UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science
Computer Science Bachelor

Interfacing users and AIs in virtual realities

Supervisor
Prof. Antonella De Angeli

Co-Supervisor
Dott. Paolo Busetta

Candidate
Pesavento Luca

Academic year 2012/2013

Index

1. Introduction 6

2. Related work 7

3. Conceptual Model and Data Structure 7
   - 3.1. Conceptual Model 7
   - 3.2. Data Structure 8

4. Conceptual design 8
   - 4.1. Interaction pattern 9
   - 4.2. Requirements 10
   - 4.3. Usability 10

5. The iterative approach 10
   - 5.1. Methodology 11
   - 5.2. Important choices and discarded designs 12
     - 5.2.1. The in-game interface and the advanced GUI 12
     - 5.2.2. Showing agents' state 12
     - 5.2.3. Labels and Icons 13
     - 5.2.4. Summary view 16
     - 5.2.5. The Map 16
     - 5.2.6. The Behavior Drawer 17
     - 5.2.7. Stopping the simulation 17
     - 5.2.8. Plans 17
     - 5.2.9. Locking Agents 19

6. Physical Design: Prototypes 19
   - 6.1. Interaction process 19
     - 6.1.1. Starting from scratch 19
     - 6.1.2. The advanced menu and the drawer 21
     - 6.1.3. Handling the tablet 21
     - 6.1.4. Clusters of agents 21

7. Requirements and Heuristics 23
   - 7.1. Requirements 23
     - 7.1.1. In-game GUI 24
     - 7.1.2. Advanced GUI 26
   - 7.2. Heuristics 27

8. Conclusions 28

Abstract

Serious games are increasingly used for training emergency operators. The new generation exploits techniques of AI to render the simulation more realistic and efficient. The trainers become sort of movie directors, who animate complex scenarios and manage the game flow. However a main problem is how to allow the operator to monitor and control the AI status.

This paper reports a project aimed at designing a user interface for displaying A.I. status in serious game. The interface had to provide feedback about the AI status and allow the operator to perform quick adjustments in it. The design challenge was related to the introduction of a new interface in an already rich and complex visual environment. the project was done in collaboration within the PRESTO project, developed by Delta Labs of Delta Informatica.

The design followed a strict user-centered approach. I started with an analysis of existing interaction paradigms, then I examined the problem with a multidisciplinary team team including game designers, developers, and training experts. Then, I proceeded to create a software infrastructure to allow this kind of interface to be implemented. After this, I started a UI design process which ended with a mock-up that shows the interface and how it is intended to work. The process involved the iterative design of many paper sketches and hand-drawn mockups, which were iteratively evaluated with experts from the Delta Labs. The final design ended in a low-fidelity prototype, which underwent a further evaluation phase.

A major result of this work is the proposal of a set of requirements and heuristics to evaluate the design that will be presented too, as well as a complete description of how to structure the interaction and some low fidelity prototypes. The major decisions and discovering will also be mentioned in the discussion.

This set of design artifacts and knowledge could be useful to anyone designing a virtual reality environment with advanced AI, with requirements that involve feedback of the system status, notably feedback of how the AI is behaving.

1. Introduction

I did this research within the PRESTO (Plausible Representation of Emergency Situations for Training Operations) project, as a follow up of my internship in Delta Labs, a Research Laboratory that develops agents based artificial intelligence for serious games. My project was to create a first and rough interface, intended mostly for debugging and for computer scientists. I then proceeded with a thesis to propose some design ideas for an end-user development system. In this case end-users are safety engineers who are unlikely to have programming knowledge. There is little research on how to present information related to artificial intelligence, so this wanted to be a preliminary study for a state-of-the-art product.

PRESTO aims to provide a layer of artificial intelligences for virtual realities, notably for serious games. A serious game is designed primarily for purposes other than pure entertainment, generally for training purposes for various scenarios, from emergency trainings, to defense, health care, and so on and so forth. Learners usually play these games that allow complex scenes that could not be simulated in the reality to be reproduced and analyzed with an expert trainer [19].

PRESTO developed from the JACK and CoJACK projects [4]. It is being developed within XVR [5]. PRESTO is at its first steps and is being developed to work on the XVR serious game. Gaining in popularity for emergency trainings worldwide, XVR is more similar to a virtual reality than to a serious game, because there are no goals or rules.

XVR provides an environment and a series of entities. It does not natively support artificial intelligence, and it requires a person to animate entities in the scene. To create a serious game, this person, also called simulation trainer, has to “animate the scene”. By moving, adding and removing entities, the environment can be animated, so that the game flow adapts to the students behavior. Nothing in the game is scripted or autonomous, the game trainer has to literally move everything, like a puppeteer who animates a scene while students play and learn.

PRESTO aims to enrich virtual realities with AI-driven “agents” that autonomously “think” and behave as expected, to ease the interaction for the simulation trainer, as handling complex simulation is harder as the number of entities increases. In this setting the simulation trainers’ role becomes more similar to that of a “movie director” than that of a “puppeteer”. They script the scene and and can, in case, tweak it. As a result, there is the need of an interface to allow the the simulation trainer to monitor and control the AI.

During the Internship at Delta Labs I implemented an infrastructure for a graphic interface, and developed a simple GUI to allow simulation trainers to accomplish basic operations. The Interface was initially meant to be usable by simulation trainers, but it proved to be hard to use for non-developers.

This paper is structured in this order: First, I will discuss the technologies and the data structure. Second, I will discuss the first ideas and the initial design of the interface. Third, I will discuss the methodologies used for the design evaluation process. Fourth, I will discuss the final design and present prototypes. In the end, I will show requirements and heuristics used to design and evaluate the interface.

2. Related work

Serious games are gaining in popularity lately, but many of them do not implement an artificial intelligence, and the ones who do use unfriendly interfaces that look like debug tools. No studies on interfaces for end-users of artificial intelligence related to

virtual realities have been published. There are some studies on interfaces for artificial intelligence in web services [3], but they are hard to adapt to this scenario. Popular serious games, such as VBS2 [6], implement artificial intelligence as series of scripts which are not user-friendly. Other non-serious games which implement interfaces for artificial intelligence, such as The Sims [7], mainly focus on entertaining the user, which is not the main goal of a serious game.

There is some research on human simulation solutions, notably products like DIGuy, Xaitment, and SpirOps AI [8][9][10]. None of this solutions offer the level of customization we wanted, and the ability to dynamically change behaviors, beliefs and somehow "tweak" the AI as the user prefers.

However, a notable amount of research in HCI can be used to inspire the design of an interface for end-user development. Notably, Jakob Nielsen's studies on heuristics for human interfaces, or Thomas W. Malone's heuristics for enjoyable interfaces. I've also used famous frameworks, such as Think Aloud [1] [11] [12].

3. Conceptual Model and Data Structure

In this chapter I will discuss how the PRESTO Infrastructure works and what data is meant to be available to the simulation trainer. The focus will be on how AI agents work and how the simulation trainer could interact with an agent. I will also discuss part of the software infrastructure, notably how the GUI communicates with PRESTO.

3.1 Conceptual Model

AI agents can be thought as mindsets, which may be composed by beliefs and decision making processes. Entities in the game can be associated to an AI agent, becoming what we will call "an agent".

Regarding how the AI Agent work, an example could be examined: a person entity (a simple model in the game) is being controlled by an AI-agent. This agent is following a fire-fighter behavior (which is the AI agent's behavior: a set of beliefs and goals), and is currently searching for a fire extinguisher (which is the state of the AI agent, each behavior has a set of unique possible states).

As illustrated in the example, AI agents have behaviors: patterns that describe how the agent behaves. Example of behaviors include firefighters, medics, civilians, and so on and so forth. Behaviors are composed of states, e.g., in what stage the behavior is, or, what the agent is doing. For example, a state of a nurse behavior could be "searching for wounded people" or "distributing vaccines".

