UNIVERSITÀ DEGLI STUDI DI UDINE
DIPARTIMENTO DI MATEMATICA E INFORMATICA
DOTTORATO DI RICERCA IN INFORMATICA

PH.D. THESIS

An architecture supporting the development of serious games for scenario-based training and its application to Advanced Life Support

CANDIDATE:
Alberto Cabas Vidani

SUPERVISOR:
Luca Chittaro

Contents

Contents ..... **iii**

List of figures ..... **v**

Introduction ..... **vii**

1. Architectures and tools for serious game creation **9**
1.1 Difficulties in serious game creation 10
1.2 Solutions proposed by industry 11
1.3 Solutions proposed by academic research 12
1.3.1 Architectures for procedural generation of learning tasks 13
1.3.2 Architectures for serious games based on predetermined learning tasks 16
1.3.3 General Observations on the surveyed solutions 23

2. The proposed Serious Game Architecture for Scenario-based Training **25**
2.1 The proposed architecture 25
2.1.1 The Scenario Simulator module 29
2.1.2 Task repository 40
2.1.3 Menu task hierarchies 46
2.2 Implementation aspects 48

3. Case study: a serious game for Advanced Life Support training **51**
3.1 Conventional training of EMS nurses 51
3.2 Application Overview 53
3.2.1 User interaction 54
3.3 The task repository for EMSAVE 58
3.3.1 Dialogs in EMSAVE 61
3.4 Scenario CTT models in EMSAVE 63
3.5 Patient states 65

4. User Evaluation **67**
4.1 Pilot study 67
4.1.1 EMSAVE prototype 68
4.1.2 Experimental design 69

4.1.3 Participants 73
4.1.4 Results 73
4.2 Evaluation of EMSAVE in ALS retraining 77
4.2.1 Procedure 77
4.2.2 Participants and setting 78
4.2.3 Results 82
Conclusions 89
Bibliography 93

List of figures

Figure 1.1. Example of a learning task as proposed in [26]. In a mini-game, the user has to identify the wrong details on the façade of a famous building. 13
Figure 1.2. A conceptual overview of the architecture proposed by Binsubaih et al. [32]. 14
Figure 1.3. A screenshot of the StoryTec editor produced within the 80Days project. 15
Figure 1.4. A screenshot of the JDoc editor, showing the interface used by senior doctors to choose patients to insert in the virtual world. 16
Figure 1.5. An example of the graphical quality of serious games created through the ED-Game Author. 17
Figure 1.6. A screenshot from a prototypical serious game proposed by McKenzie and McCalla... 18
Figure 1.7. A screenshot from the authoring toolbox AESOP 19
Figure 1.8. A screenshot of the scenario editor proposed by Protopsaltis et al. 20
Figure 1.9. The scenario editor proposed by van Est et al. 21
Figure 1.10. A screenshot from a game built using the e-Adventure platform. 22
Figure 1.11. A screenshot of the dialog authoring tool proposed by Johnson and Valente. 23
Figure 2.1. General overview of the serious game architecture. (Blue boxes represent modules, yellow cylinders represent repositories). 26
Figure 2.2. Example of an XML log file. 28
Figure 2.3. A more detailed view of the Scenario Simulator module. (Blue boxes represent modules, green boxes represent sub-modules, yellow cylinders represent repositories) 29
Figure 2.4. Example of how the Choice operator is used. 32
Figure 2.5. a) An example of the ambiguity problem. b) Eliminating ambiguity by inserting a new node. 33
Figure 2.6. Screenshots of a) user CTT model, b) NPC CTT model and c) concurrent CTT model. 33
Figure 2.7. CTT for the "Conduct Head-to-Toe Exam" procedure. 34
Figure 2.8. An error explanation shown in EMSAVE. 35
Figure 2.9. The XML code of the error model for the CTT example depicted in Figure 2.7. 36
Figure 2.10. Fragment of a state model XML file 38
Figure 2.11. Example of a variable mapping. 40
Figure 2.12 Example of a task definition in the task repository 42
Figure 2.13. An effect of type DialogEffectType. 45
Figure 2.14. Example of an XML file defining a menu. 47
Figure 3.1. Details of the user interface: a) a 3D animated icon indicates the destination of user's avatar movement after a click, b) the contextual menu related to the patient, c)

appearance of the mouse pointer hovering over an interactive object (in this case, a medical device), d) the contextual menu related to the medical device. 55
Figure 3.2. The debriefing window: a red circle denotes a wrong task selection, a green circle denotes a correct one. 56
Figure 3.3. The XML code for the Chest palpation task in the task repository. 57
Figure 3.4. Chart illustrating the execution sequence of effects described in Figure 3.3. 59
Figure 3.5. Yellow flags indicating placeholders in a map. 60
Figure 3.6. Examples of task effects: a) a message box showing the outcome of the Chest Palpation task, b) generic animation of the user's avatar moving the hands on patient's chest, c) video showing how to put the oxygen mask on patient's face, d) a 2D overlay showing the display of a medical device with data taken from the patient state model. 61
Figure 3.7. The effect of type DialogEffectType of the task called Presentation. 62
Figure 3.8. Screenshots of a) the team leader CTT model, b) the teammate CTT model, c) the patient CTT model, d) a fragment of the concurrent CTT model. 64
Figure 3.9. The XML schema definition of the element describing possible outcomes for the chest observation task 65
Figure 4.1. The interface of the EMSAVE prototype used in the first user study. 68
Figure 4.2. Chart showing the average number of correct answers to the pre-test and the post-test. 82
Figure 4.3. Chart showing the average number of correct answers to the pre-test, the post-test and the retention test. 86

Introduction

In the last 10 years, there has been a growing interest towards serious games, computer applications that employ video game technology for educational and instructional purposes. They have been applied and proved effective in many fields, such has military training [15, 34, 68], corporate training [33, 58], education (at all levels, from primary school to college and university) [5, 28, 37], emergency response training [3, 16, 40, 62] and healthcare, both as tools for training healthcare professionals [23, 57] and for treating pathologies or yield changes in health behavior [4, 20].

Serious games bring several benefits. First, as virtual reality applications, they allow users to experience situations that are impossible or hard to reproduce in the real world for reasons of safety, cost or time (e.g., emergency situations). Second, they can support the development of many skills, such as analytical and spatial skills, strategic skills and insight, decision making, social skills and collaboration [58]. Third, the main advantage that distinguishes serious games from virtual reality or other computer applications for training is the fact that they can increase engagement and motivation to learn [49] and thus be more effective in transferring knowledge [11]. Finally, serious games often have low hardware requirements that allow them to be run on common PCs, promoting a more frequent and convenient practice. This is important in many situations where old or low performing computers are utilized for training [15, 58].

Fundamental open problems in serious games research reside in the production process. First, the process is still "handcrafted and labor-intensive" [67] and needs to be streamlined and standardized. Second, to provide the instructional and educational content, involvement of domain experts is necessary, but several communication and cooperation difficulties may arise among game developers and domain experts [28]. A critical help could come from authoring tools tailored to the requirements of domain experts and to the training or learning purposes of the serious games that have to be developed [11, 15]. Moreover, very few disciplined, systematic studies about the effectiveness of serious games exist [58], but they are necessary to identify strengths and weaknesses of these tools, so as to produce more effective serious games and to promote their adoption.

Our research aimed at developing an architecture to support serious games creation, focusing in particular on serious games for scenario-based training. The main purposes of the architecture are:

- to organize and structure knowledge representation,
- to allow for a more efficient game creation process, by making code reuse among different game engines possible and by allowing multidisciplinary teams to create serious games

focusing more on knowledge representation and multimedia resources rather than on software implementation,

- to be applicable to many domains in which scenario-based training is practiced.

To demonstrate the applicability of the proposed architecture, we employed it to develop a serious game (named EMSAVE) for training emergency medical services nurses in advanced life support (ALS) procedures. We then carried out two user evaluations. The first one regarded a prototype of the serious game and aimed at assessing the acceptance of this kind of application by its potential users, its perceived usefulness and the learnability of the adopted control system, as well as to collect suggestions and requirements from the 12 participating nurses. The second evaluation involved a larger sample of 40 healthcare professionals in real training classes and regarded a more advanced version of EMSAVE. Its purpose was to evaluate the effectiveness of the serious game as a training tool for ALS procedures in two ways:

- by comparing subjects' knowledge of ALS procedures before and after training with the serious game, through a questionnaire created by professional ALS instructors,
- by assessing knowledge retention, through a subsequent administration of the same questionnaire, which took place three months after the training session.

This thesis is structured as follows. In chapter 1, we provide a description of the problems hindering serious games creation and a survey of the solutions proposed by the industry and the research community to address them. Chapter 2 describes the architecture we propose to support the creation of serious games for scenario-based training. In chapter 3, we illustrate EMSAVE, the serious game we created to train nurses working for emergency medical services and show how we employed the architecture in its development process. Chapter 4 describes the two evaluations of EMSAVE we carried out with healthcare professionals. Finally, the thesis concludes discussing future directions for our research.

1

Architectures and tools for serious game creation

The term “serious game” started to become popular both in the industry and in the research community in the early 2000s [58]. Many agree that the area of serious games was born with the release of the “America’s Army” game [68] in 2002 ([58, 67]). America’s Army had originally been conceived for promoting military recruitment but was soon adopted also for training new recruits in basic skills as shooting.

There is no widespread consensus on the definition of serious game. In his PhD thesis [66], Yusoff defines a serious game as “a learning tool that incorporates game technology for the purpose of achieving learning objectives rather than pure entertainment”. Zyda proposes a comprehensive formal definition [67] of serious games as “a mental contest, played with a computer in accordance with specific rules, that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives.” To summarize, we agree with the words of Susi et al. [58]: “there seem to be as many definitions available as there are actors involved, but most agree on a core meaning that serious games are (digital) games used for purposes other than mere entertainment”. In one word, the element that distinguishes serious games from entertainment games, as stated by Zyda, is pedagogy, that is, the educational or instructional aspect of a serious game.

1.1 Difficulties in serious game creation

The peculiarities of serious games cause several difficulties that hinder their creation process. First, to be effective in capturing and keeping users' attention, serious games need to be of comparable quality to successful entertainment games. Satisfying such requirements is extremely expensive, development costs of an entertainment game for 2008 have been estimated in the range of $10M - $25M [60]. Moreover, developing games requires the involvement of people with diverse skills such as programmers, 3D modelers, writers and sound designers. Unfortunately, serious games target very small niches (e.g., surgeons, firefighters, emergency nurses, policemen), so possible revenue for a single title would be small, in particular much smaller than the potential revenue of an entertainment game. Thus, development costs for serious games must be kept low, also considering the fact that often the organizations needing them have much less resources to invest for game development than companies producing entertainment games.

Second, development of serious games lacks widely accepted and employed frameworks or architectures which could make the game creation process more efficient, predictable and inexpensive [67]. Although some solutions to this problem have been proposed, serious games are often developed from scratch, since their code is not designed to be reused. The introduction of proper methodologies could facilitate the creation of code modules and resources that can be ported across different projects.

Third, to speed up serious games creation and to reduce costs, developers often resort to game engines (see, for example, [3, 7, 18, 32, 51]). When code is written to work with a specific game engine, porting it to a different engine requires a long work, sometimes not shorter than rewriting it from scratch. Moreover, different serious games often share several functions (at code level) and being able to use the same code across many games would reduce the amount of time and effort needed for its development. However, if code depends on a specific game engine, it can possibly be reused only for serious games based on the same engine. Several reasons could make changing the game engine necessary, for example the need to deploy a game to a different platform (e.g., on a gaming console instead of a personal computer) or the presence of some required features available only in a single game engine. An architecture reducing the dependence of code from the game engine would help mitigate these problems.

