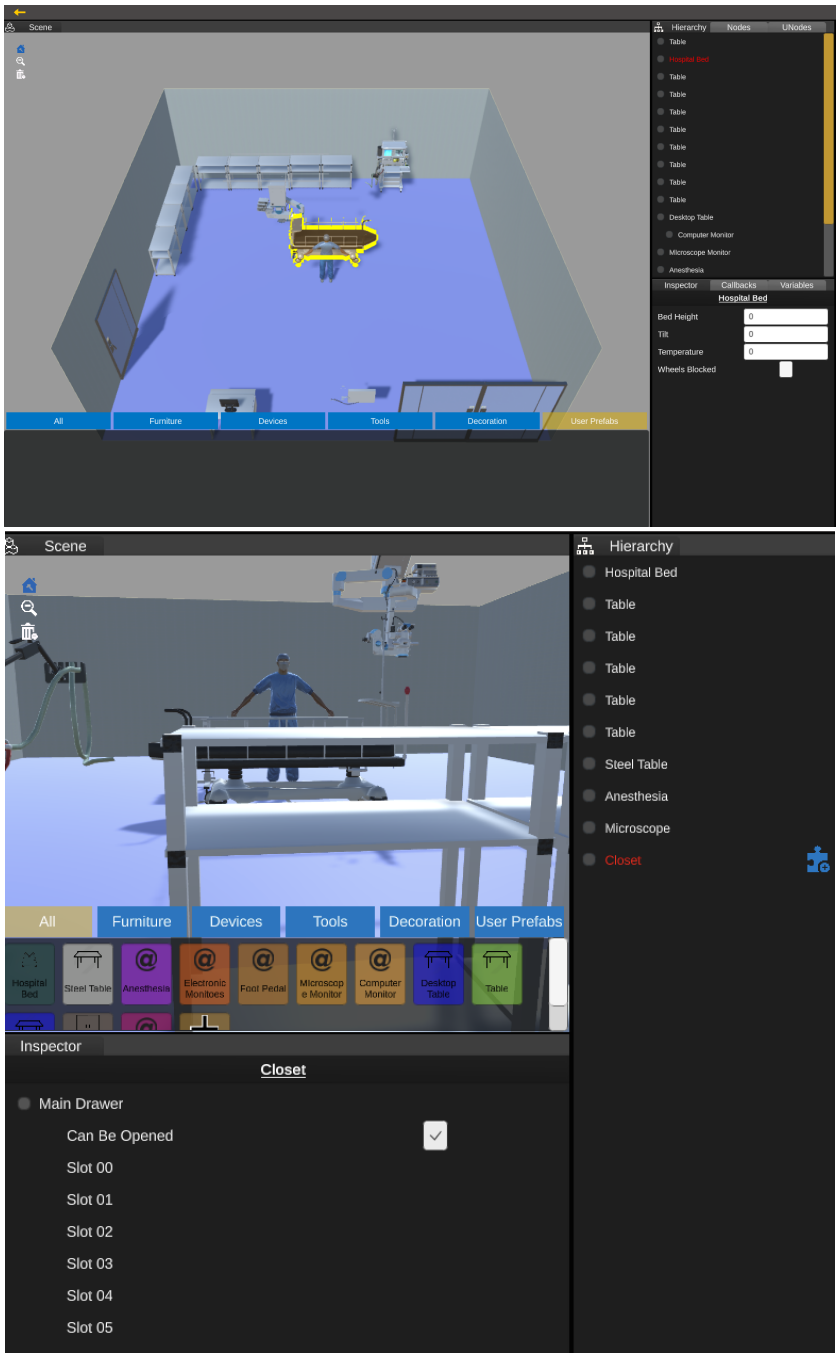

Figure 13 Editor tabs involved in context creation.

To add objects to the session we will use the scene tab where we have a visual representation of the context for the training session.

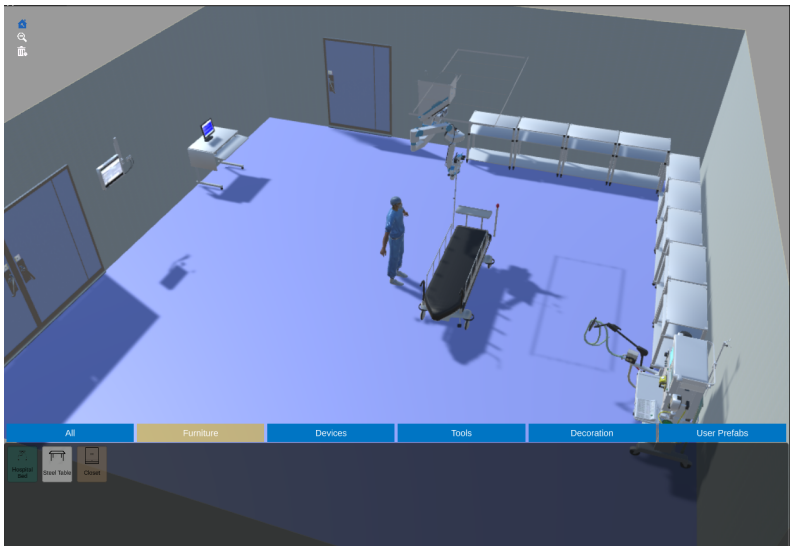

Figure 14 Scene 3D view tab

On this tab, we can switch between different interaction modes:

Figure 15 Interaction mode selector

- **Build:** Allows us to add an object to the scene.
- **Inspect:** Allows us to select an object to inspect the object attributes.
- **Delete:** Allows us to delete an object from the scene.

So as to add an object, first we have to select **build mode** and choose an object from the bottom menu:

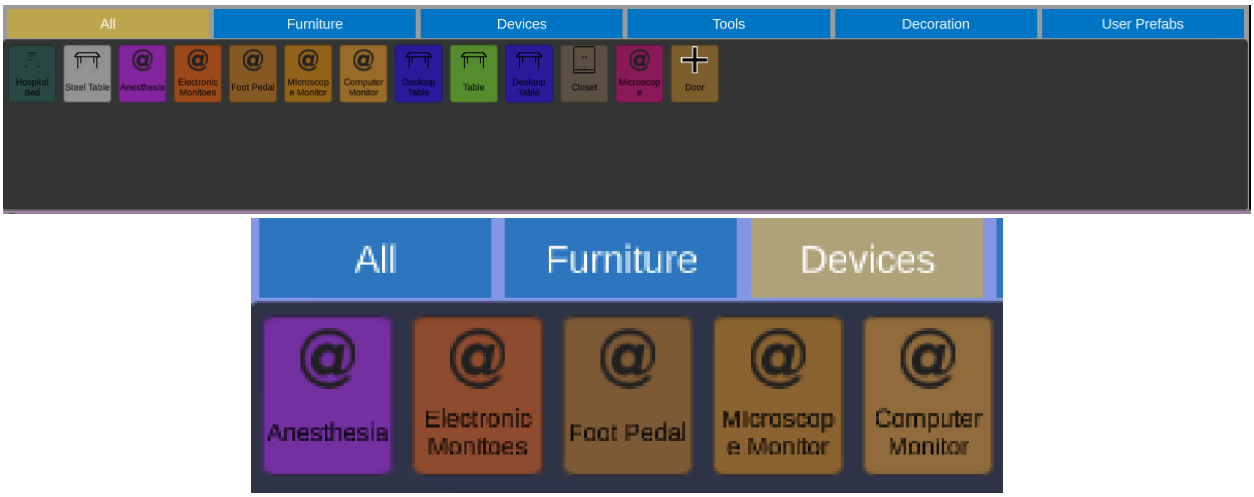

Figure 16 Items menu. On top tabs with items separated by categories. On the bottom a zoom of Device objects.

Once we select an object it will follow the mouse position on the scenario and it can be placed by right clicking. If the object position is invalid it will show a red outline over the object that we are trying to place and it will not be placed until is moved to a valid position. Moreover, if we put an object into another one, the container will acquire a **yellowish colour.**

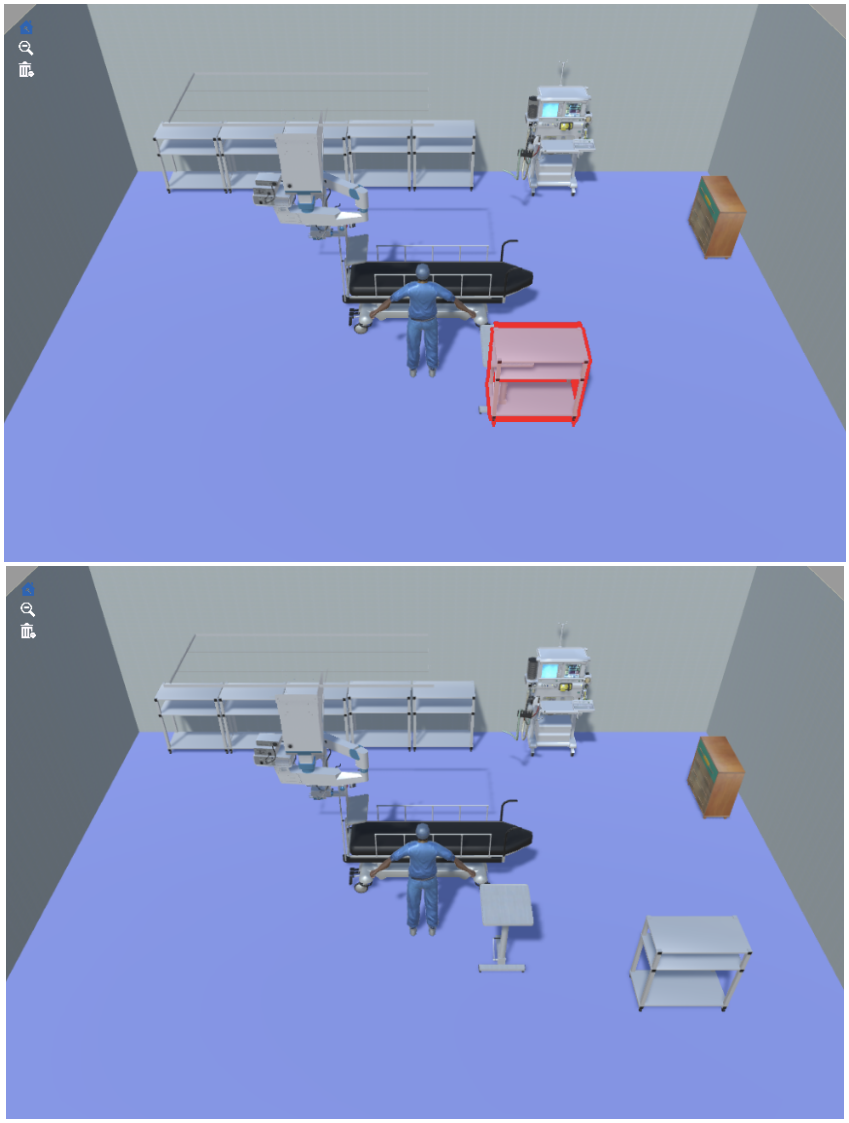

Figure 17 Placing a steel table on the session scene. On the top incorrect placement position. On the bottom correct placement of the steel table.

For deleting objects we can select the **delete mode** and it will **highlight in red** the object that is under the cursor and will be removed on right clicking. Also it can be deleted from the hierarchy context menu for a more accurate delete for objects that are handled inside others.