What is called a BDI (Belief Desire Intention) [13] paradigm is used to model the actions of the AI agents. In the following chapters, I will refer to behavior as a complex set of capabilities that are modeled around goals, procedures and the knowledge a certain character has.

The training operators should be able to see what behavior an agent is following and in what state the behavior is. This provides the essential information to understand what an agent is "doing". The information provided could be summarized in a sentence such as "this <nurse> is <distributing vaccines>" or "this <firefighter> is <searching for an extinguisher>". They can also change the behavior of an agent, but not its current state.

Agents (An entity animated by an AI agent) can be AI-Enabled or AI-Disabled. Disabling an agent stops its cognitive process, making it a static model. AI-Disabled agents can be resumed. This option may come handy to the simulation trainer in the case an agent does not behave as expected, to take back control of the simulation.

3.2 Data structure

The infrastructure is implemented with .NET technologies, mostly C#, and runs over Unity3D and consequently, Mono [16][17] [18].

PRESTO uses an Ontology system to abstract in-game concepts. The ontology is used to map entities, locations and behaviors to their relative real-world meaning, allowing for more complex reasoning, scene control, and constraints to be implemented. Ontologies are formalized in an OWL format and managed with the Sesame triple store. This is a key feature for the infrastructure to work. However, the simulation trainer should not have to interfere with the ontology for handling agents [14] [15].

The software infrastructure I worked on during my internship is meant to allow for future GUIs to communicate with PRESTO. More in detail, since PRESTO's infrastructure in composed of many elements (the ontology, the scene, the environment, agent factories, etc), developing a GUI to work directly on this infrastructure meant having to directly work inside each of these components, requiring an advanced knowledge of how they work and making the project more prone to errors and harder to maintain whilst making the development harder. What I did was to create a "GUI manager" which is an object that mediates between the GUI and the infrastructure, so that who develops the GUI does not have to go deeply inside the implementation of PRESTO to understand how to get the information. Moreover, the work-flow of the infrastructure does not resemble reality or the general idea the developer could have, meaning that some particular information may be stored in remote parts of the infrastructure or that to access a particular property a particular approach must be followed. And furthermore, the data is not represented in a human-readable manner, meaning that behaviors, states, models, etc. are composed of complex strings that no one other than the developers would understand.

A part of the work was to "wrap" the infrastructure components into objects that resembled the real-world conception of the system, making it easier to work on the platform and making data easily available. Complex strings are reworked to be human-readable.

This allows a better scalability of the project whilst reducing the risk of errors and easing debug. The approach of having an in-between object that mediates information between the complex infrastructure and the GUI allows for the communication channel to be abstracted and thus allows the development of a GUI external to the project infrastructure. More specifically, this GUI manager could mediate information over a network, and between platforms. This allows for the GUI to be developed with other technologies on other platforms and be able to communicate with PRESTO through the GUI manager. For example, a GUI developed for an Android tablet in Java could communicate with PRESTO by " remotely talking" to the GUI manager.

4. Conceptual design

In this chapter I will discuss the first interface, how it was meant to be usable by simulation trainers, its usability problems and why it was kept as a developers-only interface.

The goal of the first interface was to provide simulation trainers a way to manage the AI. There was no current way to do that, other than altering the code. The first interface had three points:

1. Display an overview table describing the agents' behavior
    a. Allow agents to be stopped / resumed (AI-Enabled/AI-Disabled)
    b. Allow basic behavior handling.
    c. Allow agent removal.

2. Display an overview table describing the entities' state and their associated ontologic meaning

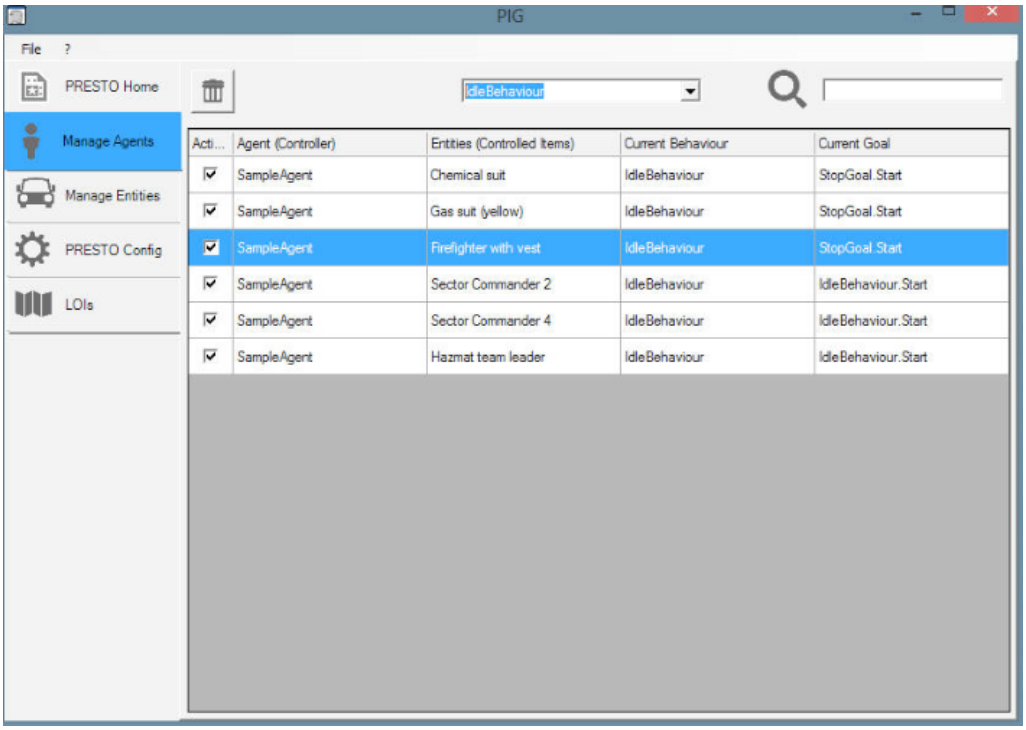
a. Allow creation of agents from entities.

3. Provide tools, such as a settings panel and a LoI (Locations of Interest) manager.

As PRESTO aims at developing a platform-independent technology, which can run on multiple virtual realities and game engines, the GUI was designed as as a completely external product from XVR, to make clear that PRESTO was separate from XVR. This GUI was developed, using technologies supported by the platform at the time (Windows Forms over Mono). Constraints included the impossibility to display interface or artifacts during the game or in-game dialogs. It was not possible to display advanced animations or complex graphics objects. Complex GUI libraries (such as WPF) were not supported. It was not possible (nor desirable) to adopt the game visual design.

4.1 Interaction Pattern

The GUI developed as a stand-alone window inside Windows OS. The interaction pattern was structured as follows: a series of tabs gave the simulation trainer access to agents, entities or settings. Each tab showed a different panel that contained a list of the selected items. Right clicking an item displayed advanced options, such as behavior change.



The first interface, which was implemented as an external tool.

4.2 Requirements

The requirements at the beginning of the project were rather simple. Below I will refer to the simulation trainer that animates the scene as to the “user”.

Functional requirements

1	Users should be able to view which entity is associated to an AI agent. This association is simply called “agent” and the user should be able to see all the agents currently in the game.
2	Users should be able to change agents behaviors.
3	Users should be able to change AI-agents state (AI-Enabled or AI-Disabled)
4	Users should be able to view agents’ goal.
5	Users should be able to view all the in-game entities and their associated ontology concepts
6	Users should be able to assign a new AI agent to an existing entity, thus creating a new agent.
7	When using the GUI, users should be able to understand which agent is which in the game,

Non-functional requirements

1	Selection must be consistent in game and in the GUI.
2	Text should be human-readable an not presented in codes or programming strings.