Finally, the necessary integration of pedagogical content with the game poses additional challenges. To produce such content, the involvement of domain experts in the design and development process, as well as continuous and effective communication between domain experts and game developers are necessary. Unfortunately, several communication difficulties may arise [28]: domain experts typically do not have the abilities to face game development problems, while game designers and developers are not familiar with the topics the game deals with (i.e., the pedagogical content). The above mentioned game engines often provide editing tools but they are not useful to the purpose of involving domain experts. Some of these tools support editing the virtual world as well as objects and characters in it, but they do not help creating pedagogical

content. Another kind of tools is employed to edit scripts controlling the behavior of objects and characters, interaction with the game and so on, through game engine-specific scripting languages. Scripts have been used to create pedagogical content but they result in engine-dependent and hardly reusable code [6]. Moreover, scripting languages are not understandable by domain experts with no experience in computer programming. Help to overcome these difficulties could come from authoring tools tailored to the requirements of domain experts and to training or learning purposes of serious games [11, 15]. Authoring tools could be employed to acquire pedagogical content from domain experts in a more efficient and structured way than simple interviews and meetings. Such tools could even support easy editing of some aspects of the game directly by the domain experts and automation of some game development activities.

1.2 Solutions proposed by industry

Through the years, the request of serious games for training has grown in many areas, such as healthcare, the military, education, emergency management and the corporate world (see [58] for a survey of serious games application areas). Consequently, the number of companies developing serious games for training has grown as well. These companies rarely provide domain experts with tools to directly edit the pedagogical content for the serious game. Serious game engines that companies have created and use often provide authoring tools but they are rarely meant to be used to create pedagogical content, rather they are meant to edit game resources (such as the virtual world and its content), in the same fashion as the engines for entertainment games. When tools to create pedagogical content are provided, they allow to use this content in predetermined virtual worlds with very limited game aspects. Moreover, details behind serious games developed by these companies are typically not made public. In this section, we survey the platforms proposed by companies to allow the creation of serious games reducing the need for game developers.

Sealund [69] is a company which provides custom serious games, developed on commission, as well as serious game engines that promise to allow the creation of games for training and learning without the help of computer programmers. At a deeper analysis, it turns out that the various engines provided only allow to create slight variants of e-learning applications based on the exchange of questions and answers between the trainee and one or more virtual trainers. Some of these engines allow to develop only 2D games, the ones with 3D features only support the creation of serious games which put the trainee in one out of a limited set of possible virtual worlds where she can only interact with virtual trainers which explain concepts and ask questions. Thus, Sealund's products are very limited in the kinds of serious games that can be developed and in the features offered by those games.

Olive [70] is a software platform to create virtual worlds for training. It provides an SDK with a "wide range of integration and customization features, and flexible, enterprise IT-ready software" (from Olive's website). No detail is given about the possibility for domain experts to create games

on their own, but the descriptions of the platform suggest that the creation tools are conceived for developers.

Bohemia Interactive flagship product, Virtual Battlespace 2 [71], provides a virtual world for military training with a large set of 3D models to be used in it and some editing tools to create missions. A scripting language is provided to create new training scenarios, much like entertainment game engines. Also in this case editing tools are aimed at developers.

Caspian Learning proposes its authoring tool for serious games, called Thinking Worlds [72]. It features visual tools that allow to choose a virtual world among the provided ones as setting for the serious game, place objects and characters in it, determine paths along which they move and so on. The Interaction editor allows to model learning activities. They can be, for example, conversations with virtual characters, multiple choice questions or other kinds of quizzes the user has to solve. The Scene Flow editor allows one to define how scenes, which can be seen as different missions, proceed. The evolution of each scene is defined through charts in which modules representing the possible events (e.g., movement of a virtual character along a path or camera position change) are combined and connected to each other. Logic operators and variables can also be used to determine when an event must be triggered. These tools reduce the amount of programming work needed to develop a serious game, nonetheless they are not built to be used by domain experts, rather by novice programmers. For this reason, Thinking Worlds still provides a scripting language to implement more advanced functions, which the Scene Flow editor does not cover.

In summary, the industry has addressed the problem of providing tools to speed up serious games creation, but the proposed solutions are too limited or still require programmers to create serious games.

1.3 Solutions proposed by academic research

In this section, we survey several architectures proposed by academic research to speed up and streamline serious games creation and to reduce the need for computer programmers during development. We group them in two categories:

- the first one gathers architectures supporting the creation of serious games in which the user has to execute learning tasks that are generated at runtime or chosen at runtime by the game engine among a set of possible ones;
- the second one gathers architectures supporting the creation of serious games in which the user has to execute a (possibly branching) sequence of learning tasks, which was predetermined by instructors.

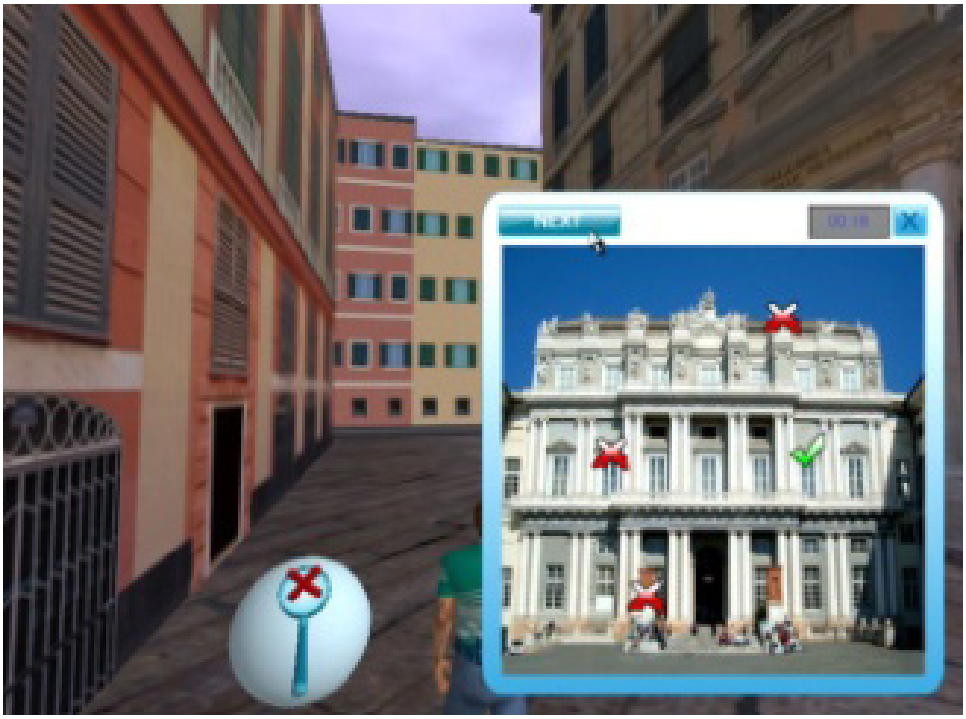


Figure 1.1. Example of a learning task as proposed in [26]. In a mini-game, the user has to identify the wrong details on the façade of a famous building.

1.3.1 Architectures for procedural generation of learning tasks

Bellotti et al. [6] deal with sandbox serious games, which are inspired by sandbox games (such as Grand Theft Auto IV [73]), i.e. games that allow the user to freely explore a virtual world in which several tasks are distributed and associated to specific locations or characters. There does not need to be a predetermined order in which these tasks have to be performed, but often completing a particular task or a set of tasks makes the user proceed in the game (for example, unlocking other tasks). Bellotti et al. use what they call the Experience Engine to determine which tasks to propose to the user, based on the provided user and task models. The user model includes several numeric fields such as skill level, preference for various task types and need for developing particular skills. Task models are composed of fields indicating, for example, task type, task difficulty, dependencies among tasks, coordinates of the virtual world where the task is placed. Values from these models are combined through a genetic algorithm to determine at runtime the sequence of tasks to perform. This differentiates the approach proposed by Bellotti et al. from well-known entertainment sandbox games in which there usually is not a precise task sequence to follow. This architecture has been employed to create serious games promoting the knowledge of the European cultural heritage. These games have the form of treasure hunts inside European cities, where the user has to reach

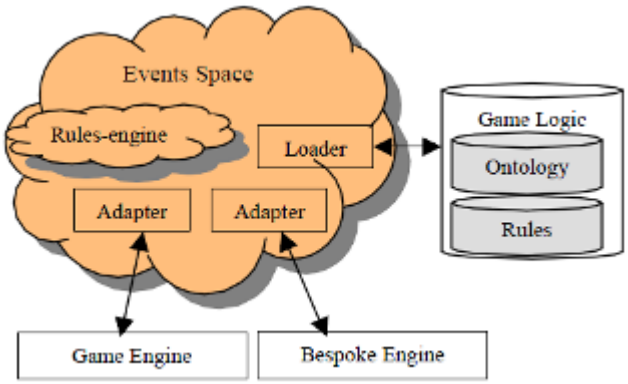


Figure 1.2. A conceptual overview of the architecture proposed by Binsubaih et al. [32].

important landmarks and play mini-games to learn about them (see Figure 1.1 for an example). To exploit the Experience Engine in a serious game, instructors have to specify values for all the fields in user and task models. This probably is the major weakness of the approach, since translating concepts such as user's skill level, user's need of learning specific skills or task difficulty to numbers is not easy and no suggestion is provided on how to do it. Moreover, it can be hard to assure the pedagogical validity of an automatically generated sequence of tasks.

The architecture proposed by Binsubaih et al. [10] employs a rules engine (namely, the JESS engine [74]) to control the game (see Figure 1.2 for an overview of the architecture). It was proposed for games in general and then also applied to serious games [8]. It aims at separating the game logic from the game engine, while often the game logic or at least parts of it are tightly coupled with the game engine, by using its proprietary scripting language. The proposed approach requires game authors to define their own ontologies, describing the virtual world content (e.g., object positions) and domain specific information (e.g., hints for a trainee using the game), and to specify rules to control the game. A Loader module imports ontologies and rules into the rules engine and an Adapter module allows communication between the game engine and the rules engine, by mapping the game engine data structures to the ontology data structures. Thus, the architecture only works with game engines providing scripting languages with data structures which can be mapped to the ontologies. Mapping is achieved by introducing variables that are used as placeholders in the scripting languages and translated at runtime retrieving data from ontologies. The portability of the architecture was tested by creating ontologies and rules for two different games and then making both of them work on two separate engines. The first game aimed at training police officers to deal with traffic accidents. It presented an accident scenario with damaged vehicles, injured people and leaked hazardous material. The user has to investigate the situation to find out how the accident happened. The investigation is carried out by interacting with smart objects in the virtual world, which store information about the accident. The second game

was not a serious game but only a way to demonstrate the performance of the rule engine to drive an NPC running away from the user chasing it.

In [24], hierarchical task network planning is used to procedurally generate scenarios for emergency rescue games. In this case, scenarios concern buildings hit by an earthquake where users have to rescue possible victims and shore damaged walls. Data needed for scenario generation are the initial state of the building and the desired training goals. Given these data, the system can procedurally create a virtual building that can be used for training. As authors underline, the system is strictly limited to the domain of rescue operations in collapsed building, so the training goals correspond to features that the collapsed building can have: wall damage level, possible fires, presence of victims trapped in the building. Moreover, the system proved able of creating scenario descriptions with the desired features, but these descriptions were never been used to create a virtual world for a serious game, so its effectiveness has not been confirmed.