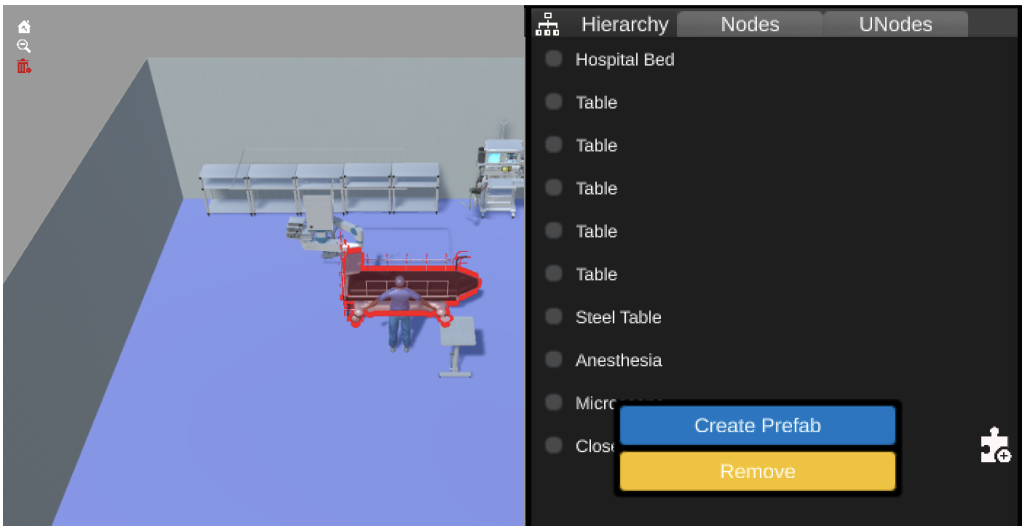

Figure 18 Ways to remove an object added to the session. On the left removing with the interaction option on the scene. On the right removing from hierarchy tab.

The scene view can be modified as follows to achieve a more accurate placing of the objects.

- **Up/Down arrow:** Scene vertical rotation.
- **Left/Right arrow**: Scene horizontal rotation.
- **Scroll button**: Zoom in/out.
- **Q/E when placing object**: Rotate object to the left/right.

After adding an object to the scene it will appear on the hierarchy tab. This tab contains the same information as the 3D scene but ordered in a hierarchical way. It helps us to modify difficult accessible objects such as objects contained into other objects.

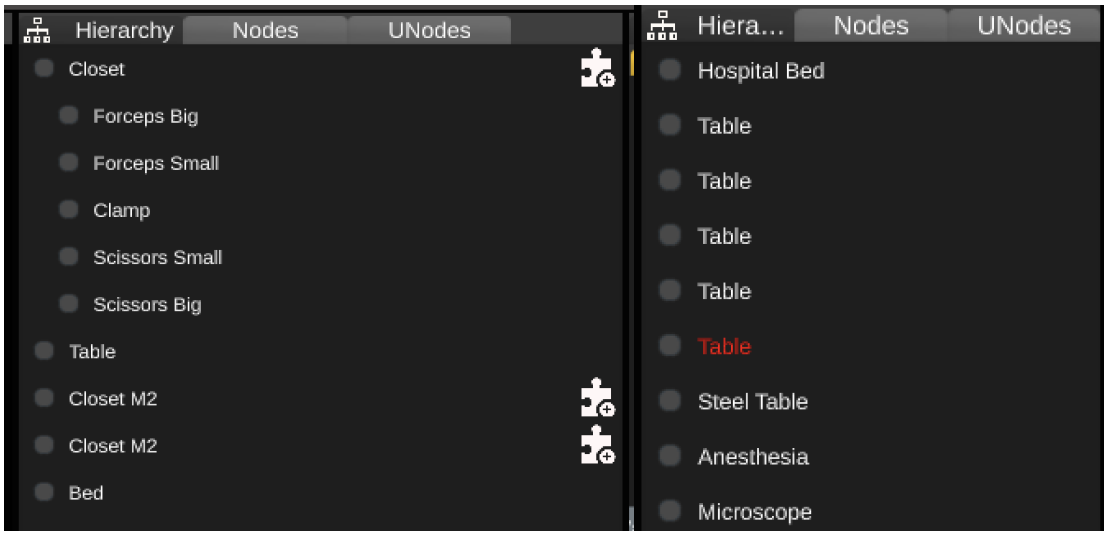

Figure 19 Hierarchy tab preview of two different sessions. On the right Table is on red as it is selected.

Moreover, when double clicking any of the object names it will highlight that object on the scene and select it as the inspecting object. An inspecting object is the object from which we are seeing the configured attributes on the inspector window.

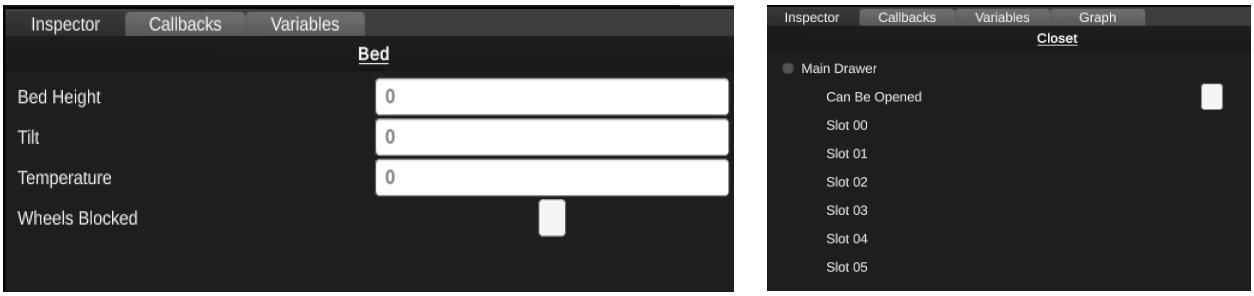

Figure 20 Inspector tab of two different objects. On the left inspector of Bed attributes, on the right attributes of a closet with 6 slots.

These attributes can be numbers, words or sentences, conditions (yes or no) or object references. We will be able to edit the first three and give them default values. The object references are filled mostly by adding other objects as children of this one or by performing certain actions during the reproduction of the session.

When we are on **inspect** mode we will see a highlight over the object we are trying to select and the selected one:

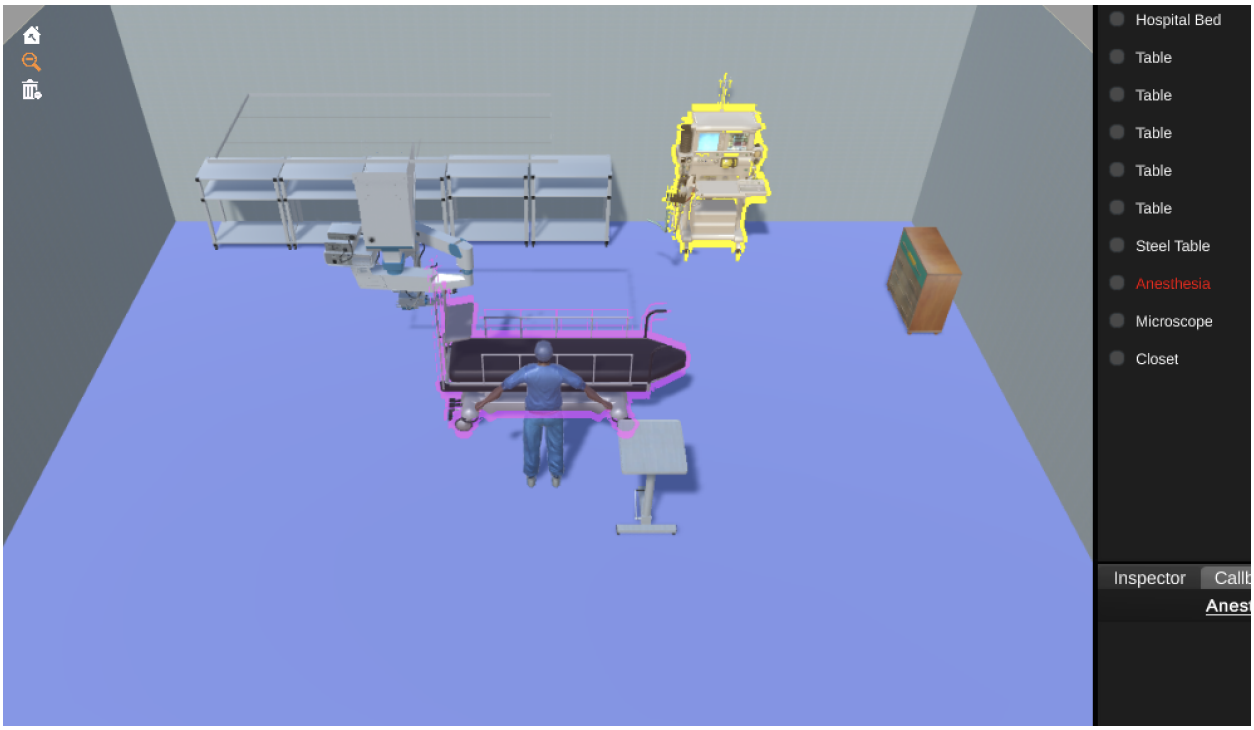

Figure 21 Object selection.

From the hierarchy window we can create a prefab. Creating a prefab will make it appear on the bottom menu of the Scene view for further use. When we create a prefab we save not only the selected object but also their children objects, their graphs and all the attributes.

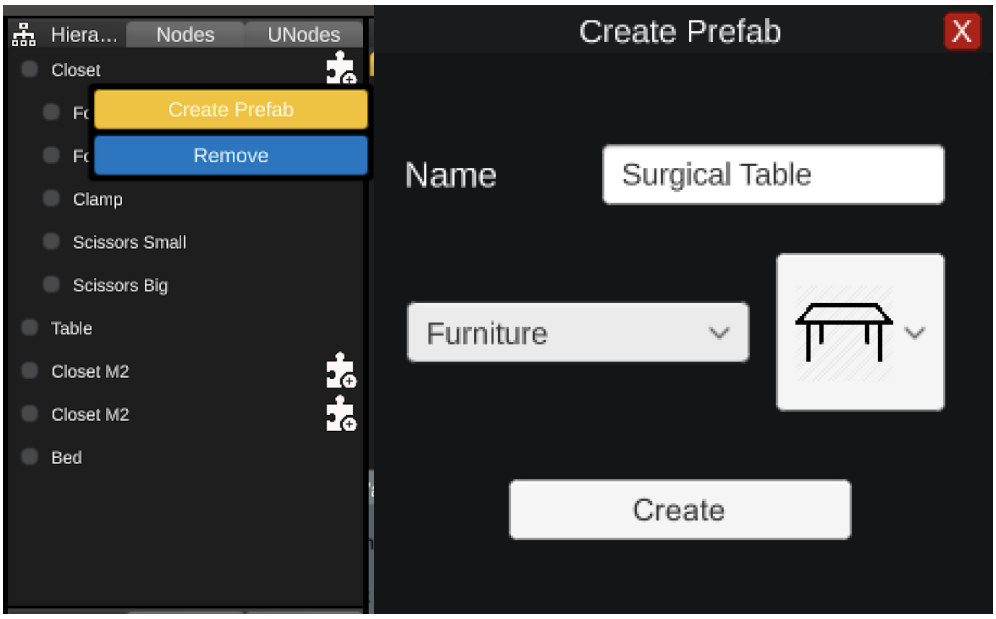

Figure 22 Prefab creation process. On the left context menu to start the prefab creation. On the right pop-up window that allow prefab configuration.

When creating a prefab, we will be asked to give to it a friendly name, icon and a type for the object in order to be displayed on the bottom menu of the scene window.