4.3 Usability

The major problem of this GUI is the fact that it imposed a complex interaction in order to be used. The fact of having a window for the game and a window for controlling the AI implied having to switch context every time an AI interaction was needed. The “alt-tab” process was cumbersome and made the interface rather annoying for the simulation trainer to use. Furthermore, showing the system state of the AI was nearly impossible, so every time the simulation trainer wanted to monitor the AI a context switch was necessary. This is one of the main result of this research: a completely external GUI is not a usable solution.

However, the developers started to use the GUI to perform debug operations and to accomplish task they could not do before, such as setting LoI’s and quickly changing the ontology and the settings. The GUI became in fact a very usable debug tool, and developers started to customize it with advanced debug tools.

This fact clearly showed that this GUI was usable by developers and programmers but NOT by simulation trainers (the end-users). The GUI was then kept as a tool for developers, while a usable interface for simulation trainers was researched.

5. The iterative approach

This section presents the design process aimed at proposing an interaction between the simulation trainers and PRESTO.

One of the most important constraints were that simulation trainers currently have no idea of what to expect from our product. Not only do they not understand how the AI is supposed to work but also how an AI-operated character should behave. The platform currently is not functional enough for a demo with them, so there is no way to demonstrate them how our product is intended to behave. As a consequence, it was impossible to interview simulation trainers for feedback regarding the interface as they do not know what to expect from the product and how to interact with it.

A wizard-of-oz test could have been done, however, when we tried to create it, it became obvious that the simulation would have been so distant from the reality that results produced during the interface evaluation were going to be totally unreliable. The simulation trainers could fail to understand how the system is supposed to work, thus giving unclear feedback.

Designing an interface on unreliable user data could have ended in a dramatic failure, and as a consequence the team decided to do experts-based evaluation for this stage of the design. Experts were developers of the platform who have been following simulations and know exactly how users interact with the product. Not only that, they also know what the final product aims to accomplish. The data produced during the evaluation with these people was considered reliable and used to redesign the interface.

5.1 Methodology

The research process was structured as a series of phases:

1. Research and understand users:

   Research was made on how the simulation trainers currently interact with such products on the market. Notably, research was focused on games which require a "master" or "puppeteer": a person that animates the scene by moving, adding and removing entities. This person would be the target of the product, and the one we were designing the interface for. We wanted to transform the "puppeteer" into something like a "movie director" that controls actors with limited automatic capabilities. Consequently, we analyzed how the interaction is currently performed.

2. Define requirements:

   I defined a list of requirements of the interface. The requirements were based on how the simulation trainer currently sees the game and on how he currently interacts with it, focusing on visual feedback and on giving the simulation trainer the ability to guess the system status at a quick glance. Furthermore, both normal and extreme situations were examined, notably scenes with high quantity of information to display.

3. Define interaction:

   I proposed how to structure the interaction for the simulation trainer to perform tasks. Many paradigms were examined and only the ones that fulfilled the requirements were eventually kept and analyzed more in-depth. This phase ended with a design of how the simulation trainer was supposed to execute both common and unusual tasks.

4. Design UI and prototype:

   With the results of the previous phase one or more UI designs were made and prototyped. From what I learned, I was able to design an UI which was functional and matched the previous requirements. A low fidelity prototype was made.

5. Receive Expert's feedback and evaluate.

   A list of specially designed heuristics were used to evaluate the design in the first phase. The prototype was then presented to the team and an Expert analysis was conducted for user feedback. The method we used for evaluating the UI was asking the experts singularly to perform tasks with the Think Aloud method. The interviews were recorded and then consulted for evaluating the UI design. At the end of every interview the experts were asked for constructive feedback, opinions and suggestions.
   The results were then annotated, requirements were modified and the process was restarted from phase 3.

Steps 3, 4, and 5 were recursively iterated as feedback was collected, eventually leading to a design that was considered optimal and was kept as final and proposed in this document.

5.2 Important choices and discarded designs

Here I will discuss discarded design and why certain decisions were made. I will also explain important choices and results of the evaluation process.

5.2.1 The in-game interface and the advanced GUI

On one hand, it became obvious that designing an in-game interface would have imposed strict limits on the interaction, making complex actions hard to carry out. This is due to the interface mediating information on top of the game interface, which adds complexity in performing tasks. In other words, displaying a piece of information makes other information concur for visibility, and the result is that every piece of information has less visibility and is less likely to be noted.

On the other hand, a completely external interface (E.G. the interface in another window) is also not a viable option as the interaction process requires the simulation trainer to "tab" back and forth many times even to carry out vary simple and basic tasks.

Extending the In-Game GUI with an external advanced GUI is the main point of my research: the idea is having a tablet show advanced information, and when needed, used to carry out exceptional tasks that can not be done within the in-game GUI. The advantages of both paradigms can be achieved without many of the disadvantages, as with clear requirements the split-attention problem can be countered.

5.2.2 Showing agents' state

Showing which entity is controlled by an agent is important. This information should be provided in a non-invasive way, however it must be clear what the information means. The best way to provide this information, which is bound to each entity in the screen, is to show an icon near or on top of the entity, to indicate a state of autonomy or autonomous thinking. The icons would be positioned near the head of characters, as this is where people expect the reasoning process to be. In the rare case that an AI agent is attached to what the simulation trainers do not perceive as a person, the the icons could be positioned anywhere near the entity.

The problem was finding an icon that gives the simulation trainer the ability to tell which entity is autonomous, what iconography to use, and how to distinguish between AI-Enabled and AI-Disabled states.

The first idea was to use a semaphore metaphor: green means the agent is AI-enabled, red means it is AI-Disabled. The problem here was that yellow had no intuitive meaning. While we could have used it to show other feedback, like when an agent is active but waiting for something to happen, it still was not clear what it was for.

The second idea had a two-way semaphore, that could be either green or red. The problem here was that it is not necessary to show one light when the other is on.

The third idea, that was a simple light that could be green for "on" or red for "off". The three ideas above solve the problem of telling which agent is AI-Enabled and which is not, but the "semaphore" metaphor resulted in a failure: the association between thinking and the semaphore was not clear. In other words, the simulation trainers would expect some imagery related to thinking and cognitive processes, not imagery related to traffic and vehicles (even if the semaphore is common in many other applications).

The fourth design was made, with comics like balloons. The idea was then improved to the fifth design, as it gives a better idea of thinking than the previous design. The bubbles are intended to be animated, making the agents look like they are thinking. As showed in the picture, the red color and an 'X' mark is used to indicate that an agent is AI-Disabled.