The 80days project [21] follows an adaptive digital storytelling approach to create digital educational games, a term often used when referring to serious games for education. The authors defined an XML based language called ICML (INSCAPE Markup Language, from the name of a previous project) which is used to describe the story behind the game. The story is made of scenes,

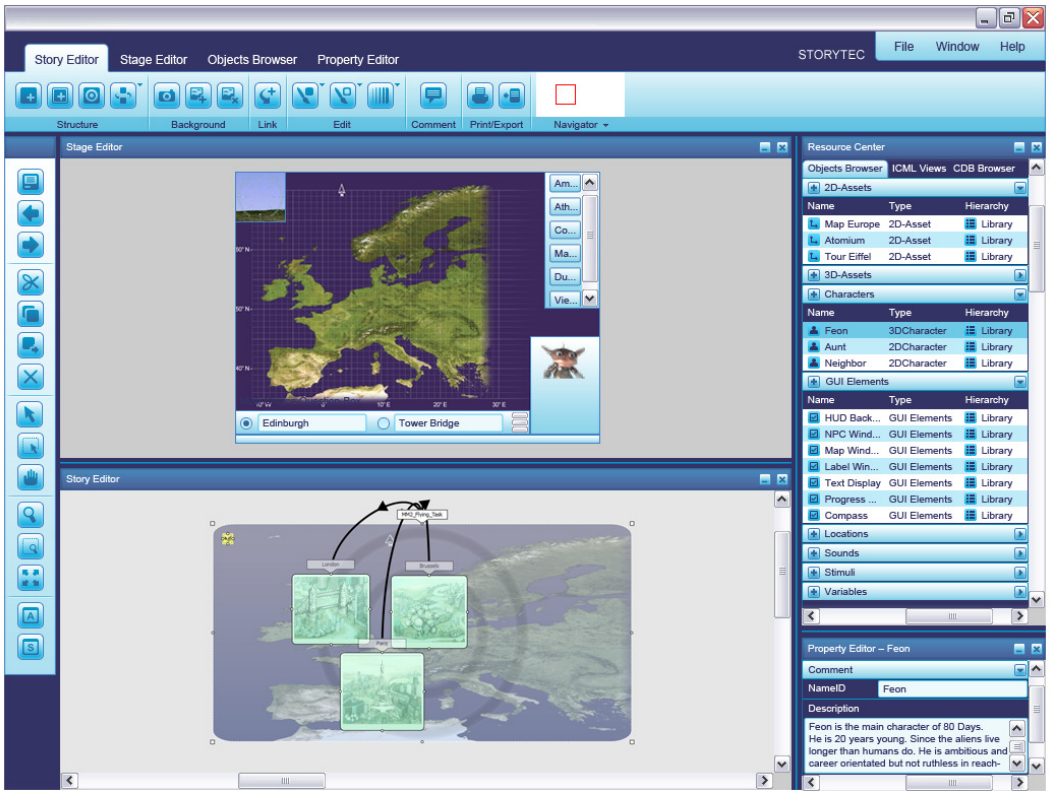


Figure 1.3. A screenshot of the StoryTec editor produced within the 80Days project.

which can be connected to one another, meaning they are consequential, or unconnected. Unconnected scenes are combined with the other scenes by the Story Engine, based on the dramaturgic, learning and gaming aspects of each scene, encoded in its ICML representation. An authoring tool to create stories and produce ICML files through a graphical user interface, called StoryTec [22], has been developed and tested, with encouraging results (Figure 1.3). It seems that this platform is aimed at a specific kind of educational serious game where users learn only through conversations with virtual characters, since no evidence of other kinds of learning tasks is given.

1.3.2 Architectures for serious games based on predetermined learning tasks

Sliney and Murphy propose JDoc [57], a serious game with the purpose of familiarizing junior doctors with the day-to-day stress of a hospital. This system aims at reducing the time senior doctors have to devote to training junior doctors. To allow direct creation of pedagogic content by domain experts (i.e., senior doctors), a Content Management System (CMS) is provided to manage resources like characters, conversations that can occur in the game, user data and so on. A graphical interface allows senior doctors to create scenarios by adding patients (Figure 1.4), together with some related data (e.g. blood test results), nurses and doctors to the hospital virtual world. Users are placed inside a virtual hospital in which they can find and assess various patients. A usability test with junior doctors showed that interaction with the system was easy and that doctors perceived it

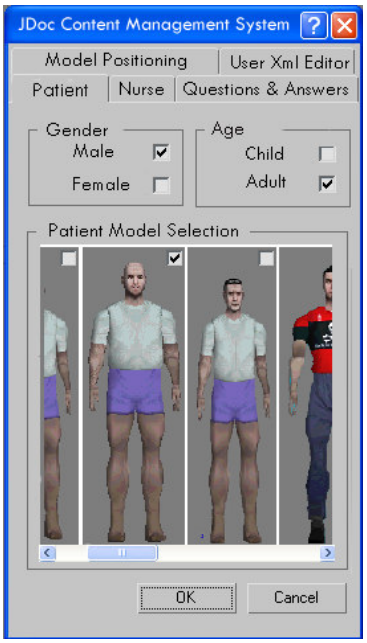


Figure 1.4. A screenshot of the JDoc editor, showing the interface used by senior doctors to choose patients to insert in the virtual world.

Figure 1.5. An example of the graphical quality of serious games created through the ED-Game Author.

as a useful tool for training. Unfortunately, the paper by Sliney and Murphy does not give further details on what information is stored in the CMS and how it is used in the game, neither on how the editing tool for the CMS works.

ED-Game Author is an authoring tool for educational serious games [64]. It provides several virtual worlds and some basic stories to be used for games. Authors can exploit these resources to create serious games in which users can explore a virtual world and encounter characters that pose them questions or give them explanations about selected subjects. Correct answers make the user progress in the game (e.g., earning her a key for a locked door). Special objects hiding hints for the user can also be found. To create a serious game through ED-Game Author, an instructor has to choose an environment as setting and then create the pedagogical content consisting of lessons and tests. The latter can be of several kinds: multiple choice questions, fill-in the blank, true/false questions, open questions. Instructors can also assign possible causes of errors to wrong answers. When a user gives a wrong answer, the error is recorded in her student profile. This way, instructors are able to keep track of user's performance not only by counting errors, but also identifying the most common error causes. The pedagogical content is added through a graphical interface and then associated to locations in the chosen virtual world. During play, when a user reaches a location associated with a test, it pops up and the user has to answer. Other characters in the game can help

Figure 1.6. A screenshot from a prototypical serious game proposed by McKenzie and McCalla.

by giving explanations about a question (if instructors specified so). The biggest limitation of ED-Game Author is its poor exploitation of the advantages that virtual reality can give. The only actions it allows players to perform are to explore the environment and go from a test to another one. Moreover, the graphical quality of the serious games built through this tool is comparable to the one of the 3D videogames of the early 90s (as shown by screenshots in the paper). It is not sure how much it could be appealing to today's students, to whom these serious games are addressed.

McKenzie and McCalla propose an architecture for serious games for professional ethics [33]. It allows to create serious games that train the user in ethical decision making (e.g., to decide whether to omit security testing to complete a critical project on schedule) through conversations with virtual characters and decisions she has to make (as shown in Figure 1.6), by choosing among the presented options. The game is divided in scenes, describing the sequence of conversations and decisions the user has to go through, modeled through properly structured XML files. A learner model, created in collaboration with domain experts, is used to represent the skill level already reached by the user and the training objectives and to change conversations and decisions at runtime, to adapt them to the needs of the user. The architecture is not yet completely implemented, so it has never been tested and no serious game has been created using it.

The authoring toolbox AESOP (Authoring Edutainment Stories for Online Players) [55] was also developed to support creation of serious games exploiting interactive storytelling techniques. In these games, the pedagogical content is implicitly built into the story the user goes through. The creators of AESOP model stories through Finite State Machines (FSMs), in which states correspond to "uninterruptible segments of storytelling" and edges to choices the user can do, leading to other states. Each state describes the virtual characters behaviors (i.e., conversation, gesture or movement) for the corresponding step of the story. A visual tool (Figure 1.7) was developed where FSMs are visualized as graphs and game authors can edit them through a graphical interface and save them in XML format. XML files are processed by a module called Engine/Wrapper, written in

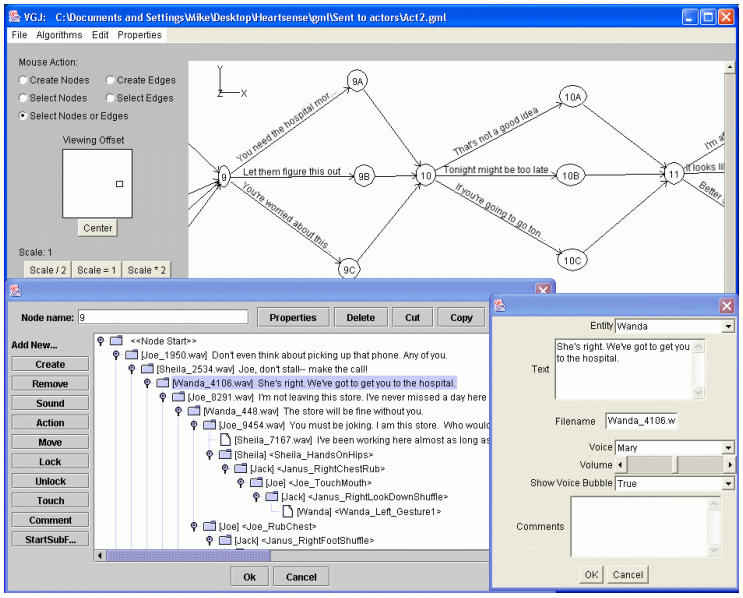


Figure 1.7. A screenshot from the authoring toolbox AESOP.

Macromedia Director's Lingo language. Its main purposes are to translate FSMs in a score using Director syntax and play the story by executing behaviors described in the FSM nodes and showing the user choices modeled by FSM edges. The AESOP toolbox was used to author Heart Sense Game [54], "a role-playing game in which you help the hero try to solve a crime and simultaneously rescue his career and find romance", having the form of a character-driven adventure story in which user interaction and learning tasks are limited to dialogues with NPCs. While authors claim that the use of XML makes their architecture game engine-independent through communication of the Engine/Wrapper module with diverse game engines, it is not specified how it is implemented.

Protopsaltis et al. [50] deal with the problem of scenario-based serious games repurposing, which refers to "the changing of a learning resource, created for a specific educational context, to a new educational context (or contexts)". In their work, authors define scenario-based serious games as "branching scenarios, where users' decisions lead to effects, both immediate and simulation wide, altering the events, characters and situations encountered". Difficulties related to serious games repurposing are similar to the ones related to their development. The authors propose to use properly structured XML files to store scenarios. These files describe the events that happen in a scenario and the consequences of user actions. They also provide a graphical editor, called mEditor (Figure 1.8), to make it easier for non-programmers to edit scenarios, by rendering XML files as tree structures and allowing drag-and-drop operations on them. The editor has been used to create scenarios for an educational game, called Happy Night Club, that explains to teenagers the dangers of binge drinking. The game has a 2D interface and the user can learn only through conversations with virtual characters. Authors do not specify to what extent their approach can be applied to other

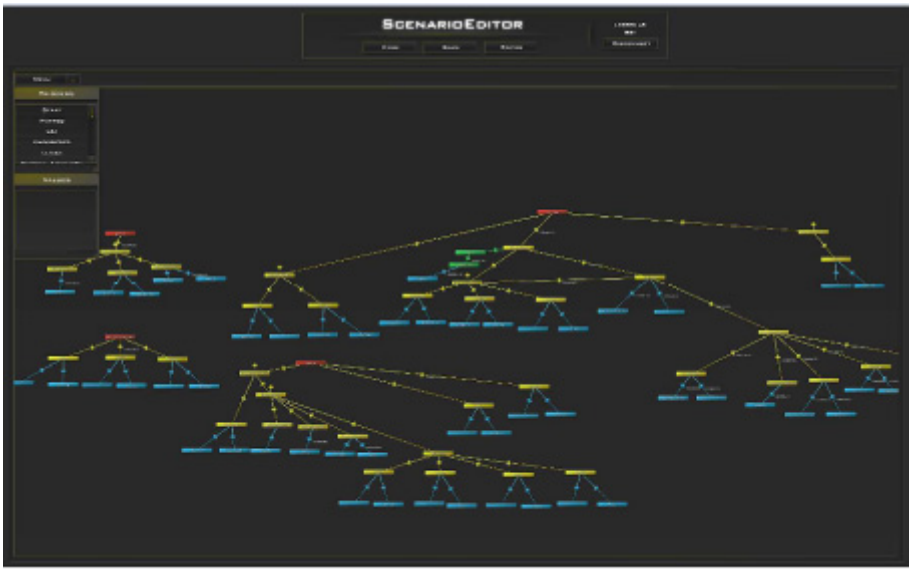


Figure 1.8. A screenshot of the scenario editor proposed by Protopsaltis et al.

kinds of games (for example, 3D first-person games) and teaching techniques, involving interactions beyond simple conversations.