5.1.2. Creating the session behavior

In order to create the session behavior we should define a series of graphs, one scene graph and many object graphs are contained at least in each session. The scene graph embraces the definition of the procedures for the session while the objects graphs enclose specific behavior for the objects.

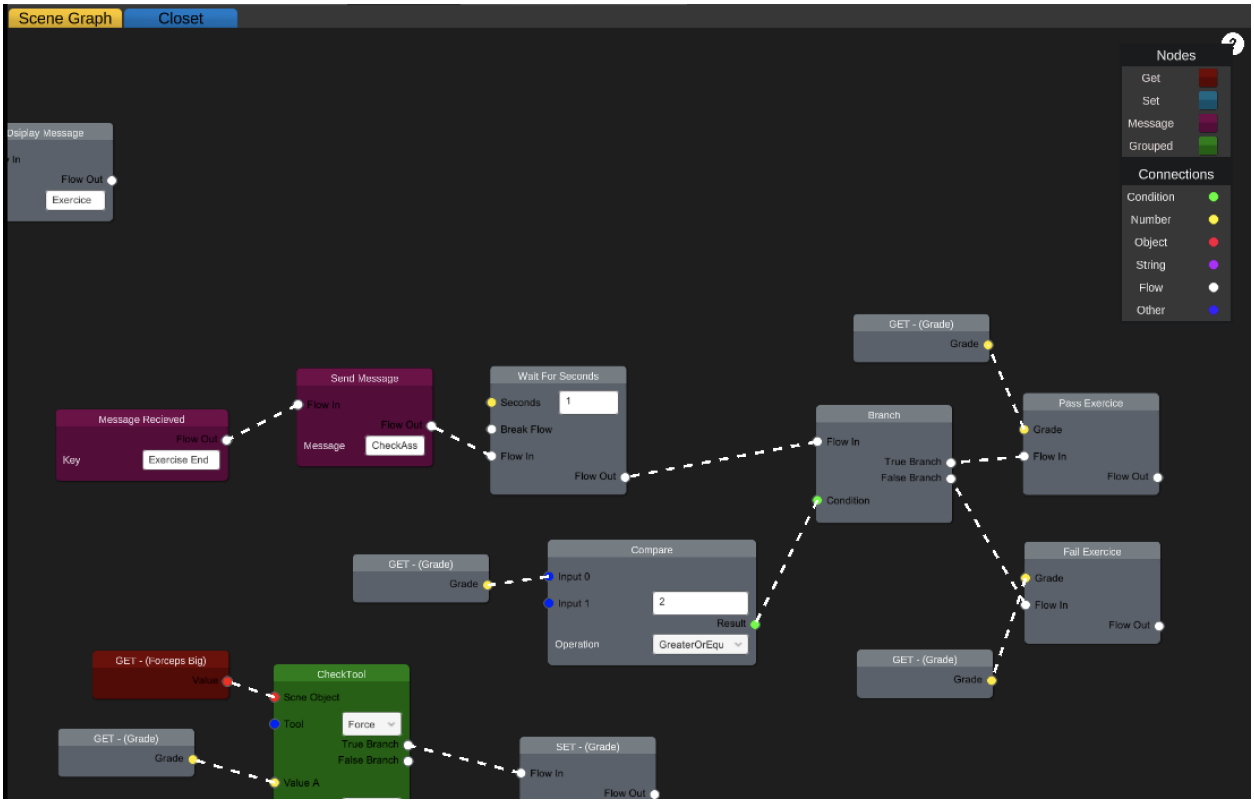

Figure 23 Scene Graph example. See 6.1.3. Composing the session for more details.

1. On **the graph tab** we can find these different graphs separated into different tabs. By means of the hierarchy window, we will be able to open new graphs for objects.

2. On **the hierarchy tab** the objects with any callbacks have an extra button that let us open the graph for the selected object.

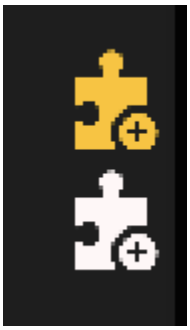

Figure 24 Object graph creation buttons.

If the object already has any graph it will open the assigned graph otherwise it will generate a new one. Graphs by default are empty if we are not building the object from a prefab, in this case the graph will contain the same as the original prefab has.

The following picture shows the node anatomy:

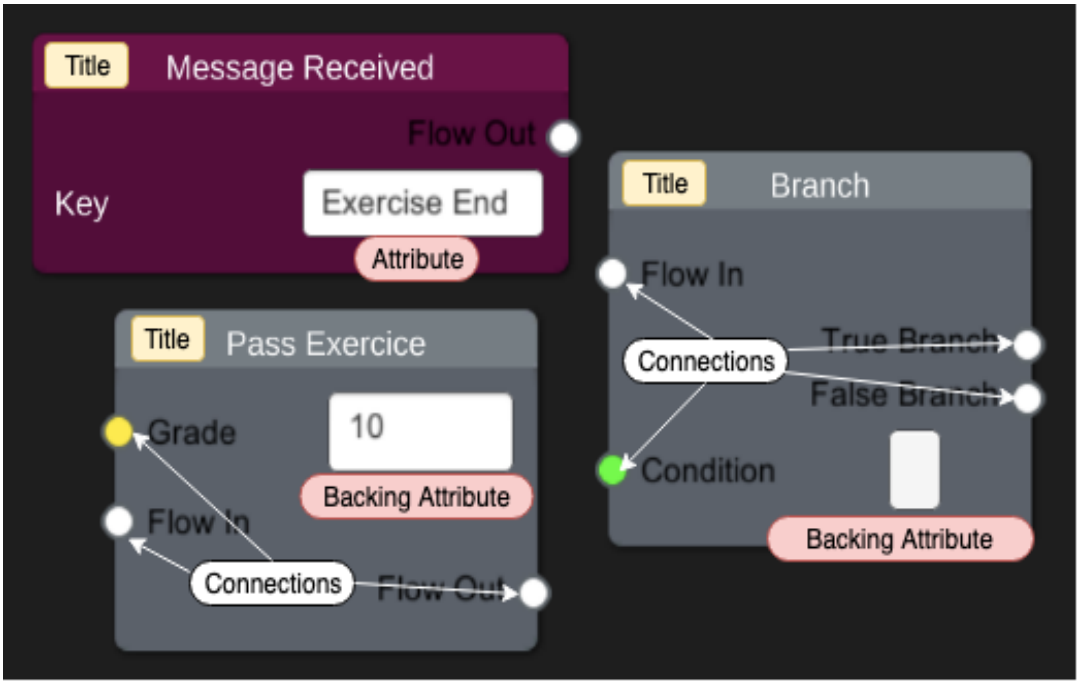

Figure 25 Node anatomy.

Where **title** is the identificative name of the node, connections are the points where we can concatenate different nodes. The **connections** on the left are **input connections** which can receive **output connections** (right ones). White connections define execution flow and other colors mainly are defined for getting or setting values.

When an input connection is not connected and is not an execution flow connection, we have the possibility of a **backing attribute** that will receive the indicated value. On normal **attributes** we cannot get the value by connecting it to other node, but we can set the value on the available UI field.

When it comes to execute a set of nodes, we have to keep in mind that:

- When starting the execution of a set of nodes it will follow a path that we will call the execution flow.
- The execution will always be started with a callback. (Message Received node)
- There can be several execution flows started at once.
- The execution ends when it reaches a node that is not connected with another node.
- When sending a message, we may start new execution flows.

In the following figure, we can see how when we receive the message **"Exercice End"**, we execute the message received node which redirect us to the branch node. On the branch node we evaluate the condition and redirect the flow to fail exercise node, which finally has no node connected to follow the flow and therefore the execution ends.

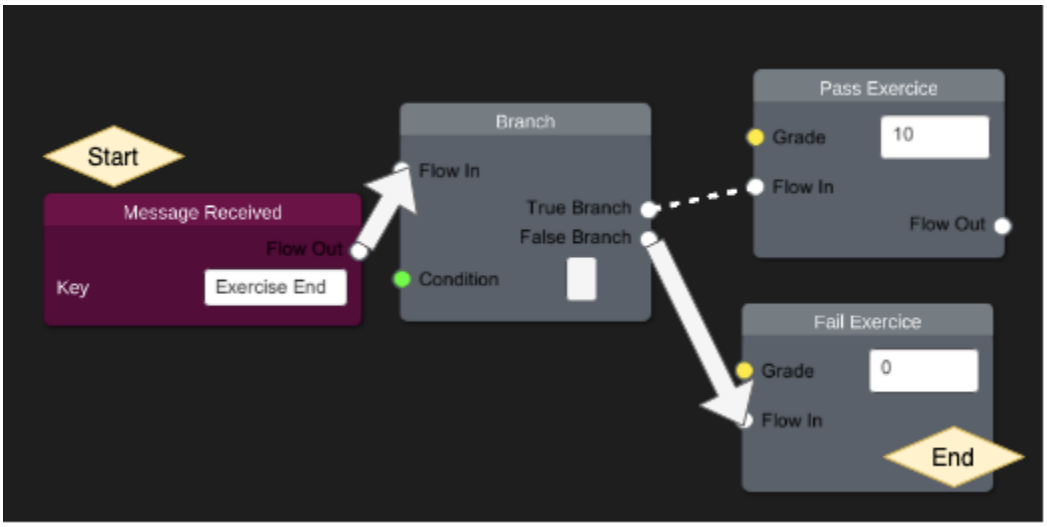

Figure 26 Graph execution example.

Figure 27 Nodes and connections colors meaning.

In order to make nodes easily identified on the graph window we can find a small help that explains the color of the nodes and the connections to make them understandable by an **end user** (**R1**), it can be collapsed/showed by moving the mouse over the help button.

There are colors that are similar but mean different things on connections and nodes, that is why nodes are displayed as colored boxes and connections are displayed as colored dots. For instance, the **grouped nodes** and **condition connections** share the green color but they are completely different.

For composing the graph behavior we provide the following set of nodes:

1. **Messages:**
   1.1. Message Received (Callbacks): Node that is executed when the message on the key field is received. When executed it continues the execution with the Flow Out node that is connected with.

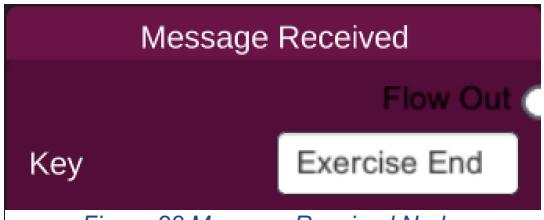

Figure 28 Message Received Node.

1.2. Send Message: Sends the message on the Message attribute when its executed, after that the execution continues with the node connected to the flow out port.

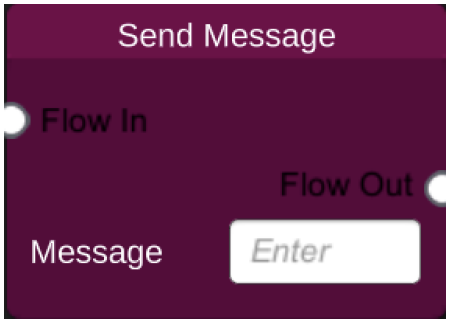

Figure 29 Send Message Node.

2. Flow

2.1. Wait For Seconds: Waits for the amount of seconds before continuing the execution with the node connected to out flow. If an execution flow comes in, on the break flow connection, it will stop the wait for seconds and continue the execution immediately.