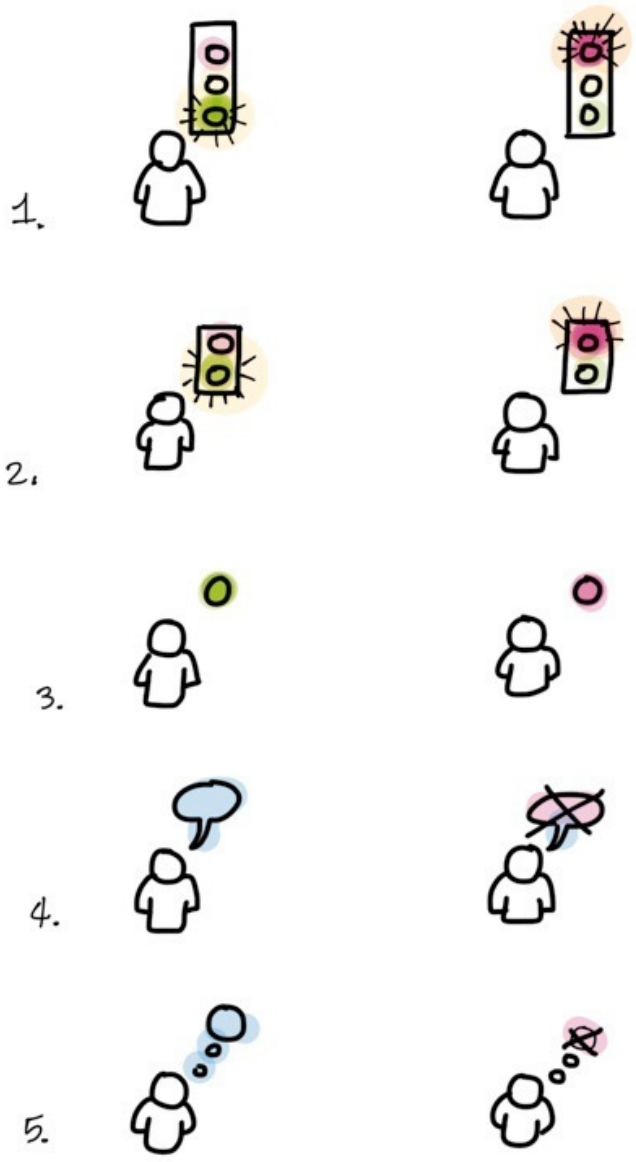
When an agent changes its current state, the ballon should animate differently, to indicate that something happened to the agent. The animation should include a little preview of the new state's icon. It is possible for the simulation trainer to tell exactly the series of actions an agent is doing just by looking at it, no interaction needed.

5.2.3 Labels and icons

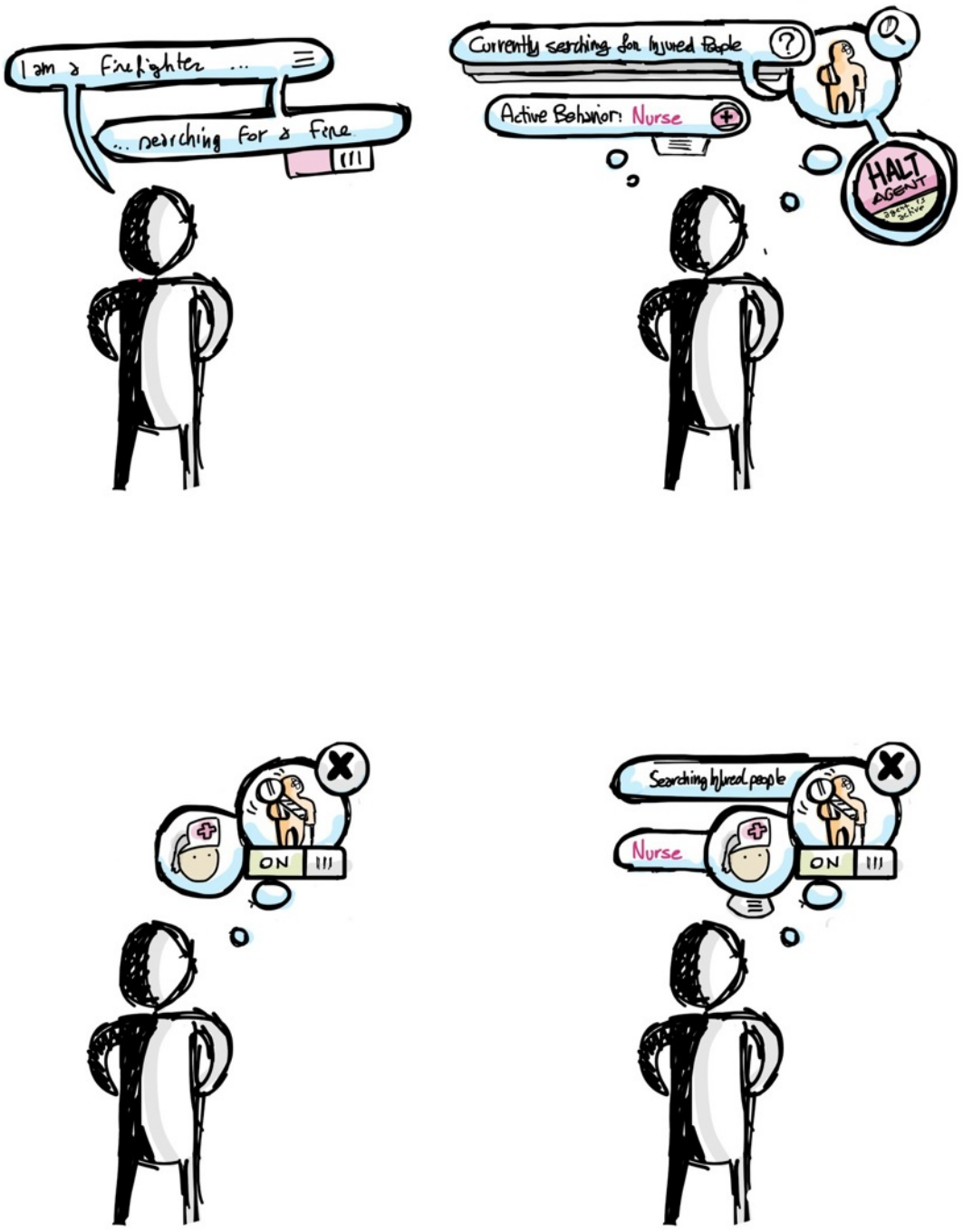
The process of showing information was thought to use labels and text to explain what agents are doing. A phrase inside a bubble could have explained what an agent were doing. This resembles storytelling patterns in comics. The problem is that reading the phrase takes some time and it is not possible to get the information at a first glance.

The design was then modified to include both icons and text. The beginner user should be able to read the labels and associate them with the icon, the expert user should recognize the icons without needing to read all the text. This intermediate design had the flaw of providing information that may not be needed.

While the new user should be provided with both the label and the icon, there is no need to provide the expert user with the label. The final design provides only icons, and for the new user, hovering the cursor on any icon shows a label with information. Not only does this removes clutter and information, simplifying the interface, but also stimulates the new user to memorize the meaning of icons.



The evolution of the agent's thinking iconography. On the left: AI-Enabled agents; on the right: AI-Disabled agents.



The evolution of the bubble interface: while the top left shows only labels, and the top right combines labels and icons, the final one (bottom left) only uses icons. Labels are displayed on mouse-over as shown in the bottom-right picture.

5.2.4 Summary view

During the evaluation, experts proposed the idea of having a tool to view quantitative information in a summary manner. This was discussed and the idea was added to the design.

This interface would show information regarding groups of agents. The purpose of this interface was to show quantitative information, such as “10 civilians are in panic”. This information belonged to the “advanced interface” and was quickly expanded to allow user-requested operations on such agents. For instance, one could want to “stop all” the agents in “panic”. This idea collected positive feedback, but there was a usability problem: the information were presented as text labels that required an unpleasant interaction to be read and understood.

The second design solves this problem and gives new interesting ideas: The focus is on providing visual feedback that can be understood at a glance, so the agents are represented by circles showing meaningful icons. Those circles are then grouped together in groups with other agents that share the same properties. Properties can be behaviors, states, or other more complex things, such as the behavior-state couple. For example, in a scenario with firefighters, nurses, and civilians, one should be able to see all the agents grouped in three groups.

The advanced operations are available for these groups, and an interesting fact is that, since agents are represented by circles, these circles can be drag-and-dropped over other groups to change agents’ behaviors. More complex ideas include providing a special circle if an entity is selected within the game, to give the ability to drag it over a behavior, thus crating an agent, or to drag an agent to an ‘X’ location to remove it.

The “overall view” with text (left) and icons (right) representations.