Van Est et al. [17] propose an authoring tool to build scenarios by composing building blocks (Figure 1.9). The basic ones are actions and events: the former are performed by the player, the latter by the game. To allow non-linear scenarios, logic blocks are employed, that allow scenario branching, such as if-then blocks. Finally variables, combined with logic blocks, allow a finer grained control over non-linear scenarios (e.g., a variable describing the result of an action can be used as threshold to decide whether an event must be performed). Authors also developed a scenario editor in which blocks are represented by nodes and can be composed through a drag-and-drop interface. A scenario is visualized as a branching sequence of connected nodes of various kinds. The scenario editor is also used to execute scenarios, while an additional application, the Communicator, is used to implement communication with the game engine through TCP/IP. This approach allows instructors to create scenarios that can be potentially executed by any game engine. However, scenario editing is not made completely game engine independent by this solution, for two reasons. First, actions and events are specific to a single game and must be defined by game developers to exploit the features of the adopted game engine. Second, modules performing communication with the Communicator application must be implemented according to actions and events available in the scenario editor. The editor was used to create scenarios for a game called Supervisor, aimed at training supervisors of oil drilling sites. In the game, the user learns to handle hazardous situations, watch personnel and take care of health, safety and environment regulations.

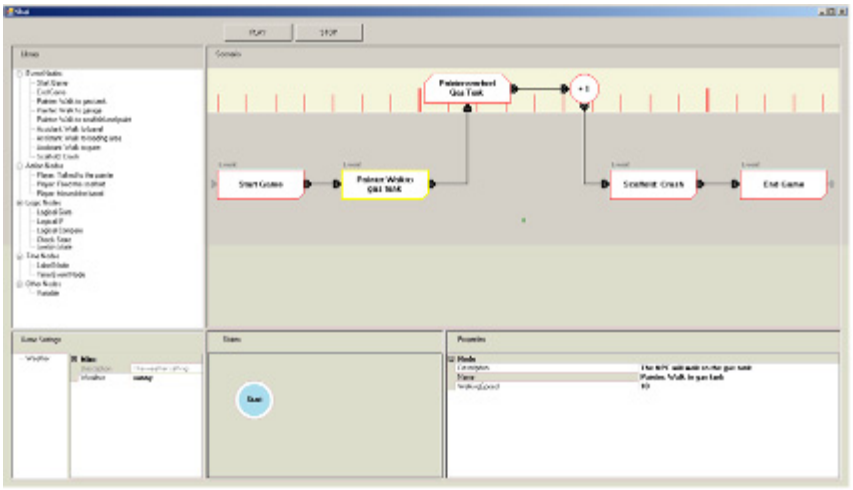


Figure 1.9. The scenario editor proposed by van Est et al.

<e-Adventure> [60] is a platform for the creation of serious games resembling the point-and-click graphic adventures that were very popular from the mid ‘80s to the mid ‘90s, such as the Monkey Island series [75]. To make the pedagogical content game engine-independent, the creators of the platform follow what they define a “documental approach” [38], exploiting XML files to describe scenarios. Since XML files are human readable, this approach also facilitates the collaboration of programmers with domain experts while working on pedagogical content. In <e-adventure>, the story is divided in scenes, each of which requires the user to carry out some tasks to be able to move to the next scene. Tasks usually consist in performing some action on objects on the screen (e.g., clicking an object and selecting an action from the related menu) or having a conversation with a virtual character. Positive or negative scores can be assigned to tasks [36], so as to create an automatic assessment of user’s performance. The platform also provides an editing tool supporting the creation of scenes, by combining the art assets provided by artists and specifying clickable objects with related actions. <e-Adventure> was exploited, for example, to create a serious game teaching the Insertion Procedure of Central Venous Catheters [35]. In this game, the user is shown pictures of the actual medical equipment necessary for the procedure and has to perform the correct sequence of actions on the different tools by clicking on them and selecting the appropriate actions from a menu. Communication with the patient is also carried out when specified by the scenario. As the authors themselves underline, the biggest weakness of this platform is the game genre to which it is limited, that is, 2D graphic adventures. Entertainment games of the last decade offer much more realistic graphical quality and interactions to users, which could not feel engaged by 2D graphic adventures (see Figure 1.10 as an example), and thus less motivated to learn. Moreover, van Est [17] claims that the customization options provided by the <e-Adventure> editor are very limited and that authoring tools require operations which are overwhelming to a domain expert with no game development experience.

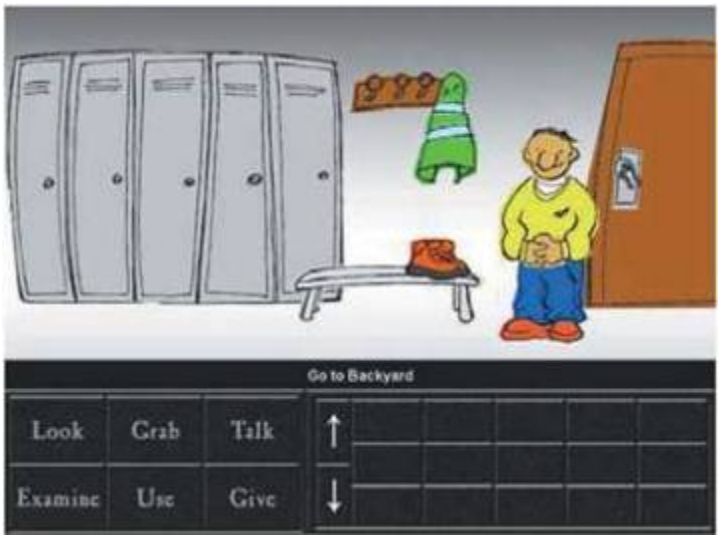


Figure 1.10. A screenshot from a game built using the e-Adventure platform.

The work presented by Johnson and Valente [27] shows how effective authoring tools can be, by allowing instructors to create content on their own with limited or no help by computer programmers. The paper describes the architecture and the authoring tools for the platform called Tactical Language and Culture Training System (TLCTS), which was exploited to build 3D serious games to teach the military foreign languages and cultures and how to use them for tactical purposes. Besides the overall architecture, the provided authoring tools are probably the major strength of this platform (Figure 1.11 shows an example of the dialog authoring tool). They allow domain experts to create pedagogical content through graphical interfaces, mainly in the form of conversations and non-verbal communication. Code for single lessons is automatically generated by the authoring tools, so the need for computer programmers is greatly reduced. Domain experts are also given the possibility to choose art assets (e.g., 3D models) to compose the virtual world in which lessons take place. Serious games produced using this platform have also been deployed and used by the military and the authoring tools proved really effective in rapidly increasing the available training material. One of these games is Tactical Iraqi, aimed at teaching the Iraqi language to American soldiers in order to be able to communicate with the population in Iraq. In the game, the user can explore a virtual world reproducing typical situations that the military can experience in Iraq, meet NPCs representing Iraqi civilians and talk with them. The game also exploits a speech recognition engine and a scoring system to assess user's performance. The authors of the paper, however does not suggest if and how this platform can be extended to serious games of different genres or with different purposes.

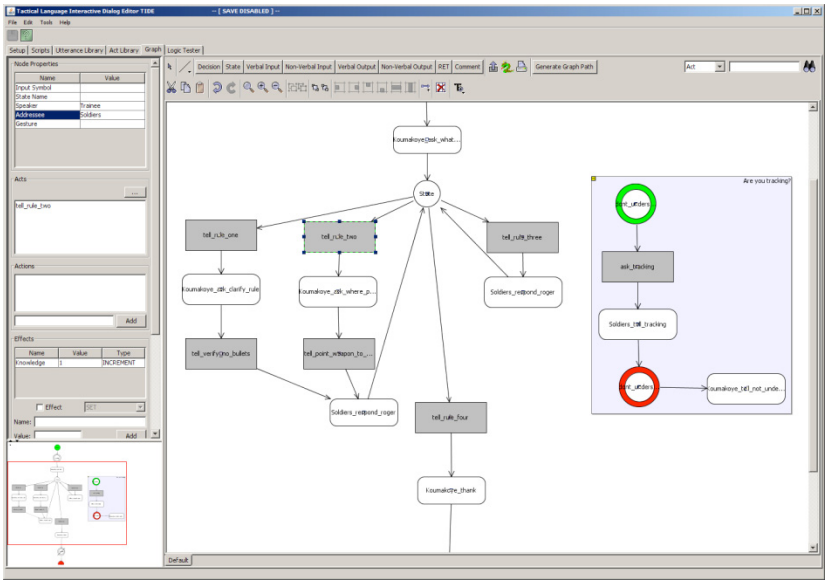


Figure 1.11. A screenshot of the dialog authoring tool proposed by Johnson and Valente.

1.3.3 General Observations on the surveyed solutions

As can be seen, there are many differences among the proposals surveyed above: some of them are suited only for specific domains (e.g., emergency rescue or military training), some others are limited to only a certain game genre (e.g., 2D graphical adventures) or to a single teaching technique (e.g. multiple answer questions and textual explanations). These limitations suggest that the development of a single tool supporting the creation of any kind of serious game in any domain is really hard, if not impossible.

However, some common aspects can be identified. First, most authors resort to the XML language, or a language derived from it, to model pedagogical content. The reasons for this choice are that it is a standard language, which can also be processed by computers while being also understandable by humans.

Second, most of the proposed architectures support the creation of serious games based on scenarios intended as narratives describing a possibly branching sequence of tasks that the user has to perform. Many apply this approach to education, but there are also examples of applications to other areas. For example, Moreno-Ger et al. employ it for clinical procedures training [35], while McKenzie and McCalla employ it for ethical decision-making in corporation environments [33].

Finally, as regards authoring tools, only Johnson and Valente [27] present a successful case study. The other attempts at creating authoring tools that domain experts can use are still in an early stage of development or have proved insufficient. The main aspects of the surveyed architectures are summarized in Table 1.1.

Authors	Game genre	GEI	Authoring tools	Types earning tasks	Learning tasks representation
Bellotti et al. [1]	Sandbox games	No	No	Puzzle-games	Annotated task tree
Binsubaih et al. [2]	Potentially anyone	Yes	No	Potentially anyone	Rule-based
Hullett and Mateas [5]	Not specified	No	No	Rescue people in collapsed building	SHOP statements
Göbel et al. [4]	Digital storytelling	No	StoryTec	Dialogs	Story objects encoded in ICML (XML-based)
Sliney and Murphy [10]	First person 3D adventure	No	CMS	Dialogs and observations	Not specified
Virvou et al. [12]	First person 3D adventure	No	ED-Game Author	Dialogs	Not specified
McKenzie and McCalla [7]	Digital storytelling	No	No	Dialogs	Scenes with possible choices for user (XML)
Silverman et al. [9]	Interactive drama	Yes	AESOP	Dialogs	FSM, edges as dialog choices (XML)
Protopsaltis et al. [8]	2D graphic adventure	Yes	mEditor	Mainly dialogs	Trees with if-then-else nodes connecting events (XML)
Van Est et al. [3]	Many	Yes	Shai	Not specified	Sequences of actions connected by logic blocks
Torrente et al. [11]	2D graphic adventure	No	Yes	Dialogs and object selections	FSM, branches corresponding to user choices (XML)
Johnson and Valente [6]	First person 3D adventure	No	Yes	Mainly dialogs	Not specified

Table 1.1. Main aspects of the surveyed architectures. The acronym GEI in the header of the third column stands for game engine independent. In the last column we specify in parentheses if XML is used to store learning task representations.