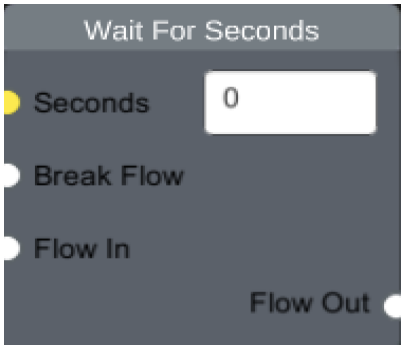

Figure 30 Wait for Seconds Node.

3. Control

3.1. Branch: When executing evaluates the connection condition. if it is true then the true branch connected node will be executed, otherwise the node connected to the false branch will be executed.

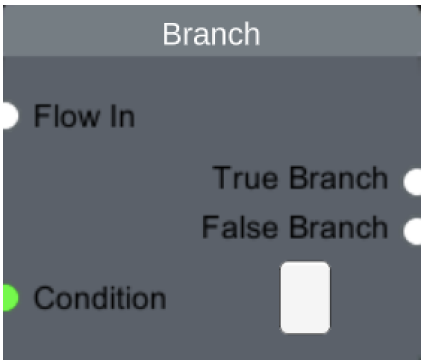

Figure 31 Branch Node.

3.2. Compare: Compares the two inputs on the ports input 0 and input 1 with the operation selected, in the Result output port we can see the result of the comparison.

Figure 32 Compare Node.

3.3. Boolean Operation: Applies the AND or OR operation to the two input conditions and outputs the operation result on the result port.

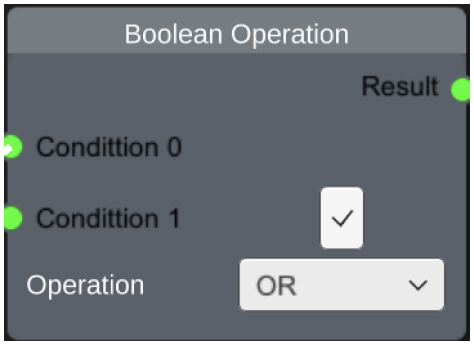

Figure 33 Boolean Operation Node.

4. Exercise Control

4.1. Fail Session: Ends the session with the indicated grade (the grade is not used on the current system but will be used to generate reports of the session)

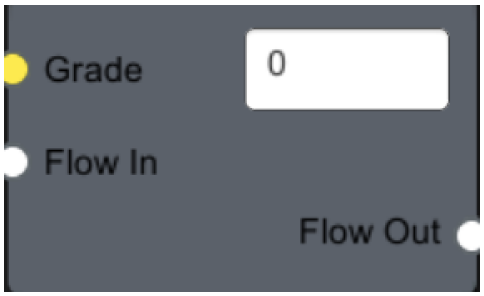

Figure 34 Fail Session Node.

4.2. Pass Session: Ends successfully the session with the indicated grade (the grade is not used on the current system but will be used to generate reports of the session)

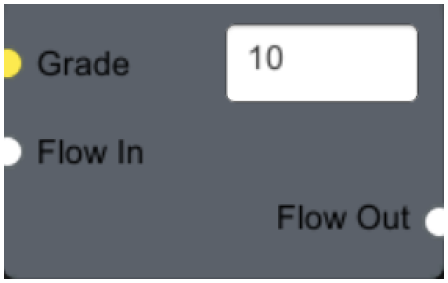

Figure 35 Pass Session Node.

5. Operations

5.1. Sum: Sums Value A and B.

5.2. Mult: Multiplies Value A and B.

5.3. Div: Divides Value A to B.

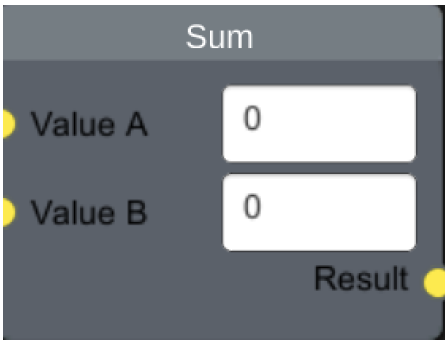

Figure 36 Sum Node.

6. Variables Nodes

6.1. Get variable: Returns the value of the selected object, variable or attribute.

Figure 37 Get Node.

6.2. Set variable: When executed sets the value of the variable or attribute. This value is stored during execution and resets every time that the execution restarts to the default value.

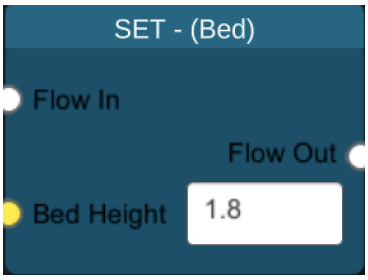

Figure 38 Set Node.

These nodes can be added from the nodes window or the context menu except the variable nodes which can be added from user variables, inspector variables of an object or the hierarchy window.

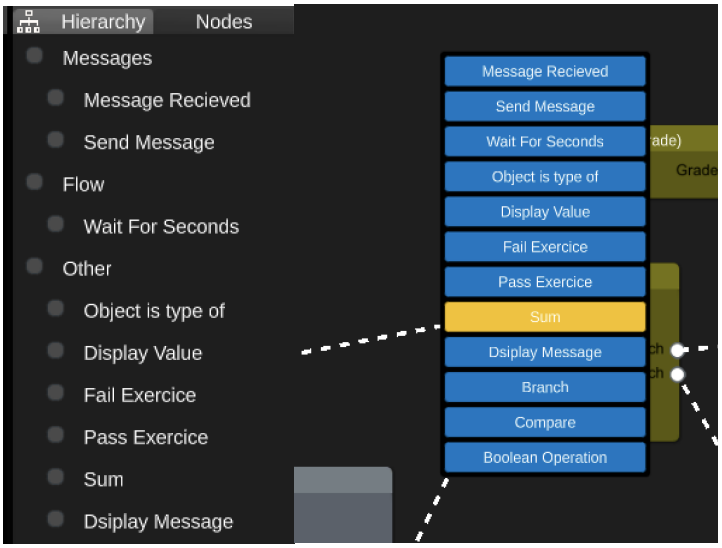

Figure 39 Nodes Menu.

Users can define global variables that can be words or sentences (**Magenta S**), numbers (**Yellow N**) or conditions (yes/no values - **Green B**) which can be used as nodes too.

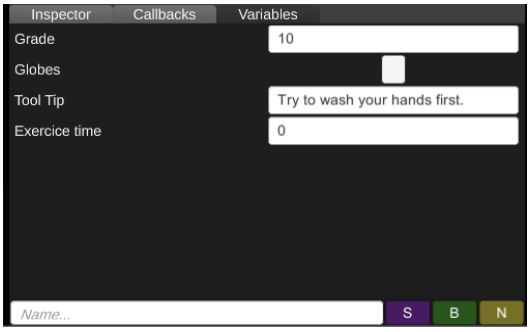

Figure 40 User variables.

The created user variables can hold default values and they can be modified later on with the different graphs. In order to add predefined messages nodes there is the callback window, which

shows the callbacks available for the selected graph. On the right, there is a play button that allows us to reproduce the application flow when this message is triggered.

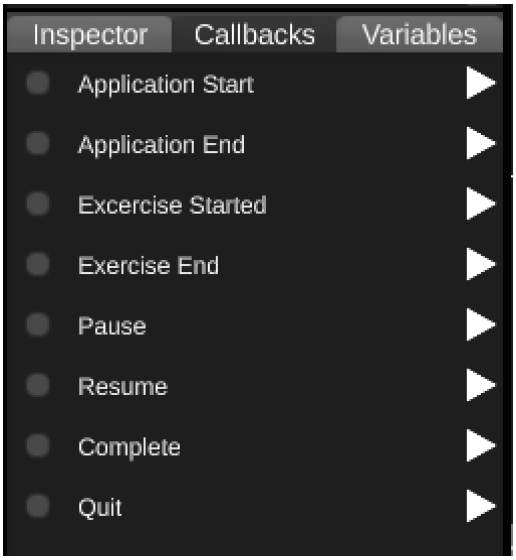

Figure 41 Scene Graph Callbacks.

When there are several nodes on the graph, we can select a group of nodes by dragging with the mouse:

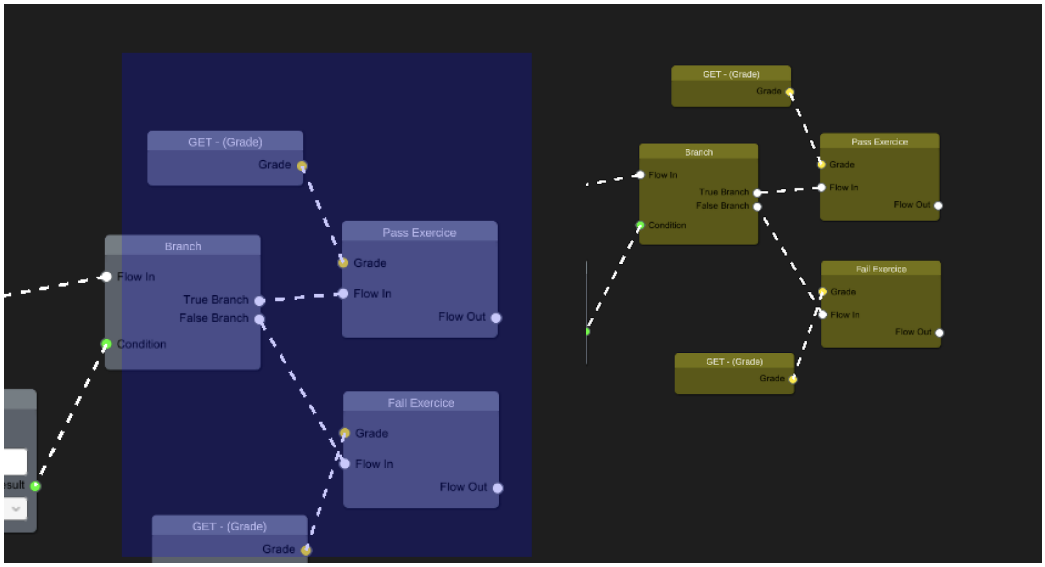

Figure 42 Select several nodes.

Also, if we have a node or a selected group of nodes we can open a context menu for grouping the nodes or deleting them:

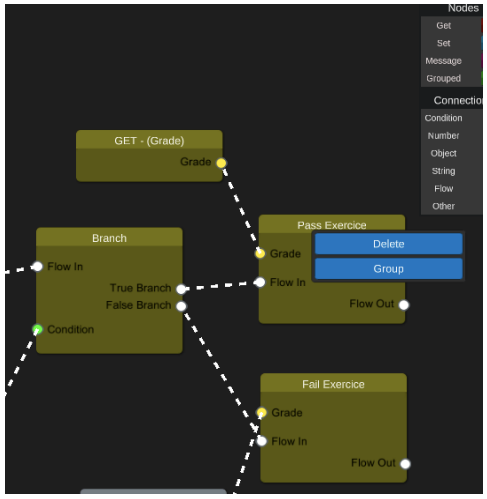

Figure 43 Context menu over a node or a group of nodes.