5.2.5 The map

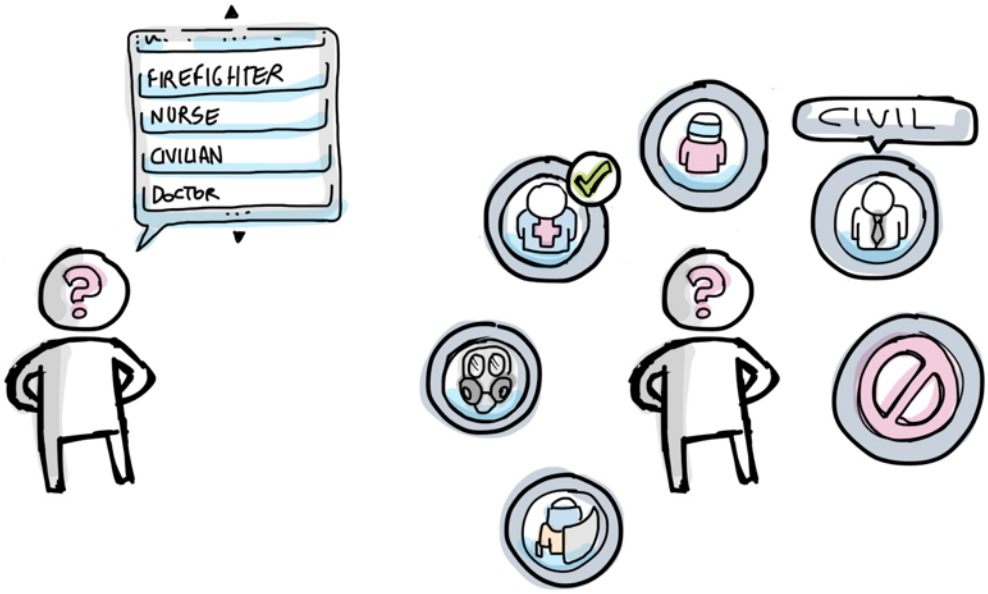
While testing the game, an interesting thing was noticed: there is not a map that gives a general view of the scene. We discussed about it, and decided it would be a great addition to our advanced interface, because it allows us to provide information about agents in a different way that was not possible in other ways. The map is intended to be a 2D representation of the scene from above. Competitors of XVR, notably VBS2, already support a feature such as this one, and the experts felt this feature would have improved the usability of the interface.

Buildings should be represented with their planimetry and the semantic meaning of locations should be showed by using colors. Pinch-in zoom should be supported, and zooming inside a location should show child-locations.
An obvious improvement of this map is to show agents with little icons, to show where they are. To keep consistency with the game, the selected agent is shown in a different way, and maybe, we could show a line to where the agent is heading.

Top left: the map view, from top. Top right: zoomed-in map. The legend changed and new locations are now displayed. Bottom: agent's pathfinding information have been added.

5.2.6 The behavior drawer.

How the user changes an agent's behavior is one of the most crucial points of the interaction. I will illustrate how this part of the design evolved during the evaluation process. The first idea was to have a list of possible behaviors, and the simulation trainer navigates the list to find the behavior he is interested in. The list was made of labels, which (see 5.2.3) were discarded soon. As icons offered a better usability, a drawer was designed to appear around the currently selected agent. The drawer would show icons for each behavior and labels when hovering icons with the pointer. Sub-menus could be supported. We think that this interface requires the simulation trainer to focus on the agent he is trying to modify, so an in-game zoom to focus the character would be a great addition.



Left: behavior drawer with labels. Right: behavior drawer with icons.

5.2.7 Stopping the simulation

The approach with this problem is slightly different from the others. We wanted to define a clear emergency way-out to stop the entire AI which was as immediate and fast as possible. We could have designed an in-game interface, but we thought something even faster was necessary. One can stop the entire AI process by pressing an “emergency” button on the keyboard. To tell the simulation trainer which one is the stop button, the key should be marked with a bright red sticker. This key should resemble an emergency button, like the ones on dangerous devices that instantly stops them. This interaction is fast because the simulation trainer does not need to search within the game for something that looks like an emergency tool. The bright red key is highly noticeable.

As a consequence, other AI actions may be bound to keys, such as “change selected agent status (AI-Enabled/AI-Disabled)”. This would require other stickers and may reduce the visibility of the previous button.

5.2.8 Plans

The system could and should support the possibility of having agents with a small number of “child” behaviors, or “plans”. A “Plan” is a subset of states of a behavior. A behavior may have particular plans that alter its functioning. For example a “nurse” agent may have the “routine” sub behavior, which makes her distribute medicines, the “emergency” sub behavior, which makes her evacuate the hospital, and the “panic” sub behavior, which makes her run away in panic. She is, in fact, a nurse, but these three sub behaviors describe how she is intended to work. This variety of possibilities could have been implemented as different behaviors, such as “frightened nurse” or “emergency trained nurse”, it is left to the reader to decide which best suits the scenario. I will show how the interface can be extended in a way to include these sub behaviors. The interface here discussed is intended as a future feature and will not be seen in the final mock-ups.

A nurse agent with three sub-behaviors. As usual, on mouse-over, labels are shown to help the simulation trainer.

5.2.9 Locking agents

One idea was to allow the simulation trainer to somehow “lock” an agent to a certain state. For example, the simulation trainer could lock a “civilian” agent in a “Panic” state, meaning that, even if the agent would normally change its state, it did not. This idea was discarded, as it allowed strange operations, such as locking particular states that would have given no meaning to the action. Moreover, it forced the AI to behave differently than expected, and could have resulted in unwanted behavior.

6. Physical Design: Prototypes

6.1 Interaction Process

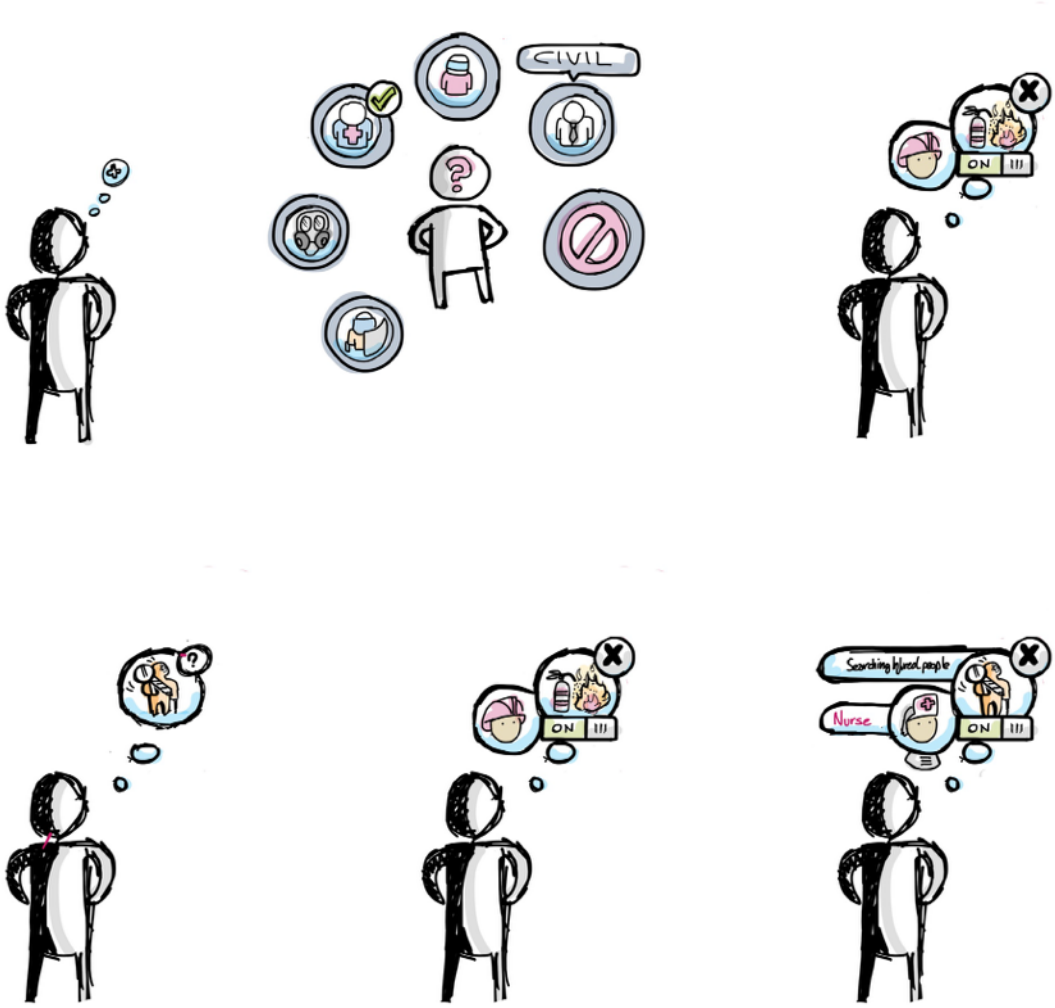
This chapter illustrates how tasks are supposed to be carried out and how the interaction is supposed to work. Important choices are motivated, while examples are provided for the reader to understand the design. Examples may be presented as stories. Refer to chapter 5.3 for more information.