2

The proposed Serious Game Architecture for Scenario-based Training

Addressing the issues with serious games development introduced in chapter 1, we propose an architecture for serious games which has the following main purposes:

- to organize and structure knowledge representation,
- to allow for a more efficient game creation process, by making code reuse among different game engines possible and by allowing multidisciplinary teams to create serious games focusing more on knowledge representation and multimedia resources rather than on software implementation,
- to be applicable to many domains in which scenario-based training is practiced, which include healthcare [2, 41, 52], emergency management [1, 53] and police corps [31].

2.1 The proposed architecture

We first introduce some key concepts on which our work is based:

- The *scenario* concept is central to our architecture. We define it as a narrative describing the correct sequence of *tasks* that should be carried out by the user and the events that should take place in the game, as well as locations, objects and characters involved.
- *Game entities* are all objects and characters in the virtual world that can have a *state* evolving during the scenario.
- A state is a set of properties relevant to the domain and depending on the instructional requirements. For example, in a medical serious game a patient state could include vital signs and several other symptoms, while in a serious game for firefighter training the state of a game entity could describe whether that object is on fire. Property values can change as a scenario proceeds. In the example above, a game entity could catch fire at a certain point of the training session and then fire could be extinguished by the user.

To make the code reusable and to make the architecture platform-independent and game engine-independent we were inspired by the document-oriented approach proposed in [38]. We exploited the XML format to store and organize the information describing scenarios (i.e., the pedagogical content) and to model some other application features, such as the structure of menus (further details in the following sections). This file format has the advantage of being processable by computer applications and also facilitating editing by game developers and domain experts [38]. XML files can be directly edited through any text editor and their hierarchical structure makes them

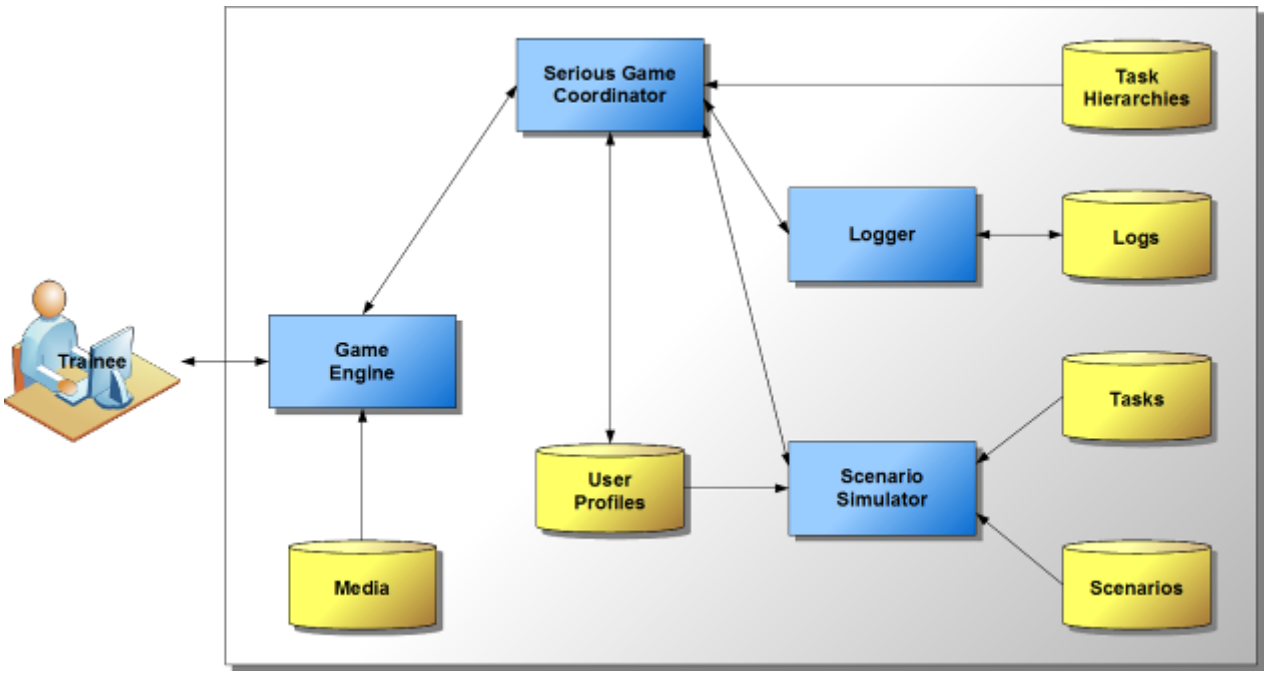


Figure 2.1. General overview of the serious game architecture. (Blue boxes represent modules, yellow cylinders represent repositories).

easier to understand than programming languages to domain experts without experience in software development. Moreover, it's relatively easy for developers to provide authoring tools to domain experts for creating and editing XML files. This authoring tools would expose the content of XML files in a more user-friendly way and allow editing them through a graphical user interface (e.g., as proposed by Torrente et al. [61]). Once available, this tools would make the creation of pedagogical content even more convenient and developer-independent.

An overview of the modules and repositories that compose our architecture is provided in Figure 2.1. The architecture allows to create serious games for scenario-based training, in which players can explore a virtual world and select tasks to perform on game entities. Correct selections make them proceed through the scenario. The game engine (represented in Figure 2.1 by the homonymous module) is responsible for simulating and rendering the virtual world, managing user interaction (both with the virtual world and with the graphical user interface) and giving feedback to the user (e.g., by playing audio samples or showing text messages). We designed the architecture such that any game engine can be plugged into the corresponding module by implementing only small changes to the code. This is possible thanks to the Serious Game Coordinator module, which acts as an interface between the Game Engine module and the rest of the architecture. After selecting the game engine on which to execute the serious game, developers only need to implement communication between the engine and the Serious Game Coordinator. Then, this module will coordinate the other ones and handle communication between them and the game engine. Communication is partly implemented through events, as proposed also by BinSubaih et al. [9] or by van Est et al. [17]. To receive messages the game engine must subscribe to events produced by the Serious Game Coordinator, while to send messages the game engine must call public methods the Serious Game Controller makes available.

The Logger module records information about user interaction and exploration of the 3D world, produced by the game engine, and outcomes of tasks, produced by the Scenario Simulator, in appropriately structured XML files (kept in the log repository, labeled Logs in Figure 2.1), following a schema we defined. These data can be used to analyze user performance during or after training sessions, for trainees' auto assessment and to report progress during training sessions. In more detail, the Logger module records the following information:

- names of tasks selected by the user and whether the selection was correct,
- 3D coordinates of the destination of every user movement,
- the target of each mouse click (e.g., an interface component or a 3D object),
- the coordinates of each waypoint that is reached by the user's avatar, which are special points in the 3D world that the user's avatar gets to before performing specific tasks.

A timestamp is assigned to each recorded event, so that their sequence can be reconstructed.

Figure 2.2 shows an example of a log file. At the beginning, the name of the player is recorded followed by the list of events, in the PlayerName element. The first stored information is the timestamp of the event, in the homonymous element. Then, for each event the values of several parameters are reported. Parameter names depend on the type of the event, specified in the initial GameEvent element. For example, a TaskSelectionGameEventType element stores the target on

```
<EMSAVELog xmlns:xsi="http:"//www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <PlayerName>Andrea</PlayerName>
   <GameEventList>
      <GameEvent xsi:type="WaypointGameEventType">
         <TimeStamp>2010-07-14T16:27:27.5944095+02:00</TimeStamp>
         <X>-12.61424</X>
         <Y>2.255441</Y>
         <Z>1.49274135</Z>
         <Vertical>0</Vertical>
         <Horizontal>-1.0978539</Horizontal>
      </GameEvent>
      <GameEvent xsi:type="TaskSelectionGameEventType">
         <TimeStamp>2010-07-14T16:27:30.2635621+02:00</TimeStamp>
         <Object>Patient</Object>
         <KeywordPath>Presentazione a paziente</KeywordPath>
         <Outcome>Right</Outcome>
      </GameEvent>
      <GameEvent xsi:type="WaypointGameEventType">
         <TimeStamp>2010-07-14T16:27:33.1197255+02:00</TimeStamp>
         <X>-12.61424</X>
         <Y>2.255441</Y>
         <Z>1.49274135</Z>
         <Vertical>-0.473972619</Vertical>
         <Horizontal>-1.12364757</Horizontal>
      </GameEvent>
      <GameEvent xsi:type="TaskSelectionGameEventType">
         <TimeStamp>2010-07-14T16:27:35.0528361+02:00</TimeStamp>
         <Object>Patient</Object>
         <KeywordPath>Spoglia torace</KeywordPath>
         <Outcome>Right</Outcome>
      </GameEvent>
      <GameEvent xsi:type="MouseClickGameEventType">
         <TimeStamp>2010-07-14T16:27:39.3120797+02:00</TimeStamp>
         <MouseTarget>MenuWindow</MouseTarget>
         <TargetElement>Chiudi simulatore</TargetElement>
      </GameEvent>
   </GameEventList>
   </EMSAVELog>
```

Figure 2.2. Example of an XML log file.

which the task was performed, the name of the task and its outcome, that is whether the selection was right or wrong.

In the following, we describe in detail the Scenario Simulator module and the various repositories depicted in Figure 2.1.

2.1.1 The Scenario Simulator module

Real world scenario-based training is usually carried out in form of a simulation, with one or more instructors directing it and role-playing actors impersonating the different characters involved. These simulations are based on scenarios, which are written by instructors and used to direct training sessions and evaluate trainees' performance. Trainees usually have to execute a sequence of tasks to correctly handle the situation described by the scenario and thus successfully complete the training session.

Serious games can be employed to transfer real world training simulations to a virtual world with synthetic characters, but to make it possible an appropriate and effective way to model scenarios is necessary. A formal representation of scenarios, together with visual tools for editing them, would allow domain experts to create training scenarios that could directly be incorporated in games. They could then be exploited to control the game as well as evaluate user performance by comparing her task selections against the correct sequence of tasks described in the chosen scenario.

The Scenario Simulator module (Figure 2.3) receives user's task selections from the Serious Game Coordinator and exploits scenario models in the scenario repository (labeled Scenarios in

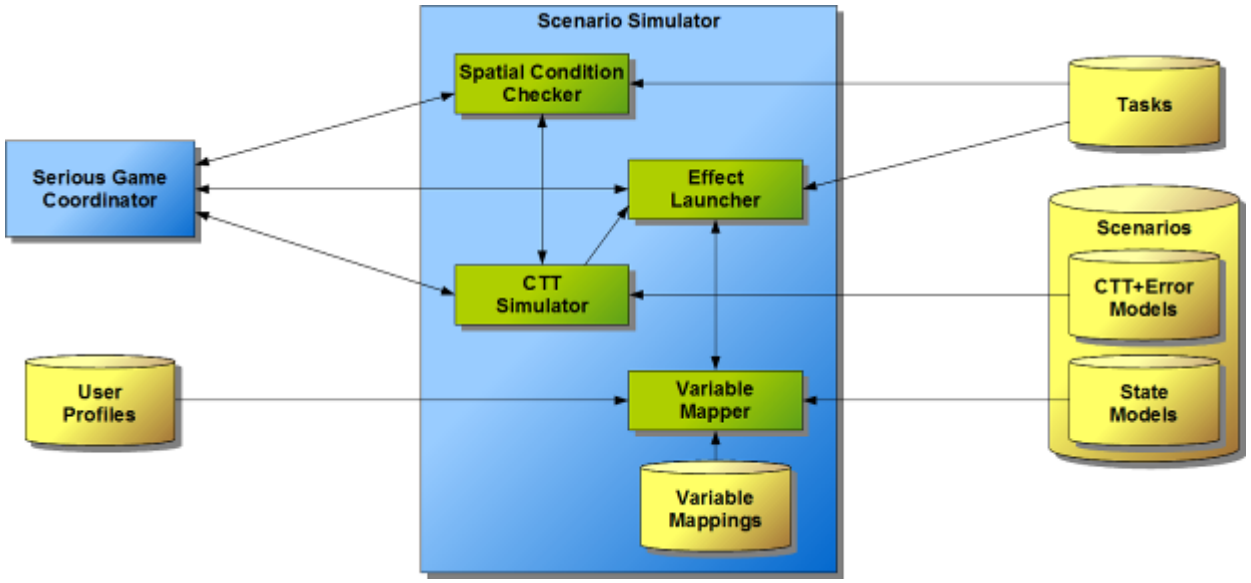


Figure 2.3. A more detailed view of the Scenario Simulator module. (Blue boxes represent modules, green boxes represent sub-modules, yellow cylinders represent repositories)

Figure 2.3) to determine whether those selections are correct. Then it checks whether the spatial conditions for the selected task are satisfied, through the Spatial Condition Checker module. If they are not, the Scenario Simulator module tells the Serious Game Coordinator to do the necessary actions to satisfy them or to ask the user to do them (depending on what specified in the spatial conditions descriptions, as explained in section 2.1.2.1). When spatial conditions are satisfied, the Effect Launcher module tells to the Serious Game Coordinator what effects must be executed (effects definition is described in section 2.1.2.2). Then, based on the CTT model of the scenario (described in section 2.1.1.1), the CTT Simulator module determines which tasks are correct selections for the user based on the last task completion.