When grouping nodes, a window will be displayed to the user to give it a friendly name, after that the selected nodes will be replaced by the grouped one as we can see on the following pictures:

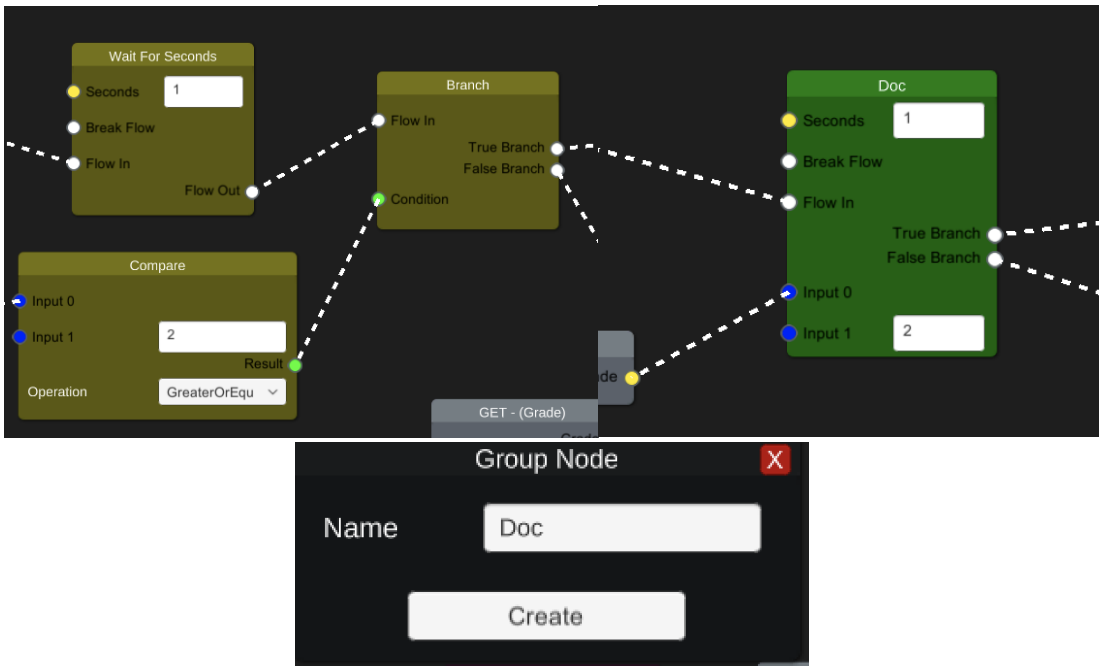

Figure 44 Process of creating a grouped node. On the top left the selection of the nodes that are selected for group. On the top right the grouped node. On the bottom the pop-up window that appears when creating a group node.

To add nodes to the graph we can drag from other tabs to obtain the following nodes:

- **Node and UNode tabs:** Create the dragged node or grouped (user defined) node.
- **User Variables and Inspector:**
  - Get: Create a getter node for the property.
  - Set: Create a setter node for the property.
- **Hierarchy:** Create a getter node for the scene object (Always return an object reference).
- **Callbacks:** Create a receive message node with the dragged callback

Figure 45 Drag from other tabs to graph tab example.

5.1.3. Extending the system

As we exposed on R2, we want our system to be as extensible and scalable as possible, to achieve this the previously showed interfaces are populated dynamically.

The objects shown in the bottom menu can be extended or modified by updating the system (**R2**) as they are stored on a scriptable object as follows:

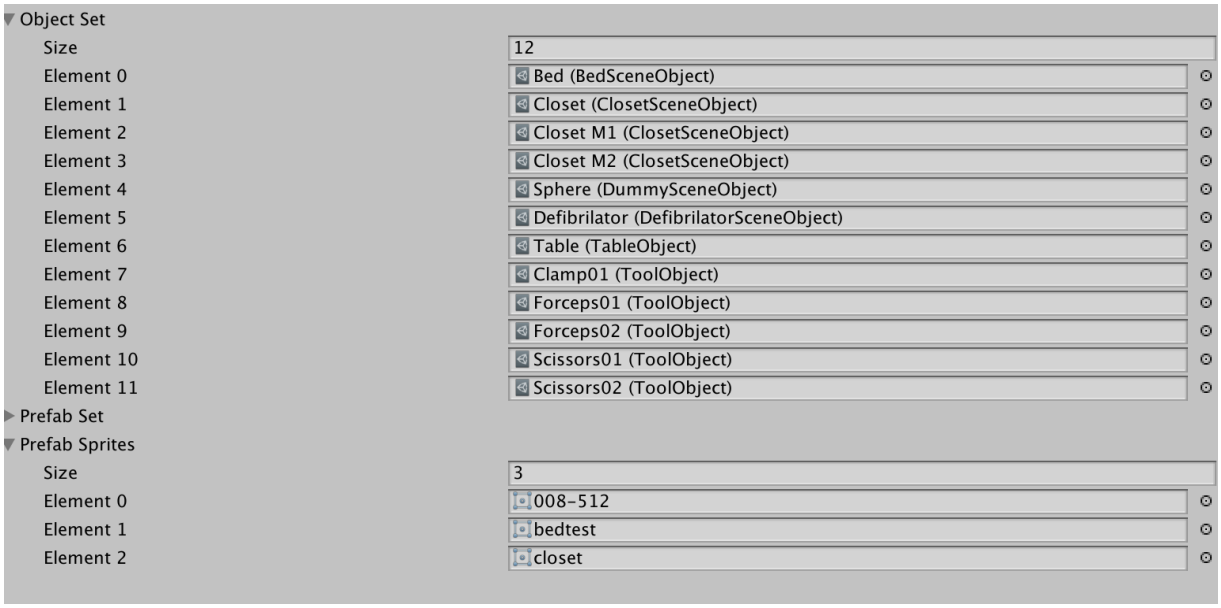

Figure 46 Defined object set.

By adding a new item to the object set, it automatically can be added to a session but if we modify an existing one it will replace the old one by the new when updating the editor/player on the platform. If a new object is added to the system and we try to reproduce the created session on an older player version it will not reflect the changes.

When adding an object we can also provide a default behavior (**R2)** for this by extending a base class as follows:

```
public class ClosetSceneGameObject : SceneGameObject
```

And adding it to the associated prefab to the object:

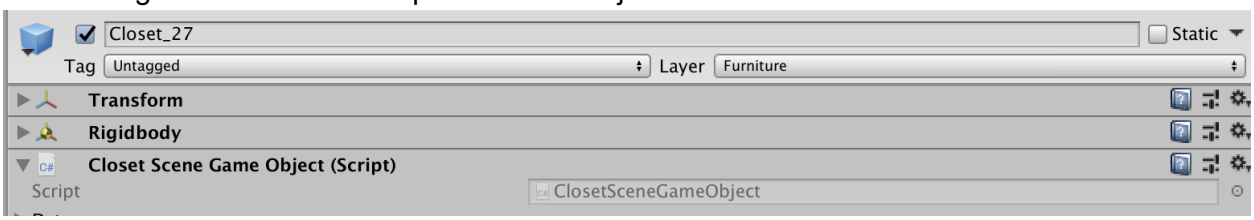

Figure 47 Definition of the object prefab.

For the inspector attributes are directly derived from C# code of the associated scene object in the object set, which is defined as follows:

```
public struct BedData{
    public float bedHeight;
    public float tilt;
    public float temperature;
    public bool wheelsBlocked;
}


[CreateAssetMenu(fileName = "BedSceneObject", menuName = "NT/Scene/Bed")]
public class BedSceneObject : SceneObject<BedData> {

}
```

This allows us to create objects with different data contents and extend or modify them as needed. Note that when we create the item on the object set, it should be of the defined SceneObject class (we will extend more on object and data extensions on the results section).

The nodes are also extensible (**R2**) by developers creating a new node that extends the base NTNode or any of the available variants:

```
public class BranchNode : NTNode
```

When creating a new node you can define the inputs:

```
[NTInput]  public DummyConnection flowIn;
```

Output and their return values:

```
[NTOutput] public DummyConnection trueBranch;
  override object GetValue(NodePort port){}
```

Next node that should be executed:

```
public override NodeExecutionContext NextNode(NodeExecutionContext context){}
```

Node execution:

```
public override IEnumerator ExecuteNode(NodeExecutionContext context){}
```

Visual and listing options:

```
public override string GetDisplayName(){}
public override List<string> GetPath(){}
public override int GetWidth(){}
```

5.2. Training sessions

Once the training session is designed you can reproduce it on VR. In this view there is no extra information about the visual programming nor the callbacks. When the session is started the respective messages are sent to the graphs and therefore the execution starts.

As we can see this part is completely standalone (**R3**) and can be deployed on many VR devices (**R4**). Finally, when modifying any object it has the ability to send a message to start a new flow or to modify the data of the object that is being modified.

For instance, an example of reactive object would be:

```
private void OnTriggerEnter(Collider other) {
        if(other.tag == "Tool"){
                HoldItem(other.GetComponent<SceneGameObject>());
                MessageSystem.SendMessage("Item Added " + data.id);
        }
}
```

In this object we listen to the trigger enter callback and check that the other object tag is “Tool” and we send the respective message to allow the execution of the graph associated with it. Note that all objects need specific development for the VR behavior and there are specific functions that need to be called to affect the scenario as desired.

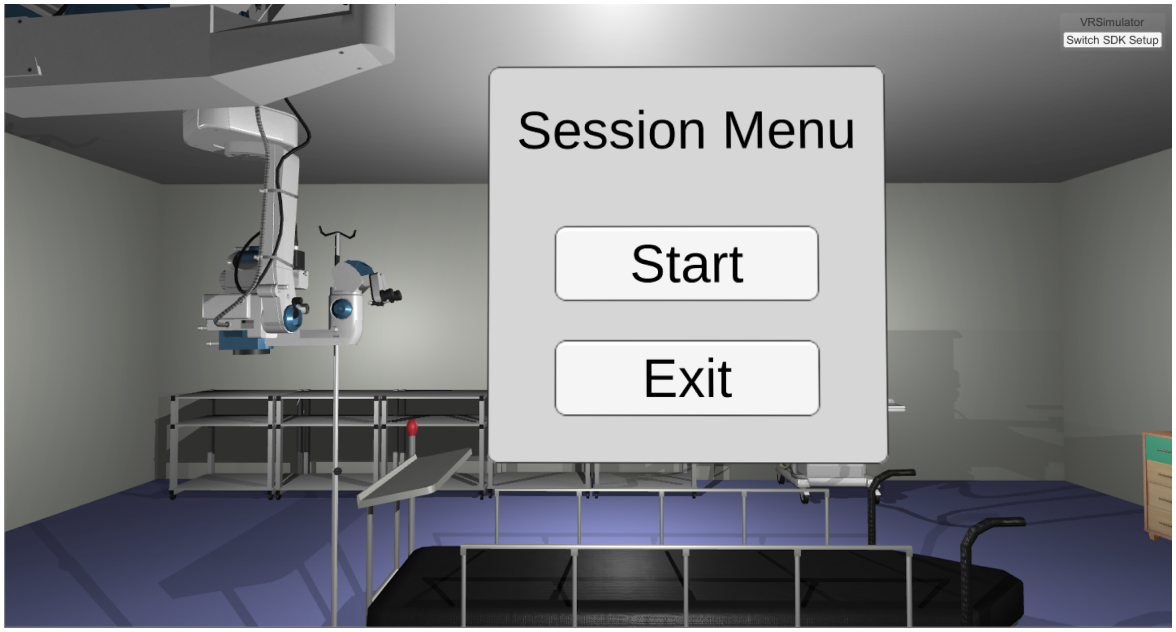

Figure 48 Start session menu.

Figure 49 Inside of a session view.