6.1.1 Starting from scratch

The first thing the simulation trainer should notice is that when selecting an entity (for example a person), an animated “bubble” icon with a plus sign appears over the agents head. If clicked, the behavior drawer appears and the context of the game is zoomed

to the agent. If a behavior is selected, the game goes back to its previous state but this new agent has an animated series of bubble over its head. The agent then moves autonomously.

The simulation trainer is able to tell the new agent is AI-Enabled from the bubbles moving over its head. When the simulation trainer clicks the agent, the bubble changes its shape, but an advanced menu does not instantly appear. This is due to the fact that we want the simulation trainer to be able to handle entities like he does normally, and selecting them should not display unwanted information. When the simulation trainer moves the cursor over the icon, it changes for the third time, and a question mark appears. This is to make the simulation trainer understand he can click the bubble to get prompted the advanced menu. If he does, the advanced menu shows.



Top: the process of adding an AI agent to a person entity. Bottom: the process of accessing an agent's advanced menu.

6.1.2 The advanced menu and the drawer

This menu should be intended as the home screen for the agent. It shows three pieces of information: activation, behavior and status. Hovering icons shows the user a label with text. If that is not enough, hovering a label shows advanced information or a description in the external GUI. This is a remote case so the split-attention problem does not apply. Disabling an agent or clicking on the 'X' sign instantly closes the interface. Hovering the pointer on the behavior icon shows a different animation to indicate that the icon in click-able. Doing so opens the behavior drawer.

6.1.3 Handling the tablet

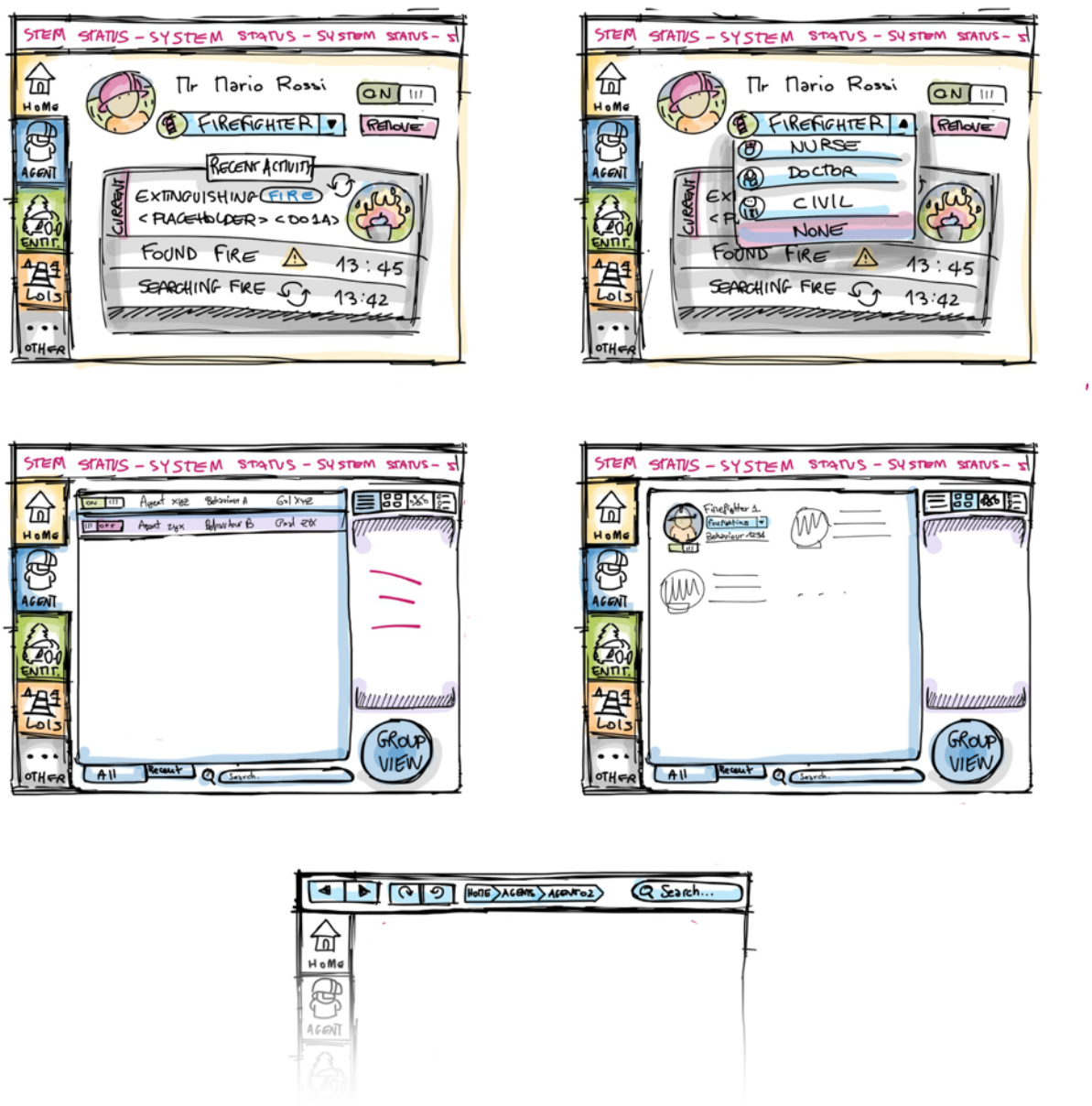
The split-attention problem is solved mostly requiring the possible interaction processes to be performed within the in-game GUI or in the tablet. This means that if the simulation trainer is using the in-game interface to carry out a task, he should not be prompted for input in the tablet. Vice versa, if the simulation trainer is using the tablet he should not be using the in-game interface and the game itself. Consider halting the simulation when the tablet is being used (this might interfere with the simulation, however). The only exception to this is that the tablet can be used rarely to display information to the simulation trainer. The simulation trainer could want to check the tablet, but should not have to grab the tablet in order to access this kind of information. An example of this is providing a text description of behaviors when they are under the pointer. This text information is useful only a few times, but if the simulation trainer does not know what a behavior does, it is fundamental.

The tablet also shows contextualized information, in the case the simulation trainer wants to have more information that the in-game GUI can not provide. The following table explains how the external GUI should contextualize information.

In-game state	Expected External GUI status
Nothing selected	Status View or Map
Entity or agent selected	Enhanced Status View or Map
Agent selected and advanced menu open	Advanced Agent information
Hovering a label	Detailed description of the label meaning

The advanced GUI allows the simulation trainer to carry out tasks such as agents management and group control. Moreover, it is intended to show a comprehensive list of agents and entities for faster browsing and enabled for searching and filtering. The external GUI supports *undo* and *redo* functionalities. This is a simple feature that did not require particular studies.