In the following sections, we illustrate how modules composing the Scenario Simulator and the repositories depicted on the right side of Figure 2.3 are defined and used.

2.1.1.1 Scenario representation

The main models exploited by the Scenario Simulator are the ones representing the sequences of tasks that the user and the game entities have to perform.

To this purpose, we exploited task models, which are used in human-computer interaction (HCI) to analyze the logical activities that support the achievement of users' goals [45]. They can be exploited for different purposes such as system design, development and usability evaluation. They describe the tasks needed to achieve a particular goal or a set of goals. These tasks are typically represented in a hierarchical structure. Task descriptions can range from abstract activities to concrete actions.

In a serious game for scenario-based training, players have to perform a correct sequence of tasks that leads to a specific goal, and there could be various sequences that lead to the same result. We propose to use task models to describe and structure these tasks. Moreover, task models can be represented by a proper data structure, to be processed by the game engine or game authoring tools. Task models can thus support game production in different ways:

- When the design process requires to define a scenario or multiple scenarios (which might illustrate, for example, single stages of the game), each scenario could be formalized through a task model. Then, task models could help identify requirements of the game and thus support design decisions.
- Game building tools could exploit the task modeling formalism to directly involve people with no particular skills in game development (e.g. experts of a specific domain) in the game creation process. For example, the tools could provide the expert with a graphical user interface to model instructional content in terms of task models.
- At runtime, the task model could be employed by the game engine to support and monitor procedure execution. For example, it could be used to present the player with the actions she can perform, retrieving them from the data structure, based on the history of actions already taken.

- A task model representation of scenarios can support replay and evaluation of players' performance. Having a structure telling what sequences of tasks should be carried out to successfully complete the game, it is possible to compare them with players' actions and to report errors. This feature is particularly useful in serious games, because it allows assessment of the learning progress.

In this section, we introduce the task modeling formalism we adopted and illustrate how we employed it in our architecture.

ConcurTaskTrees

To model scenarios, we adopted the ConcurTaskTrees formalism (hereinafter, CTT), a formal notation originally proposed by Paternò et al [46] in the domain of interface design and development. It has been used to design and develop hypermedia [44] and safety critical applications [47], to support usability evaluation [30], and to analyze user interactions with web applications [42].

The main features of CTT are:

- hierarchical structure - tasks are arranged in a tree structure, useful for decomposing the problem into smaller parts;
- graphical syntax - the hierarchical structure is represented through a graph which reflects the logical structure of the hierarchy;
- temporal operators - tasks are linked with their siblings, which are tasks grouped under the same node (called parent), through temporal operators describing temporal relations among them;
- concurrency modeling - concurrent models can be used to describe temporal relations among tasks belonging to different models.

A CTT model is built in three phases:

1. tasks are decomposed and arranged in one or more hierarchical tree-like structures;
2. temporal relations among tasks are identified;
3. objects are associated with each task, whose execution involves interaction with one or more objects.

Different temporal operators allow users of CTT to create complex temporal dependencies among tasks. Temporal operators defined in the notation are:

- Order independence (T1 |=| T2). Tasks T1 and T2 can be performed in any relative order;
- Concurrency (T1 ||| T2). Tasks T1 and T2 are performed at the same time. They do not need to have the same duration, so they can overlap;
- Synchronization (T1 |[]| T2). Tasks T1 and T2 have to synchronize on some exchange of information;
- Enabling (T1 >> T2). Termination of T1 enables execution of T2;
- Choice (T1 [] T2). T1 and T2 are both enabled, but when one of the two is chosen, the other one is disabled;

- Enabling with information passing (T1 []>> T2). When T1 terminates it provides some value for T2 and enables it;
- Deactivation (T1 [> T2). T1 is interrupted by T2; when T2 terminates, then T1 is reactivated;
- Iteration (T1*). T1 is repeated until another task deactivates it;
- Finite iteration T1(n). T1 will be performed n times;
- Optional task [T1]. Performance of T1 is not mandatory.

Besides the hierarchical modeling and graphical syntax, which are common features of many task modeling formalisms, there are two peculiar features of CTT that make it particularly suitable for our purposes.

First, CTT allows one to model concurrent processes, a feature not easily found in other task modeling formalisms [45]. This can be useful in serious games design, because it allows a serious game to concurrently manage and control tasks performed by one or more user's avatars and by all the game entities involved in a scenario. Moreover, interactions among characters and game entities can be modeled. For example, it is possible to model a procedure on which several users have to collaborate or to simulate consequences of user tasks on game entities in the virtual world, such as state changes.

Second, the CTTE (ConcurTaskTrees Environment) application [39] supports a relatively easy and quick visual creation and editing of CTT models. It also allows to export CTT models into XML format. This makes the exploitation of task models by other applications very convenient: a developer can easily create a parser for the CTT XML syntax that extracts the needed information from the exported model.

The application of CTT in our architecture

To model scenarios, we exploit the concurrent features of CTT, combining several models:

- the *user CTT model* describes the correct sequence of tasks that the user has to perform,
- for each game entity performing tasks in the scenario, one *game entity CTT model* describes sequences of tasks the game entity has to perform in the virtual world,
- a *concurrent CTT model* is used to specify temporal relations among tasks belonging to the

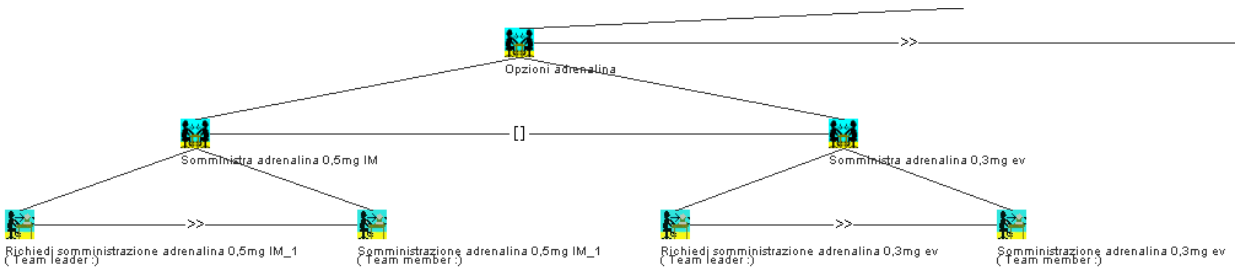


Figure 2.4. Example of how the Choice operator is used.

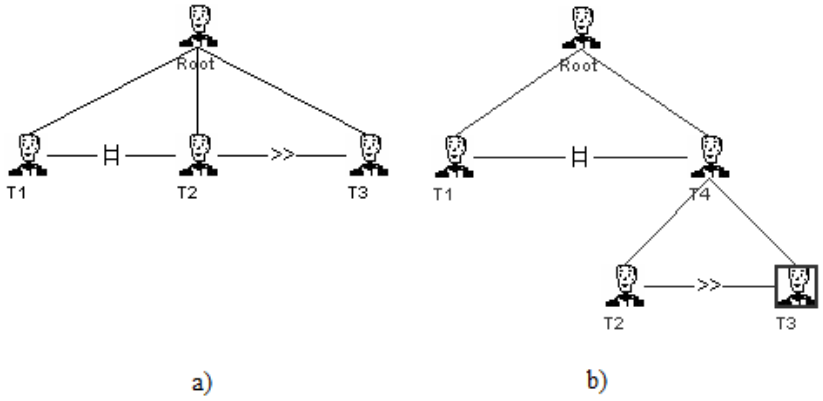


Figure 2.5. a) An example of the ambiguity problem. b) Eliminating ambiguity by inserting a new node.

other CTT models.

Leaf nodes in the CTT models represent tasks the user or game entities can perform, while internal nodes serve the purpose of grouping their child nodes. Generally, game creators are free to group tasks to their preference and build trees with any number of levels. However, there are special cases in which grouping becomes mandatory, to ensure the correct simulation of the sequence of tasks:

- when the Choice temporal operator (introduced in section 2.1.1.1) is used, all possible alternative nodes must be connected in sequence via the Choice temporal operator and then grouped under the same parent (this holds also for alternative nodes which are roots of subtrees). For example, in the model depicted in Figure 2.4, there are two nodes connected by a Choice temporal operator ([]). When a leaf child of one of them is executed, the children of the other one will not be selectable, and thus executable, anymore;

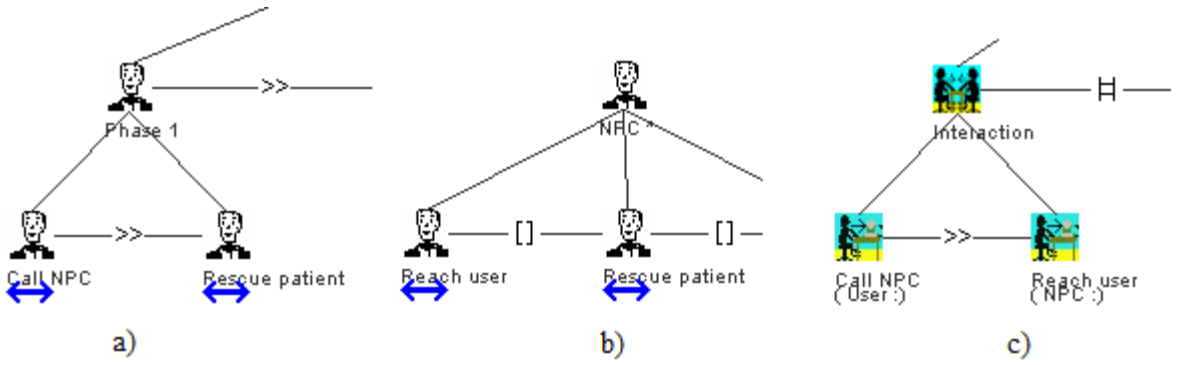


Figure 2.6. Screenshots of a) user CTT model, b) NPC CTT model and c) concurrent CTT model.

- when the Order Independence temporal operator is used in a subtree, if in the same subtree other different temporal operators are present, the subsequent nodes which are not connected through the Order Independence temporal operator must be grouped under a new node, which then has to replace the first one of the sequence in the tree. Figure 2.5a shows an example of a tree in which an Order Independence temporal operator is followed by a different temporal operator. Figure 2.5b shows how the previous tree must be transformed by grouping nodes to ensure correct simulation.

The interpretation of temporal operators by the Scenario Simulator depends on the model. In the following, we describe how different models are interpreted by the Scenario Simulator.