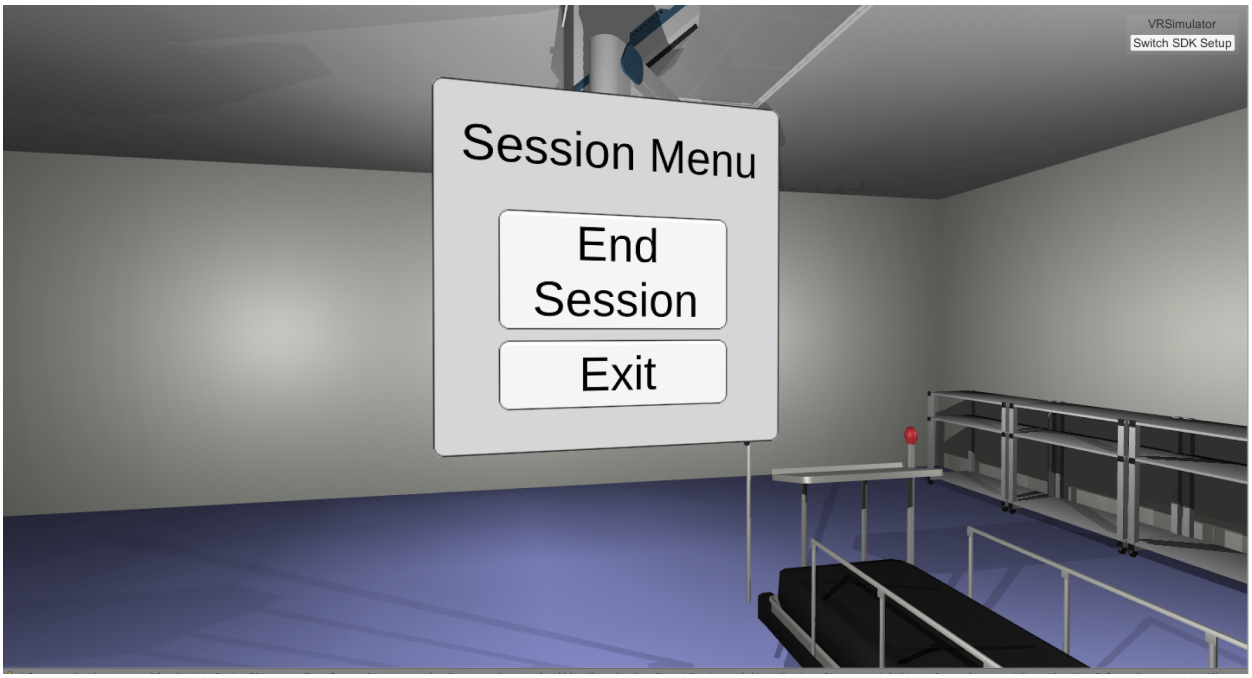

Figure 50 Session Pause Menu.

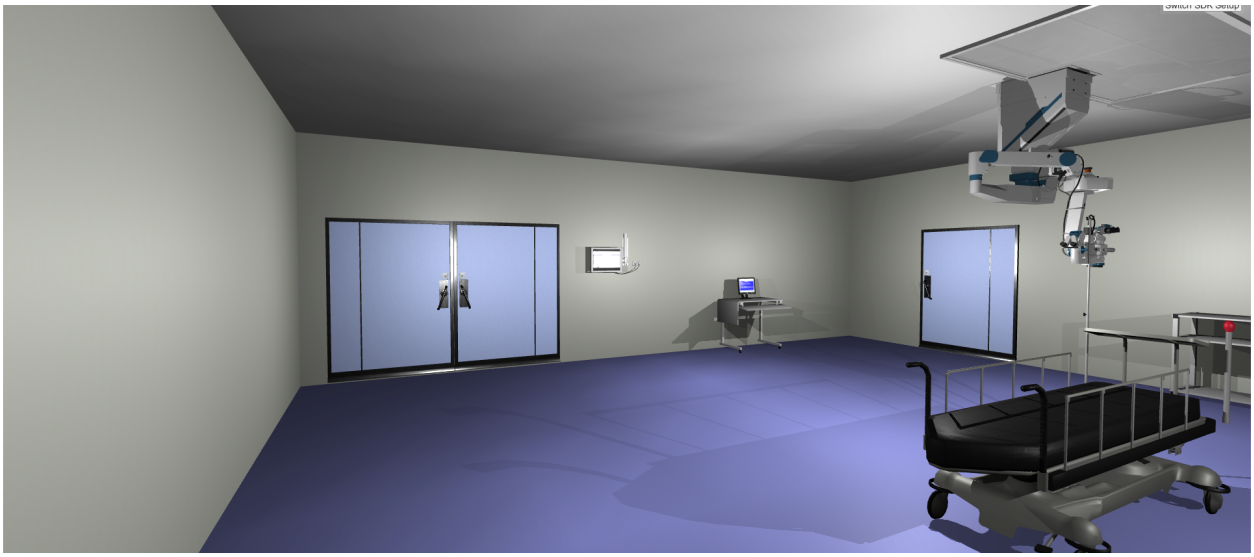

Figure 51 VR view of a room created with the session editor.

Figure 52 End of session feedback.

6. Results

We will author a training session which will consist on the preparation of a surgical table for a traumatological operation. We will do it in a simplified version as this will allow us to explain and manipulate better the example session.

For this we will define two users:

- **Trainer:** Will be a professor who wants to see if a student is able to reproduce the steps shown in class for the preparation of the surgical table.
- **Trainee:** Will be the student who has to prepare the surgical table with the steps that the professor has previously explained in class.

Now we will split the training session into three main steps:

- The trainee washes his/her hands.
- The trainee puts on the gloves and the table cover.
- The trainee places the surgical instruments in the correct order. In this step there will be only 5 surgical instruments.

Note that there will be no information displayed to the trainee on the session as she is supposed to know the procedures beforehand. For the demonstration we will first show how we propose the authoring of this training session to be and how this session is performed later on by the trainee.

6.1. Authoring training session

For authoring the training session we will assume that all the needed objects are already available on the system except the scissors instrument. And for the nodes we will assume that only the nodes previously listed are available on the system and we will describe the definition of the new needed ones.

When new objects and nodes are added to the system we will also need to generate a new version of the tool that supports the added ones. Old sessions generated with previous versions will remain the same (if some node or object in use for the session is changed on a new version of the system, the available one on the current version of the system where we are executing it will be used).

6.1.1. Define Objects

Firstly, we need to create the scissors instrument and introduce it on the object set. For creating the scissors object we will use the available `SurgicalInstrument` object which has no callbacks and no user configurable data. We will create it on the available context menu under NT/Scene/SurgicalInstrument and configure it as follows:

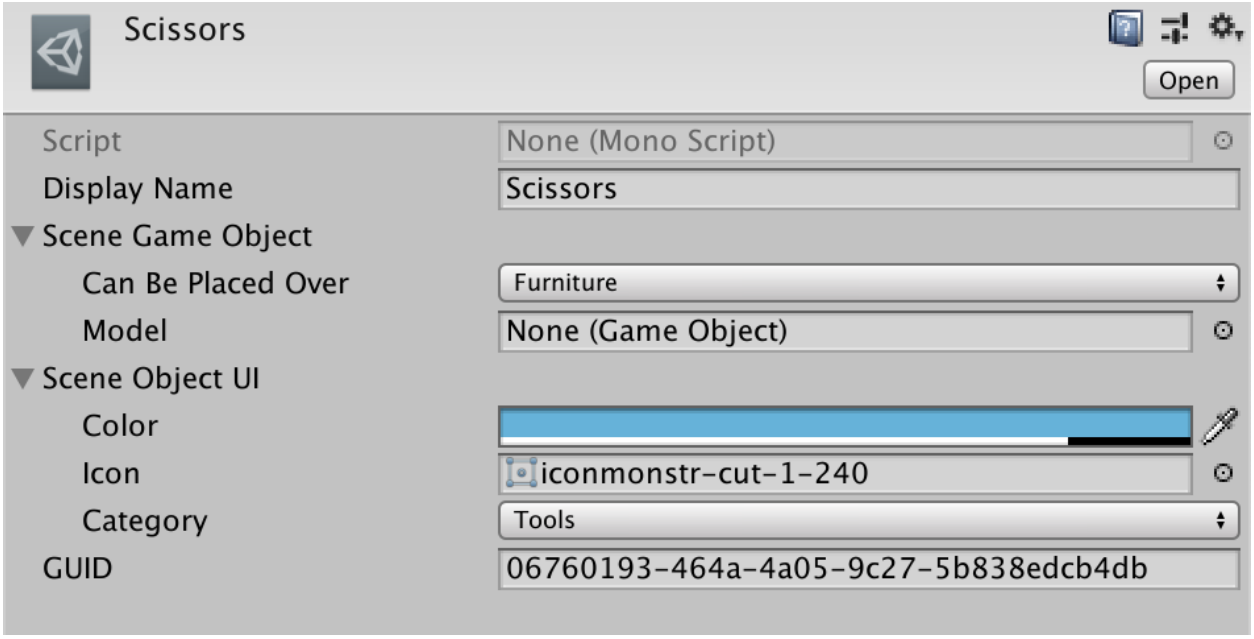

Figure 53 Scissors object definition.

Secondly, we have to create a prefab with the desired 3D model and the behavior it should have. For the behavior we will use the already defined `ToolSceneGameObject` which is a simple `SceneGameObject` that implements the `ITool` interface and therefore it has an associated tool type.

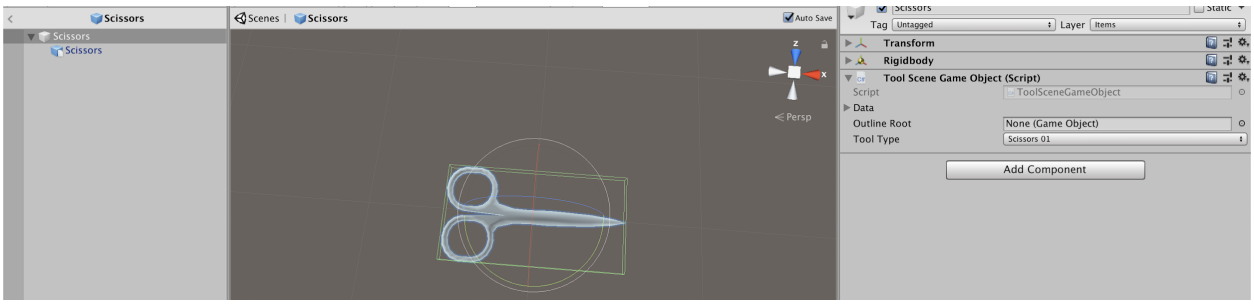

Figure 54 Scissors Object Prefab.

Furthermore we will add to this object the necessary VRTK [23] (A productive VR Toolkit for rapidly building VR solutions in Unity3d that provides us most of the necessary interaction with

objects and sets up a layer of abstraction with the concrete VR platform) components to allow the user to grab it and place it on specific zones.

Finally, we will need to assign the created prefab to the scissors object and add this new object to the object set and we can use it.

Figure 55 On the top addition to the object set of the scissors object. On the bottom view on the 3D scene of the added scissors.