Furthermore, the GUI provides the functionalities described in paragraphs 5.2.5 and 5.2.6.

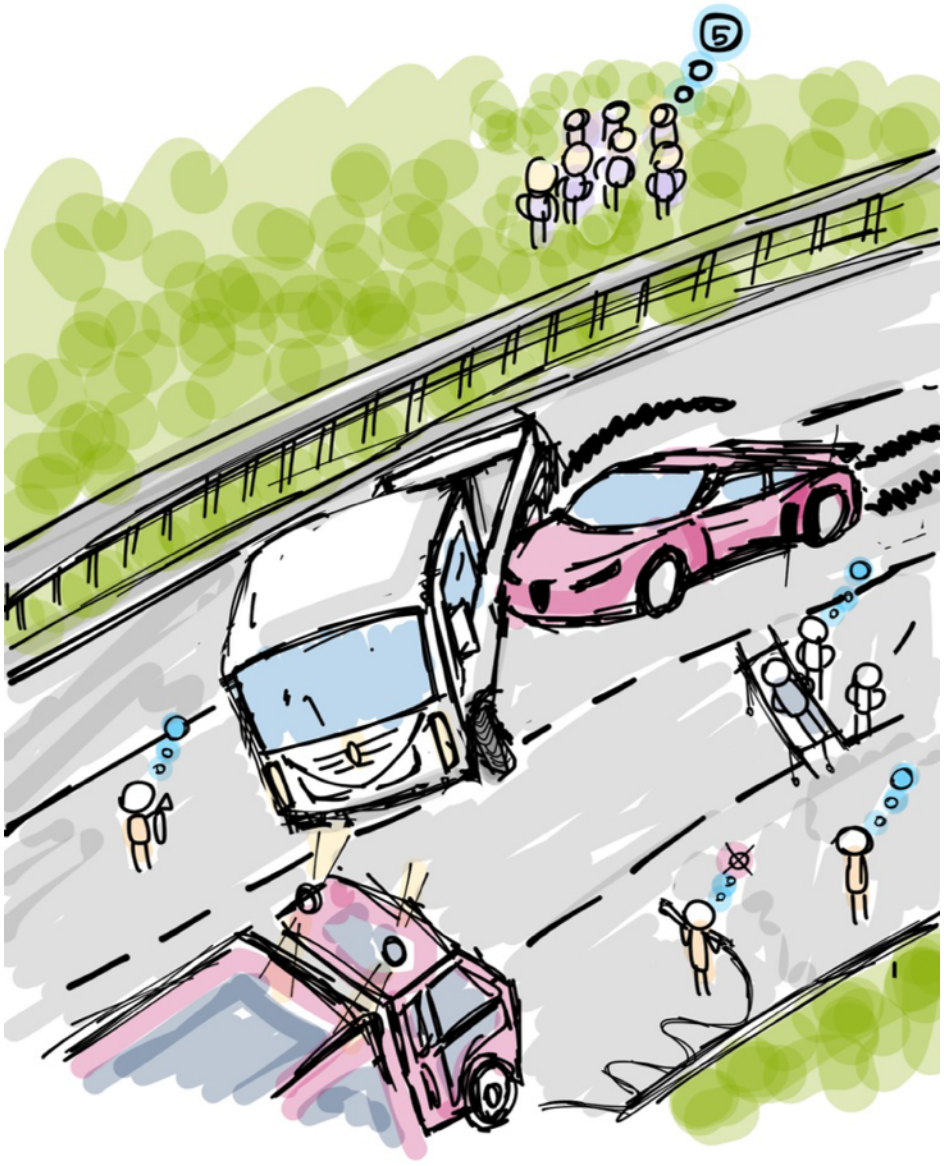


Top: advanced GUI with an agent selected. Middle: Agents view. Bottom: status bar for the tablet GUI.

6.1.4 Clusters of agents

The problem of having many agents on the screen is serious. If information is displayed for each single agent, there is a risk of displaying too much information, eventually overlapping icons and whatnot. A solution for this is grouping icons for groups of stacked agents. Referring to the “thinking” icons describe earlier, we could group them in a single icon and show a counter of

people that are thinking. Zooming in until there are not many agents on the screen causes the bubbles to split and behave as normal. Clicking on the stacked bubbles automatically zooms in enough to split the bubbles.



A simple scene. A group of agents can be seen on top. Their bubbles are grouped into one.

7. Requirements and Heuristics

7.1. Requirements

The requirements were established and inferred during the product design and the user evaluation phases of the design cycle.

7.1.1 In-Game GUI

User-Related Functional Requirements (the user is the simulation trainer):

1	The user should be able to tell which Entity is being controlled by an Agent with no or little interaction needed
2	The user should be able to interact with the game the same way he does with or without the in-game interface.
3	The user should be able to tell which Agent is active and which agent is stopped with no or little interaction needed
4	The user should not feel overwhelmed or be presented with too much information when many agents are present in the same scene. Group the information or temporarily hide it.
5	The user should be given constant feedback of the system status, for example when agents change their status.
6	The user should be presented with clearly designed icons which represent states or behaviors. Furthermore, labels and other help should be available to the user which does not recognize said icons, with little interaction needed. This implies that the system will provide icons and labels for the new user, while the advanced user will be able to interact with the interface only by looking at icons. This is meant to speed up the interaction, as recognition is faster than recall.
7	Advanced information should be available for the user if needed, but the priority should be on keeping the interface easy and minimal.
8	The user should be able to see the status of a selected agent.
9	The user should be able to manipulate (e.g. move, select, animate) a controlled entity as he does currently.
10	The user should not be directly prompted with an advanced AI menu directly when he selects an agent. This is because the user may want to handle the entity and not the agent. The opposite behavior may cause unwanted dialogs to appear.
11	When an agent is selected, the user should be able to open an advanced menu for interacting with the AI.
12	The user should be able to open the advanced menu directly when selecting an entity.
13	The user should be able to infer the agent behavior and the current agent state from the advanced menu.
14	The user should be able to stop an active agent or resume a stopped agent from the advanced menu. (AI-Enable/AI-Disable)

15	The user should be able to change the currently selected agent's behavior from the advanced menu.
17	The user should be given a way to identify which entities are connected with the selected agent's reasoning process. For example, there should be a link between a firefighter extinguishing a fire and the fire itself. This link may be represented by a colored highlight or through a dashed line that connects the agent and the entity.
18	The user should be given an emergency-way-out, enabling him to stop an agent which is not behaving as expected.
19	The user should be given a way to "pause" or "halt" the simulation, for training purposes or as an emergency way out. As a consequence, the user should be able to resume a paused game, effectively restoring the game status as it were before the simulation was halted.
20	The user should be presented with imagery that he associates to real world concepts.
21	The user should be able to attach to a non-controlled entity a new agent.

Non-Functional requirements:

1	The interface should be minimal: it should only show needed information and it should not contain clutter.
2	The interface should be easy to navigate: the user should be able to go back to the state where he was in case of a wrong action.
3	The interface should not overlap other in-game interfaces.
4	The interface components should scale and adapt information differently based on the amount of agents currently in the scene.
5	The interface should not obstruct the normal flow of the game interaction.
6	The interface should have animations which serve as a link to what is happening. This is related to giving the user feedback of the status. For example, when changing one agent's current behavior an animation should show that the agent's AI has changed and the agent is now thinking differently.
7	Colors should be used cleverly to differentiate different concepts and to link similar ones.
8	Shaded menus, or differently colored ones should be shown for not-active agents.
9	The interface should show brief animations when an agent's state changes, indicating what is happening.