The user CTT model describes which tasks can be performed by the user at each moment of the simulation. For example, if she has to call an NPC (Non-Player Character) before trying to rescue an injured patient, the two tasks have to be connected by an "Enabling" temporal operator in the user CTT model. Based on this relation, the Scenario Simulator could, for example, prevent the user from (or give a negative score for) rescuing the person before having called for help

Temporal operators in the game entity CTT models, instead, are used by the Scenario Simulator to tell the game engine what tasks game entities have to perform and to describe temporal relations among them. The Scenario Simulator uses this information, together with the concurrent CTT model, to control the actions performed in the virtual world by game entities. The concurrent CTT model is used by the Scenario Simulator to know temporal relations among user's and game entities tasks. All leaf tasks executed by game entities and the user (i.e., tasks included in game entities and user CTT models) must be inserted in the concurrent model and connected to their siblings through the appropriate temporal operator. For example, to tell that a certain game entity task must be executed after a certain user task, the game entity task must be the right sibling of the user's task and an "Enabling" temporal operator must connect them. Figure 2.6 exemplifies this case. The user CTT model fragment in Figure 2.6a depicts an hypothetical "phase 1" of a rescue procedure

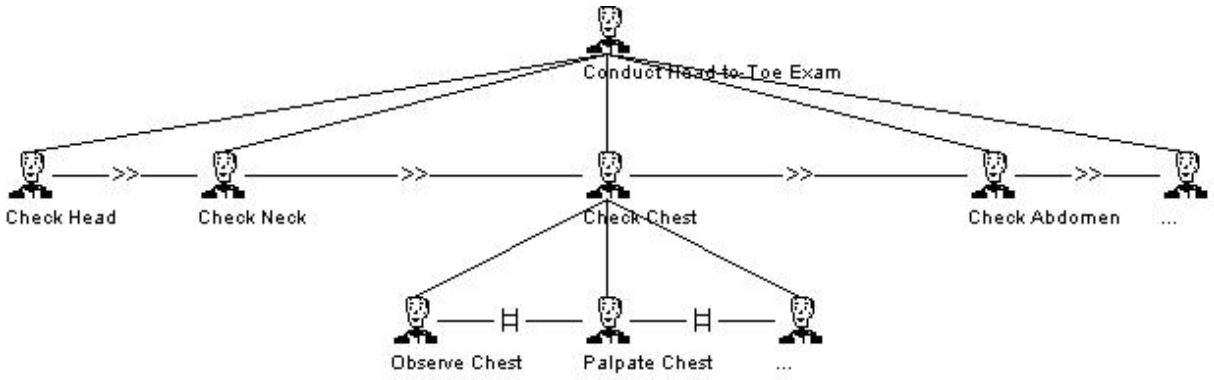


Figure 2.7. CTT for the "Conduct Head-to-Toe Exam" procedure.

(internal nodes can be named arbitrarily and serve only to the purposes of giving a logical structure to their child nodes or to ensure correctness of the simulation). Figure 2.6b, instead, shows a fragment of the NPC CTT model, with a root node, labeled “NPC”, which is repeatable, as shown by the ‘*’ symbol close to the label. Since the root is repeatable, every task in the tree is always repeatable. Moreover, each task can be executed at any time, since they are all connected to each other by Order Independence temporal operators. The concurrent CTT model in Figure 2.6c puts the tasks from the two previous models together. The “Call NPC” task in this model is a reference to the homonymous task in the user CTT model, while the “Reach user” task is a reference to the homonymous task in the NPC CTT model. From this concurrent CTT model, the Scenario Simulator knows that, after the “Call NPC” task is executed, the NPC has to perform the “Reach user” task. As a consequence, the Scenario Simulator tells the Serious Game Coordinator to execute the effects of that task (e.g., to make the NPC game entity move towards user’s avatar), which will then communicate it to the game engine.

Error models

As described in [43], information in task trees can be exploited to support user's interaction with different kinds of automatic help. More precisely, it can be used to inform users about (i) tasks that can be executed at a particular moment of the simulation (i.e., tasks that satisfy temporal constraints described in CTT models) (ii) subtasks they should perform to complete a task, and (iii) correct sequences of tasks that enable a particular task. For example, if we consider the procedure “Conduct

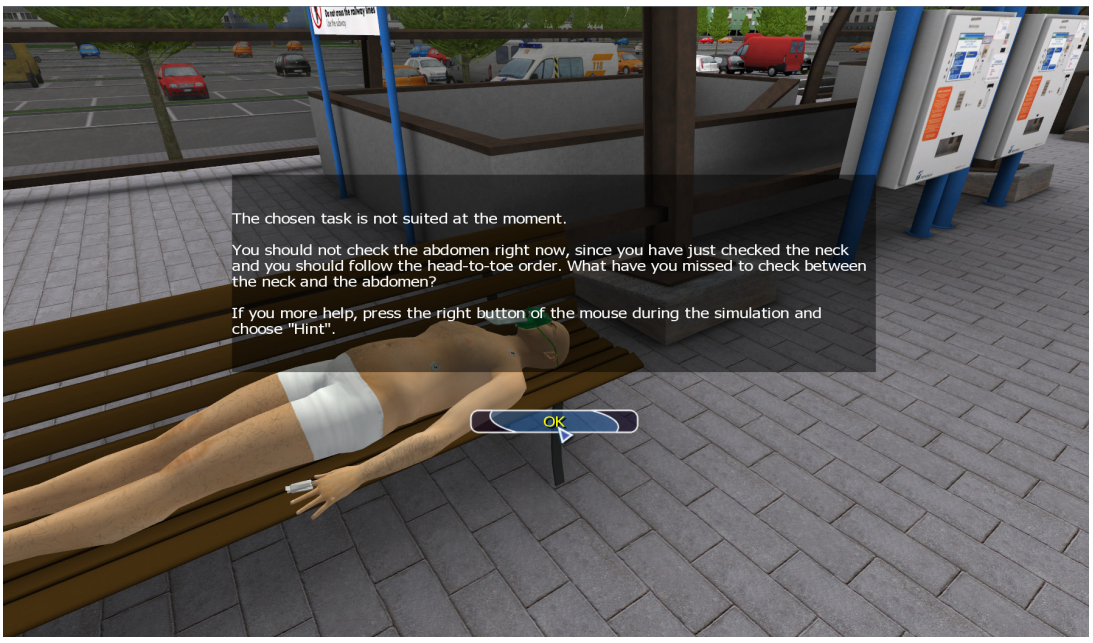


Figure 2.8. An error explanation shown in EMSAVE.

Head-to-Toe Exam" in Figure 2.7, the system can tell the user that, at the beginning, only the task "Check Head" can be executed. After completing it, the system can tell that "Check Neck" can be executed, and then all the subtasks of "Check Chest" will be executable at the same time. To complete "Check Chest", the system will suggest to complete all its subtasks, while any permutation of those subtasks would be suggested as a correct sequence to enable the "Check Abdomen" task after the completion of "Check Neck".

These kinds of help can be useful for trainees to know how they could advance in the simulation, but, unfortunately, they are not enough to make users understand their errors when they choose an unsuited task. More precisely, they can provide reasons why a chosen task is not suited, but only in terms of tasks to complete first, if any. Moreover, if a task is not executable at a particular moment of the simulation and will never be, a training system based only on CTT would be able to tell trainees that the chosen task is unsuited since then on, but not to explain why. To overcome such limitation and provide trainees with more tutoring feedback about wrong choices, we propose to couple CTT models with specifically designed error models (as shown in Figure 2.3, in the CTT+Error models repository). In intelligent tutoring systems, error models are employed to model users' misconceptions [12], such as typical errors that users commit during training. In scenario-based training, typical errors can be the omission of a task in the application of a procedure or the execution of a task that is common-sense, but scientifically wrong (e.g., people usually try to keep standing or sit down a fainting person, who instead should be helped to lie down to easily supply blood to the brain). To provide tutoring feedback to users who commit typical errors, our architecture allows trainers to associate each node of the user CTT model with a typical error list, i.e, a list of unsuited tasks that inexpert trainees typically perform after completing that node. For each task in the typical error list, trainers can write an explanation about why it is unsuited and should not be performed at that moment. In this way, if a trainee chooses an unsuited task in the typical error list, the Scenario Simulator module can not only determine that it is unsuited, but also retrieve the more informative explanation. For example, considering again the procedure "Conduct Head-to-Toe Exam" in Figure 2.7, if novice users typically perform "Check Abdomen"

```
<ErrorModel>
   <TaskErrors>
      <!--Errors for tasks preceding "Check Neck" should be insert here-->
      <TaskError>
         <TaskIdentifier>Check Neck</TaskIdentifier>
         <DefaultExpl>You have just checked the neck[…]</DefaultExpl>
         <TypicalErrorList>
            <TypicalError>
               <ErrorTaskName>CheckAbdomen</ErrorTaskName>
               <SpecificExpl>You should not check[…]</SpecificExpl>
            </TypicalError>
<!--Other explanations for typical errors after "Check neck", explanations for the following
tasks and closing tags should be here-->
```

Figure 2.9. The XML code of the error model for the CTT example depicted in Figure 2.7.

immediately after “Check Neck”, the task “Check Abdomen” could be inserted in the typical error list for “Check Neck” with an explanation like the one shown in Figure 2.8.

Of course, enumerating all possible trainees' errors may be very challenging and time consuming for trainers, so we allowed them to specify also more general explanations that will be shown for all less typical errors. More precisely, each node of the user CTT model can be associated to a description that sums up the state of the simulation after completing that node and provides some advice on how to continue. The association is defined by developers at design time, using the TaskIdentifier element (shown in Figure 2.9) to specify the name of a task to which a typical error list is associated. If, at runtime, the user selects an unsuited task and it is in the typical error list of the last executed task, the specific explanation is provided. Otherwise, the more general hint is provided instead. For example, if after completing the task “Check Neck” of the procedure “Conduct Head-to-Toe Exam” in Figure 2.7, the user selects a task that is not a subtask of “Check Chest” and it is not in “Check Neck” typical error list, the system will use the general explanation associated to “Check Neck”, which can sound like “You have just checked the neck. Complete the head-to-toe exam in the right order by conducting the required exams on the chest”. The XML code for the error model (see Figure 2.9 for an example) has an entry (TaskError) for each task in the user CTT model. Besides the TaskIdentifier element mentioned above, each entry contains a mandatory default explanation (DefaultExpl) and an optional list of specific explanations (SpecificExpl), one for each typical error (referred to with the corresponding task name).

In general, the explanations provided by trainers should not offer the explicit solutions, but should be intended as hints to make users reason about possible choices. Users who could not guess how to proceed, even after the hint, can always rely on the explicit help (i.e., the list of suitable tasks) that the SG can automatically offer by exploring the CTT.

2.1.1.2 State models

As can be seen in Figure 2.3, the scenario repository not only stores CTT models and error models, but also state models. There must be a state model for each game entity with state. A state model describes how values of properties in a state change as the scenario proceeds and is stored in a properly structured XML document. The information that must be included in state models varies across different domains and also across different serious games in the same domain. For example, for serious games in the domain of medical personnel training, the information that must be included in the patient state may vary depending on the purpose of the serious game. For instance, in games for triage training (e.g., the ones proposed in [26, 29, 32, 63]) only symptoms relevant to the triage procedure need to be represented, such as wounds, blunts and conscious state. Instead, in serious games for training healthcare professionals working inside the hospital, the set of vital signs and symptoms to be represented would be much larger, since in the hospital patients are attached to several devices that continuously monitor numerous parameters.

For these reasons we defined an outline that game developers must use to define state models for a specific domain or a specific serious game. An example of a state model we used for a patient

```
<StateModel>
   <EntityName>Patient</EntityName>
   <StaticData>
      <PersonalData>
         …
      </PersonalData>
   </StaticData>
   <States>
      <State name="State 1">
         <VitalSigns>
             <CardiacFrequency>65</CardiacFrequency>
            …
         </VitalSigns>
      </State>
      <State name="State 2">
         <VitalSigns>
             <CardiacFrequency>143</CardiacFrequency>
            …
         </VitalSigns>
      </State>
   </States>
</StateModel>
```

Figure 2.10. Fragment of a state model XML file

in a medical serious game is shown in Figure 2.10. Every state model file must begin with a StateModel element, which contains an EntityName, a StaticData and a States element, which in turn includes one or more State elements having a string attribute describing their name chosen by developers. These are the only mandatory elements in state model files, elements nested inside them can be customized by developers. The EntityName element reports the name of the game entity to which the state model is associated. So, to correctly employ state models, a name must be assigned to the corresponding game entity (as allowed by any game engine) and the same name must be specified inside the EntityName element. The StaticData element must contain data that is common to all the states and does not change as the game proceeds. In the example, we included in this element the personal data (e.g., gender). State elements must be used instead to store data changing as the game proceeds. Each State element must contain the same XML elements, structured in the same way, the only thing that can change is the value contained in each element. As the scenario proceeds, game entity state changes are described by changes in values in the corresponding state models. In the example above, we nested in the State element an element describing the vital signs of the patient. It contains several other elements, such as the one reporting cardiac frequency, which changes from “State 1” to “State 2”.