6.1.2. Define Nodes

For this example, we will need also to define one extra node that receives one scene object and compares it with an enum of tools.

```
public class CompareTypes : NTNode {
	[Input(ShowBackingValue.Never, ConnectionType.Override)] public
SceneGameObjectReference scneObject;
	[NTOutput] public bool result;
	[NTInput] public Tools tool;

	public override object GetValue(NodePort port) {
		if(port.fieldName == nameof(result) ){
			SceneGameObject scgo = GetInputValue<SceneGameObject>
			(nameof(scneObject), null);

			if(scgo != null && scgo is ITool){
				ITool t = (ITool) scgo;
				return t?.GetToolType() == tool;
			}
			else
			{
				return false;
			}
		}
		else
		{
			return null;
		}
	}

	public override string GetDisplayName(){
		return "Object is type of";
	}

	public override int GetWidth(){
		return 300;
	}
}
```

As we can see this node simply compares the type of an object which implements the interface of ITool with an enum of listed tool. However, the tools should implement the ITool interface otherwise they will cannot be compared as needed for the exercise. With this code the node already appears in the system and can be used by final users (**R2**).

6.1.3. Composing the session

We will start with the user defined variables for this exercise:

- **Grade**: Trainee performance giving a score 0 - 10.
- **Clean hands:** Has trainee washed his/her hands?
- **Gloves on**: Has trainee put on the gloves before manipulating any instrument?
- **Cover table on:** Has the table cover been put on the table?

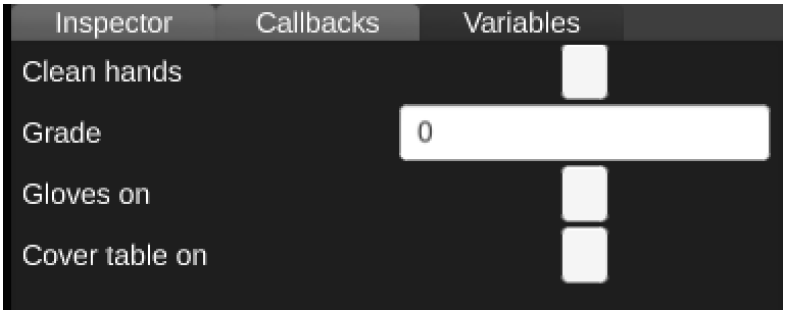

Figure 56 User defined variables.

Firstly, in the graph of the sink we will set to true the variable clean hand when the trainee washes his/her hands:

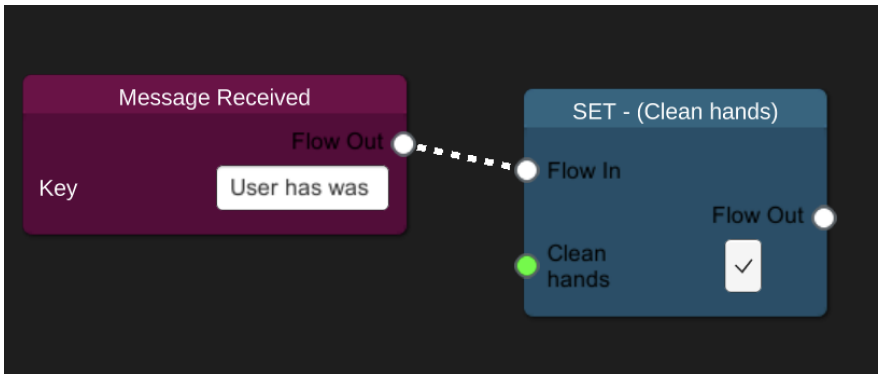

Figure 57 Sink graph.

Secondly the graph of the gloves will check if the trainee has his/her hands clean and if they have not, they would fail the exercise as it will break the basic protocol. Otherwise we will set the corresponding variable to indicate that the user has the gloves on:

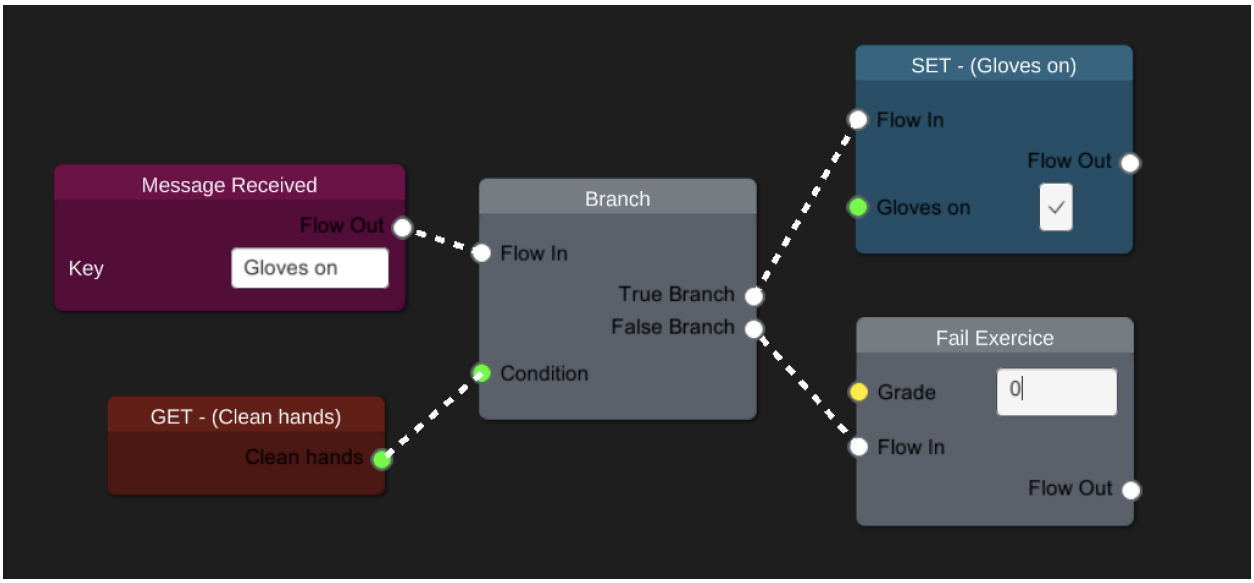

Figure 58 Gloves on graph.

Thirdly for the table cover and surgical objects we will define a behavior similar to the gloves one but this time checking if the user has both gloves on and the hands clean. If they have not we will automatically fail the exercise.

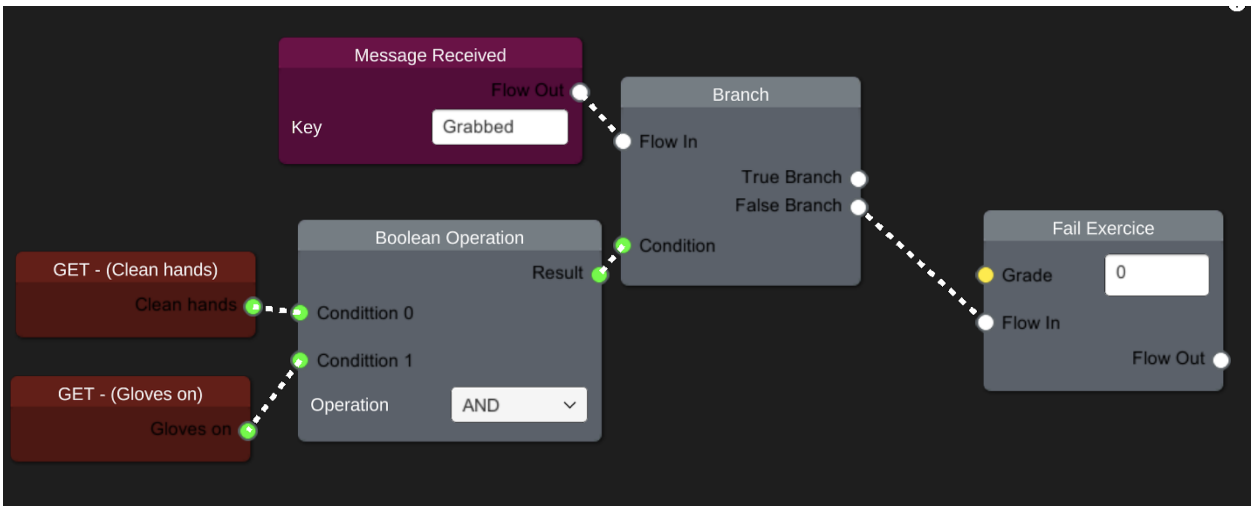

Figure 59 Table cover and surgical instrument graph.

On the surgical table we will set to true the cover variable when the user puts on the cover table. Note that the table will not let the user put to cover if any object is already on the table.

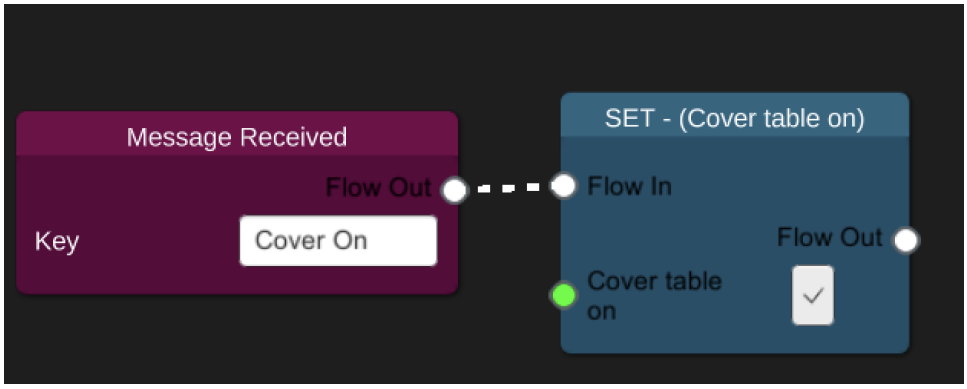

Figure 60 Table Cover on graph.

Finally, for the scene graph we will first send a message that will trigger all the checks and later on if the grade is enough the trainee will pass the exercise or otherwise will fail it:

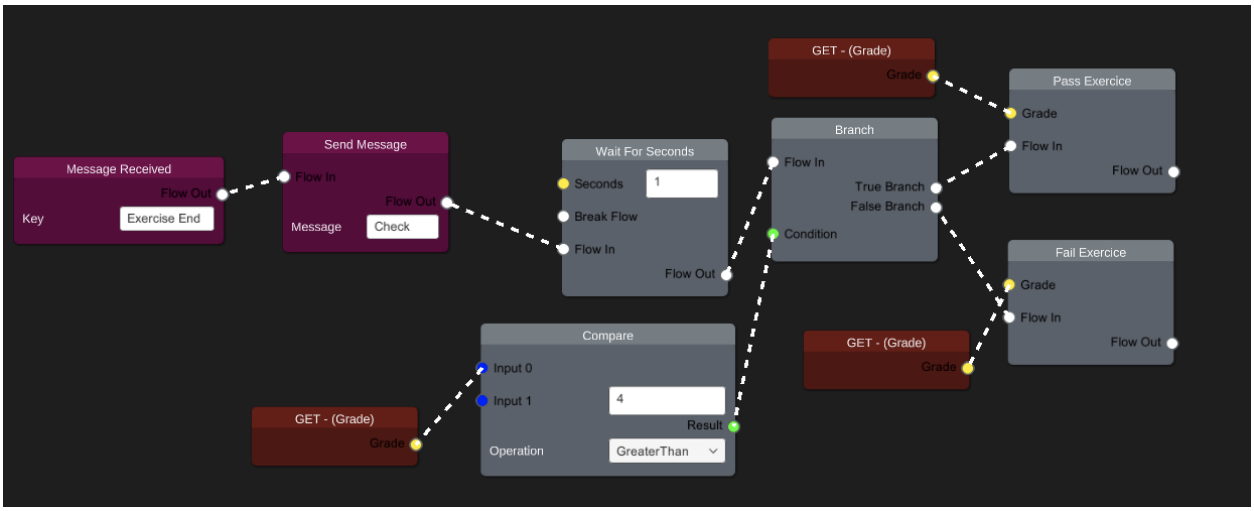

Figure 61 Session graph: Check for the accumulated grade and end the session.