10	The in-game interface alone should allow the user to see the agent status for each agent, add agents to entities, stop agents, change agents behaviors, and lastly perform emergency way-outs.
11	The interface labels should be clearly readable and scaled.
12	No interface should be displayed when the user is performing particular procedures, such as configuring the students.
13	The interface should “speak” the user language, using common words and following real-world conventions [1].
14	Feedback, statuses, behaviors and messages should be in plain language, and should not contain codes.

7.1.2 Advanced GUI

The requirements here presented will NOT include requirements related to tablets or platform-specific requirements. Requirements for touch-screen interfaces can be found in other studies, while this collection of requirements aims to be as platform-independent as possible.

User Related Functional Requirements (the user is the simulation trainer):

1	The user should be able to see, at a glance, the status of a selected agent.
2	If an entity is selected, the user should be able to see, at a quick glance, the AI properties of the selected entity. For example, if it is a simple entity, ontology properties and/or agents that could operate with the entity should be shown. Otherwise, if it was an entity controlled by an agent, agent information should be shown.
3	The user should be shown complete information about a selected agent, including state, behavior, activation, history, and possibly including a BDI graph/configuration (The BDI graph is left for future studies).
4	The user should be able to activate, deactivate, change the behavior of, delete or add agents from the external GUI, without needing to interact with the game.
5	The user should be able to guess important information at a glance without having to provide input to the external GUI. This requirement is due to the fact that the interaction needed to switch from one interface to another makes it frustrating to have to go back and forth from one to another.
6	If a task needs to be carried out on the advanced GUI, the interaction should be structured in a way that does not involve the game. E.G.: tasks that require the advanced GUI can be carried out without having to go back to the game during the process.
7	The user should be able to manage all the agents or subsets of them. Provide a way to view all the agents or a subset of them, and allow for multiple selection and provide multiple selection options, such as “stop all selected agents”.

8	The user should be able to search the agents, filter them with a pattern, or sort the list.
9	The user should be able to change the display method for multiple items. This is a good usability feature present in other software to display multiple elements. The usual patterns include: list, list with icons, icons, squares.
10	If the engine allows it, the user should be able to see a topographic map of the scene, highlighting agents, indicating where they are and possibly where they are going/what they are doing.
11	The user should be able to access a "stats" panel which includes statistical information of how the system is behaving. For example, how many agents are in the "Panic" state. As a consequence of point 7, it should be possible to perform advanced options on these subsets, such as stopping all the agents in a "Panic" state.
12	The user should be able to quickly return to a "home" state, the state that simply has the interface show information for the in-game selected entity.
13	The user should be provided some degree of entity management inside the advanced GUI. Since entity management is not the focus of this research, no requirements have been defined for entity handling.

Non-Functional Requirements

1	The interface should show the same selected agents as the in-game GUI. This means that agents which are selected in-game are also selected in the advanced GUI. Vice-versa, if the selection is changed form the advanced GUI, the in-game selected agents/entities should change.
2	The interface should show important information at a glance, without needing the user to change the context to interact with the advanced GUI.
3	The interface should follow platform conventions of the platform it is deployed on.
4	The interface should provide enough control over the environment to allow specific tasks to be carried out only using the advanced GUI.
5	The interface should "speak" the user language.
6	The interface should use imagery the user recognizes to evidence concepts.
7	The interface should present text in a clear and human readable way.
8	The interface should be minimal.
9	The advanced interface should show advanced information when the user is selecting a behavior or handling statuses. Advanced information should include a description and possible causes/outcomes of statuses/behaviors.

10	The interface should resemble the in-game interface, using the same imagery, icons and language, the interface should be consistent and follow real world conventions. [1]

These requirements are the result of a series of iterations over a design process. They may result useful for anyone designing an interface to control the AI inside a virtual reality. These specific requirements may be limited in the actions available to perform on an agent, but they can be easily adapted to many other realities.

7.2. Heuristics

Here I will discuss heuristics used to evaluate the interface. These heuristics have been developed during the design process and are mostly adaptions of other studies. See references.

1	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time [1], Moreover, the game should react in a consistent, challenging, and exciting way to the player's actions [2].
2	The system should speak the user's language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order [1].
3	Allow users to tailor frequent actions [1].
4	Minimalist design: The interface should not contain extra information. Every piece of information competes with other pieces of information and diminishes their relative visibility [1].
5	Make effects of the Artificial Intelligence (AI) clearly visible to the player by ensuring they are consistent with the player's reasonable expectations of the AI actor [2].
6	Users should be given controls that are basic enough to learn quickly yet expandable for advanced options [2].
7	The interface should be as non-intrusive to the Player as possible [2].
8	Many people are afraid of granting autonomy to agents because of the fear that the agent will make a bad decision without their consent. The interface should make the user feel comfortable and in a safe environment [3].

8. Conclusions

In this document I analyzed how to structure the interaction between simulation trainers and an artificial intelligence, precisely between simulation trainers and the PRESTO product. The first design was showed, and the process that lead to the final design was explained. The major design choices were explained, notably the need of two separate interfaces. Consequently the final design was presented along with refined requirements and heuristics. The in-game interface was discussed in-depth, showing various processes, such as creating an AI-agent, and the Advanced interface was showed and its main features were presented. Another important result is a series of requirements and a collection of heuristics used to evaluate the interface.

This work may become useful to anyone wanting to interface people with cognitive processes or artificial intelligence, to anyone who wants to present information about how a piece of artificial intelligence is working, and in general to any product that exposes some level of AI-tweaking to the user.

Future work may revolve around providing the simulation trainer a way to interfere with the set of cognitive parameters that AI agents use to make decisions. A deeper study of how to present to the simulation trainer BDI-related information may be necessary, as the algorithm does not "think" as a simulation trainer could expect, and, since this kind of information is complex to show, how to display it in the already complex setting of the serious game may be a crucial aspect. Another aspect not covered is how to show the simulation trainer what entities are affecting agents' actions, pathfinding information, and how to cleverly animate icons and how to graphically design the iconography is left to future studies.

Acknowledgements

Thanks to my supervisors Antonella De Angeli and Paolo Busetta for all the help in writing and revising this document. Furthermore, thanks to the developers and experts from the Delta Labs, Matteo Pedrotti, Mauro Fruet and Jacopo Grandi, for the user studies, the feedback, and the help in writing this document.

References

[1] Jakob Nielsen, 10 Usability heuristics for User Interface Design, Usability Engineering, 1995.

[2]: Using Heuristics to Evaluate the Playability of Games, Heather Desurvire, Martin Caplan, Jozsef A. Toth., CHI 2004, Vienna, Austria.

[3]: Autonomous Interface Agents, Henry Lieberman, Massachusetts Institute of Technology.

[4]: CoJack: aosgrp.com/products/cojack/
AOS: aosgrp.com

[5]: XVR: www.e-semble.com/en/Products/XVR/In_general/
E-Semble: www.e-semble.com/en/home/

[6]: VBS2: products.bisimulations.com/products/vbs2/overview

[7]: The Sims: thesims.com

[8]: DIGuy: www.diguy.com

[9]: Xaitment: www.xaitment.com

[10]: Spir.Ops AI: www.spirops.com

[11]: Thomas W. Malone, Heuristics for designing enjoyable user interfaces: Lessons from computer games, 1982.

[12]: C. Lewis and J. Rieman, Task-Centered User Interface Design: A Practical Introduction, 1993-1994.

[13]: Rao, A. S. & Georgeff, M. P, "BDI agents: From theory to practice", 1995.

[14]: .net: www.microsoft.com/net

[15]: Mono www.mono-project.com/

[16]: Unity3D unity3d.com/

[17]: OWL 2 Web Ontology Language Document Overview, W3C Recommendation 11 December 2012. www.w3.org/TR/owl2-overview/

[18]: Sesame: www.openrdf.org

[19]: Tarja Susi, Mikael Johannesson and Per Backlund, Serious Games – An Overview, 2007