The check message will trigger several execution flows that will check if each slot on the table has the appropriate instrument. On the following figure we will see the complete version of the check for one slot and a simplified version with a grouped node that help us to make it easy to repeat the node set for all the table slots.

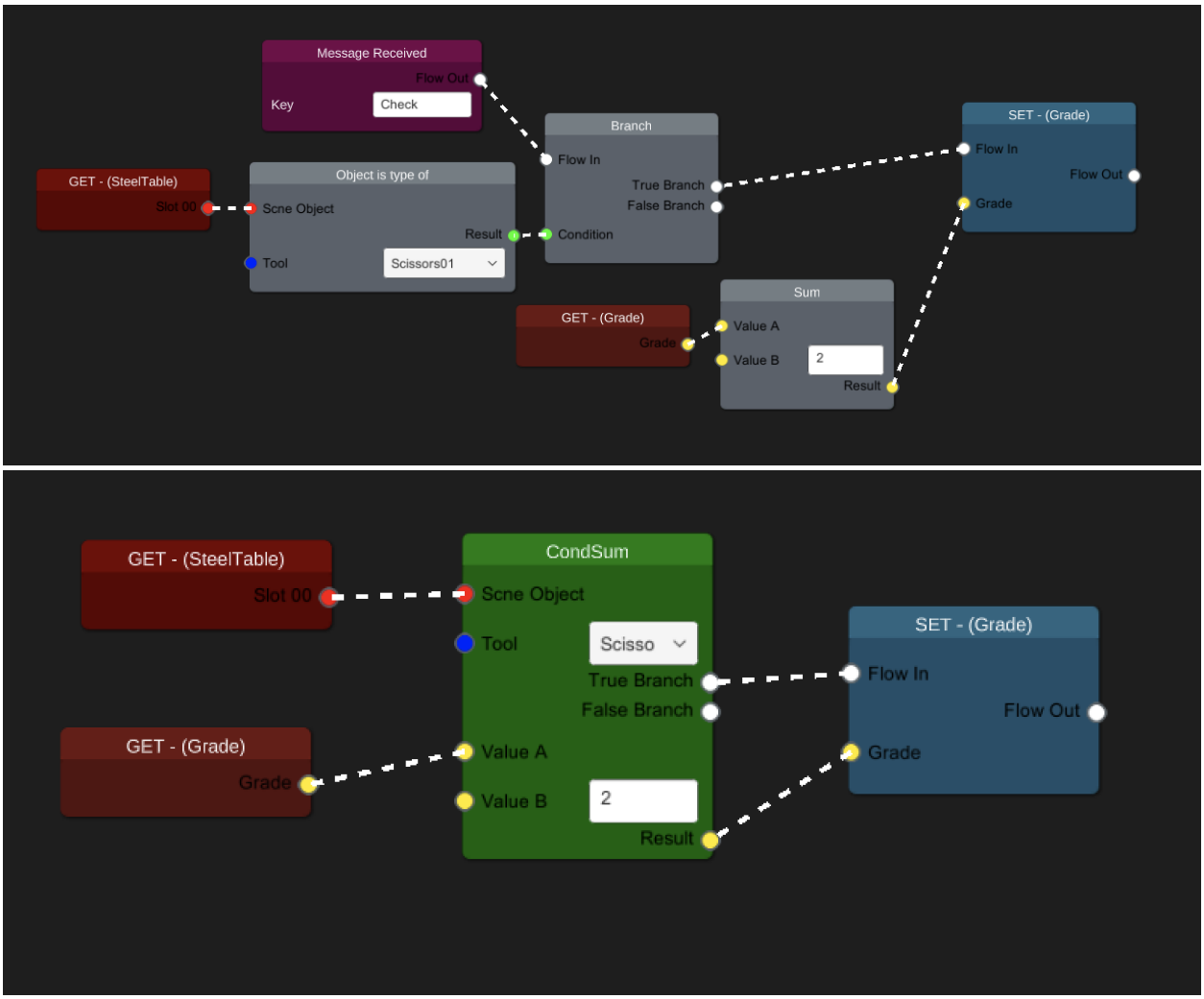

Figure 62 On the top: Nodes to add 2 points if the object is correctly place. On the bottom simplified version of the same behavior with a grouped node.

For the table cover we will check if it is placed and if it is not, we will subtract two points of the final grade.

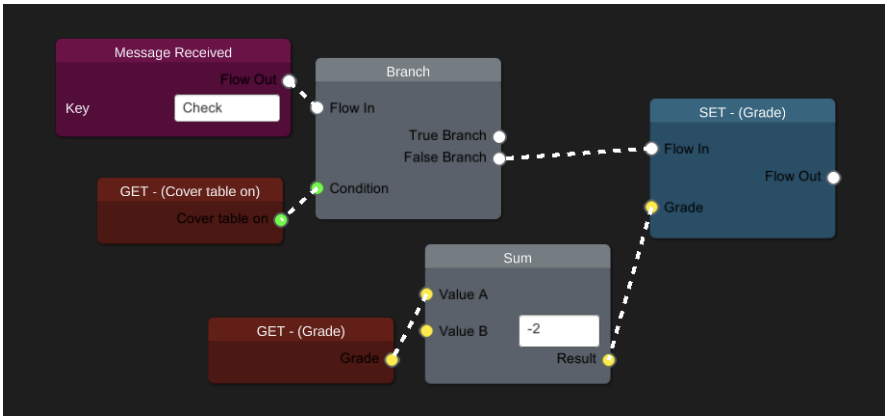

Figure 63 Missing table cover on demo session.

7. Conclusions and Future Work

To conclude we can affirm that the designed system fulfills all the listed requirements as:

- **R1** is satisfied since all the design of the authoring training session process was made specifically for end users. Furthermore, it does not introduce advanced programming concepts and tries to simplify the process as much as possible.

- **R2** is satisfied as shown on the extensibility and results sections. This requirement is one of the most important as it will allow us to extend this project and make it completer and more functional.

- **R3** is satisfied by the fact that the final system can be exported as a single application that do not need extra programs or software to run.

- **R4** is satisfied as we are using a cross platform development framework and unity which allow us to handle this multi device. Despite the fact that we have to make multiple compilations for the different devices.

Although this final master thesis was focusing mainly on building a solid base for authoring training sessions it also provides an example of a simple session and the process that should be followed for creating these sessions.

As future work we propose to expand the system and build demo sessions to identify and complete all the base pieces needed for a real-world use of it. Furthermore, user studies will be needed to validate and adjust the final UI for the proposed system.

A close view of the future tasks that will improve the system will be:

- Expand node set to allow end users to compose a high variety of sessions.

- Expand object set to allow more diversity on composing sessions scenarios and interactions between objects.

- Default behaviors for objects for a realistic and accurate behavior.

- User registration and cloud save will allow users to compose sessions on one device and export them to other devices without any cost nor additional knowledge.

- VR interface for selecting session on mobile devices will be important as right know we should manually place the folder of the session on the devices that are non-desktop based.

- Experimentation on better interaction on VR environments as this will allow better presence sense and can improve the performance of the training sessions.

8. References

[1] C. Scaffidi, M. Shaw, and B. A. Myers, “Estimating the numbers of end users and end user programmers,” *2005 IEEE Symp. Vis. Lang. Human-Centric Comput.*, pp. 207–214, 2005.

[2] H. Lieberman, F. Paternò, M. Klann, and V. Wulf, “End-User Development: An Emerging Paradigm,” in *End User Development*, 2006, pp. 1–8.

[3] “(PDF) Domain-Expert Users and their Needs of Software Development.” [Online]. Available: https://www.researchgate.net/publication/237088702_Domain-Expert_Users_and_their_Needs_of_Software_Development. [Accessed: 19-Jan-2019].

[4] a. K. Dey *et al.*, “a CAPpella: programming by demonstration of context-aware applications,” *Proc. SIGCHI Conf. Hum. factors Comput. Syst.*, vol. 6, no. 1, p. 40, 2004.

[5] B. Argall, S. Chernova, M. M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Rob. Auton. Syst.*, vol. 57, pp. 469–483, 2009.

[6] A. K. Dey, T. Sohn, S. Streng, and J. Kodama, “iCAP : Interactive Prototyping of Context-Aware Applications,” *LNCS*, vol. 3968, pp. 254–271, 2006.

[7] G. Ghiani, M. Manca, F. Paternò, and C. Santoro, “Personalization of Context-Dependent Applications Through Trigger-Action Rules,” *ACM Trans. Comput. Interact.*, vol. 24, pp. 14:1-14:33, 2017.

[8] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, “Handbook of Robotics Chapter 59 : Robot Programming by Demonstration,” *Robotics*, vol. chapter 59, pp. 1371–1394, 2007.

[9] J. Aleotti and S. Caselli, “Grasp recognition in virtual reality for robot pregrasp planning by demonstration,” *Proc. 2006 IEEE Int. Conf. Robot. Autom. 2006. ICRA 2006.*, pp. 2801–2806, 2006.

[10] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman, “Practical trigger-action programming in the smart home,” in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI ’14*, 2014, pp. 803–812.

[11] J. Huang and M. Cakmak, “Supporting mental model accuracy in trigger-action programming,” in *UbiComp*, 2015.

[12] J. Brich, M. Walch, M. Rietzler, M. Weber, and F. Schaub, “Exploring End User Programming Needs in Home Automation,” *ACM Trans. Comput. Interact.*, vol. 24, pp. 1–35, 2017.

[13] M. Resnick *et al.*, “Scratch: Programming for All,” *Commun. ACM*, vol. 52, pp. 60–67, 2009.

[14] “{Unreal Engine} aaa.” .

[15] M. Takatsuka and M. Gahegan, “GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis And Visualization.”

[16] J. Meyer-Spradow, T. Ropinski, J. Mensmann, and K. Hinrichs, “Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations,” *IEEE Comput. Graph. Appl.*, vol. 29, no. 6, pp. 6–13, 2009.

[17] M. Nagendran, K. S. Gurusamy, R. Aggarwal, M. Loizidou, and B. R. Davidson, “Virtual reality training for surgical trainees in laparoscopic surgery,” *Cochrane Database of Systematic Reviews*. 2013.

[18] E. C. Hamilton *et al.*, “Comparison of video trainer and virtual reality training systems on acquisition of laparoscopic skills,” *Surg. Endosc. Other Interv. Tech.*, 2002.

[19] N. E. Seymour *et al.*, “Virtual reality training improves operating room performance: results of a randomized, double-blinded study,” *Ann. Surg.*, vol. 236, no. 4, pp. 458–464, Oct. 2002.

[20] T. P. Grantcharov, V. B. Kristiansen, J. Bendix, L. Bardram, J. Rosenberg, and P. Funch-Jensen, “Randomized clinical trial of virtual reality simulation for laparoscopic skills training,” *Br. J. Surg.*, 2004.

[21] C. R. Larsen *et al.*, “Effect of virtual reality training on laparoscopic surgery: Randomised controlled trial,” *BMJ*, 2009.

[22] F. Mantovani, G. Castelnuovo, A. Gaggioli, and G. Riva, “Virtual Reality Training for Health-Care Professionals,” *Cyberpsychol. Behav.*, vol. 6, pp. 389–395, 2003.

[23] TheStonefox, “VRTK,” *GitHub repository*. 2018.

9. Appendix

Source code available on github: https://github.com/sigr3s/NursingTraining