To create a state model for a specific domain, developers must work together with domain experts to identify the data it must include and how to structure it. Then, the XML schema of the state model we provide must be extended, in order for the architecture to be able to understand the

state model and to correctly interpret value data types. Data stored in the state model can be accessed through variables, which are mapped by the Variable Mapper module, described in section 2.1.1.3. Through this approach, the architecture can be applied to any domain without modifying the code, but only customizing XML document schemas.

2.1.1.3 The Variable Mapper sub-module

To access the information stored in state models and in the user model at runtime, game creators can use variables, whose name must start with the at sign (@). These variables can be seen as placeholders that the Variable Mapper sub-module substitutes at runtime, by retrieving data corresponding to variables from the state models.

Mappings between variables and state models elements are defined in properly structured XML files of which you can see an example in Figure 2.11. The root element is a VariableMappings element, which can contain one or more VariableMapping elements. Each mapping associates a variable to an XML path. The Variable element contains the string corresponding to the variable name, while the XMLPath element can contain a UserModelPath element or a StateModelPath. The former is used when the variable must be substituted with a value from the user model and contains a string describing the XML path starting at the root of the document. An XML path is a sequence of elements name nested within each other, divided by the slash symbol (/), which bring to the element whose value must be substituted to the variable. The StateModelPath is more complex and contains other elements:

- the GameEntityName element stores the name of the game entity with which the state model is associated,
- the PathType element can either contain the “Static” or “Dynamic” string, respectively specifying whether the path refers to static data stored in the state model or to data stored in the States section of the model,
- the Path element contains the relative XML path of the value with which the variable must be substituted. If PathType is Static the path is relative to the StaticData element in the state model, otherwise it is relative to the State element corresponding to the current state.

In Figure 2.11, for example, the variable called "CardiacFrequency" is mapped to the CardiacFrequency element inside the VitalSigns element of the patient state shown in Figure 2.10.

When developers define a new state model, they also have to define an XML file with variable mappings. This way, variables can be used to refer to data stored in the state model, without having to remember XML paths. Moreover, as described above, variables can be used to retrieve data that changes as the scenario proceeds: the same variable can refer to a different State element inside a state model, depending on the moment in which its value is retrieved.

2.1.2 Task repository

CTT models describe temporal and hierarchical information about tasks. The task repository (labeled Tasks in Figure 2.3) stores additional details for each task appearing in a CTT model (i.e., each leaf in the tree).

The task repository is written in XML format. We defined an XML schema which determines how the information has to be laid out, so as to create a standard way to describe tasks. Thanks to the schema, serious games creators are able to define their own task repository tailored to the domain of interest, so that the Scenario Simulator can employ it. The task repository includes definitions for both tasks that can be performed by the user and by the NPCs.

For each task, the task repository stores information about (for an example, see Figure 2.12):

- its *name*, which must be unique, since it is used as key when searching for a task,
- its *performer*, i.e., the virtual character or game entity which performs the task (e.g., the user or an NPC),
- its *target* (optional), i.e., the game entity on which the task is performed,
- *spatial conditions* (optional), i.e., conditions that must be satisfied to allow task performance,
- *effects* (optional), i.e., consequences of the task on states, virtual world and the serious game graphical interface (explained in more detail in section 2.1.2.2).

While name, performer and target are string values that need no further explanation, in the

```
<VariableMappings>
   <VariableMapping>
      <Variable>CardiacFrequency</Variable>
      <XMLPath>
         <StateModelPath>
            <GameEntityName>Patient</GameEntityName>
            <PathType>Dynamic</PathType>
            <Path>VitalSigns/CardiacFrequency</Path>
         </StateModelPath>
      </XMLPath>
   </VariableMapping>
</VariableMappings>
```

Figure 2.11. Example of a variable mapping.

following we explain in more detail spatial conditions and effects.

Figure 2.12 shows a fragment of an XML file defining a task for a medical serious game called "Chest palpation", as can be seen in the TaskName element. The performer is specified in the RoleLabel element, which reports the label used to identify the performer in the game. Similarly, the target of the task is specified in the TargetLabel element. Conditions are described by single Condition elements nested inside a Conditions element. Condition elements are explained in detail in section 2.1.2.1. Similarly, several Effect elements describing the effects of the task are nested inside an Effects element, as explained in section 2.1.2.2. Each Task element can contain only one Conditions and one Effects element, but they both can contain one or more Condition or Effect elements.

```
<Task>
   <TaskName>Chest palpation</TaskName>
   <RoleLabel>TeamLeader</RoleLabel>
   <TargetLabel>Patient</TargetLabel>
   <Conditions>
      <Condition xsi:type="PositionConditionType">
         <Solver>System</Solver>
         <PositionLabel>GenericAnimationOnPatientTorso</PositionLabel>
         <ThresholdDistance>0</ThresholdDistance>
      </Condition>
      <Condition xsi:type="OrientationConditionType">
         <Solver>System</Solver>
         <OrientationLabel>GenericAnimationOnPatientTorso</OrientationLabel>
         <ThresholdRotation>0</ThresholdRotation>
      </Condition>
   </Conditions>
   <Effects>
      <Effect  xsi:type="SerialEffectsType">
         <Effect xsi:type="ParallelEffectsType">
            <Effect xsi:type="ChangeCameraEffectType">
               <UserStoppable>1</UserStoppable>
               <CameraLabel>AtPatientChest</CameraLabel>
            </Effect>
            <Effect xsi:type="SerialEffectsType">
               <Effect xsi:type="AnimationEffectType">
                  <Duration>4</Duration>
                  <GameEntityLabel>TeamLeader</GameEntityLabel>
                  <AnimationName>GenericActionChest</AnimationName>
                  <Loop>true</Loop>
                  <Speed>1</Speed>
               </Effect>
               <Effect xsi:type="TextMessageEffectType">
                  <UserStoppable>1</UserStoppable>
                  <Message>@ChestPalpation</Message>
               </Effect>
            </Effect>
         </Effect>
         <Effect  xsi:type="PositionEffectType">
            <UserStoppable>true</UserStoppable>
            <GameEntityLabel>TeamLeader</GameEntityLabel>
            <PositionLabel>TeamLeaderSafe</PositionLabel>
         </Effect>
         <Effect xsi:type="OrientationEffectType">
            <Duration>0</Duration>
            <GameEntityLabel>TeamLeader</GameEntityLabel>
            <OrientationLabel>TeamLeaderSafe</OrientationLabel>
         </Effect>
      </Effect>
   </Effects>
</Task>
```

Figure 2.12 Example of a task definition in the task repository

2.1.2.1 Task spatial conditions

The creators of a serious game could require that some spatial conditions are satisfied before the effects of a task are executed. These are the only conditions which are not controlled by the scenario, since the user's avatar and the other game entities can change position and orientation in the game world during the training session. However, to guarantee consistent results, it is often required that a game entity has a specific position and orientation. For example, when the user's avatar and an NPC have a dialog, it could be required that their corresponding 3D models face each other. These three types of spatial conditions can be specified in task descriptions:

- distance between performer and target (e.g., during a dialog the performer and the target 3D models cannot be too far from each other),
- presence of the target in the performer field of view (e.g., before a dialog the performer 3D model should look at the 3D model of the target to speak with),
- position and orientation of the performer in the 3D world (e.g., the game engine could need the performer 3D model to be in a precise position and/or have a precise orientation before performing an animation on an object).

Figure 2.12 shows how conditions are defined. The condition type is specified inside the Condition element. The following Solver element specifies whether the spatial condition must be satisfied by the user or by the system. When a spatial condition has to be satisfied by the system, the Scenario Simulator module asks the game engine, through the Serious Game Coordinator, to move and/or rotate the task performer in order to satisfy the conditions. Similarly, when a spatial condition has to be satisfied by the user, the Scenario Simulator module asks the game engine, through the Serious Game Coordinator, to show a message suggesting that she needs to move closer to a certain point and/or to rotate towards a certain object. This distinction is useful since, in some cases, trainers want the user to carry out these actions, for didactic purposes, i.e., to make the trainee remember to satisfy that condition also in the real world. In other cases, this is not necessary and the spatial condition has to be met only to make the scenario progress. For example, in a medical serious game teaching how to deal with deaf patients, the instructor could need to teach that when speaking to a deaf person, it is necessary to be in her field of view. Explicitly asking users to do it in the serious game would reinforce learning. On the other hand, sometimes 3D models must execute animations which involve interaction with other 3D models. To avoid incoherent behaviors, such as unwanted compenetrations between 3D models, the serious game creators could require a game entity to be in a precise position and have a precise rotation before executing the animation. This condition can be satisfied by the system, with no intervention by the user, since in this case there is nothing to learn.

After the Solver element, there are elements peculiar of the specific type of condition. Positions and orientations are specified not using 3D coordinates, but labels that refer to the names of placeholders inserted by game creators in the virtual world. This allows the reuse of task descriptions in different virtual worlds, since placeholders with the same labels can be used by simply changing their position and orientation coordinates.

2.1.2.2 Task effects

Task effects are defined using the Effect elements that can be seen in Figure 2.12. Each effect has a type among the following:

- ChangeStateEffectType triggers state changes (e.g., a heart rate variation in a virtual patient could be the effect of a defibrillation task performed by the user),
- VideoEffectType triggers playback of videos (e.g., videos showing details that cannot be reproduced by the virtual world at a sufficient level of fidelity) and audio (e.g., virtual patients' respiratory sounds),
- TextMessageEffectType triggers the display of text messages (e.g., outcomes of medical tests),
- PictureEffectType triggers the display of pictures (e.g., of a body part affected by a pathology),
- ChangeControlVisibilityEffectType triggers the visualization of specific parts of the graphical interface of the serious game (e.g., a window showing vital signs of a virtual patient),
- AnimationEffectType triggers playback of animations (e.g., performed by a virtual character),
- ChangeCameraEffectType triggers changes in virtual camera parameters (e.g., position and orientation of the camera),
- ChangeGameEntityVisibilityEffectType triggers game entity visibility,
- PositionEffectType and OrientationEffectType respectively trigger changes in position and orientation of 3D objects,
- DialogEffectType triggers playback of dialogs between virtual characters.

When a task has multiple effects they must be nested under an Effect element of type SerialEffectType or ParallelEffectType (as happens in Figure 2.12). These elements do not only serve the purpose to group effects but also to define their execution order:

- effects nested under an Effect element of type SerialEffectType are fired in sequence, that is, each effect can start only after the previous one has completed (the order in which they are listed in the XML document determines the order in which they are executed),
- effects nested under an Effect element of type ParallelEffectType are all fired at the same time.

Effect elements of type SerialEffectType can be nested under Effect elements of type ParallelEffectType, so that serious game creators are free to define any effect sequence they need.

There are two elements that can be used in each Effect element, except for the ones of type SerialEffectType and ParallelEffectType:

- the UserStoppable element contains a Boolean value which must be true (or 1) if user input is required to complete the effect and false (or 0) otherwise,

- the Duration element contains a value of type double describing the time it takes to the effect to complete.

An example of an effect requiring user input to complete is the effect of type TextMessageEffectType in Figure 2.12: to leave the user freedom to read the message at her own pace, the UserStoppable element for that effect must be set to 1 and the serious game could show, for example, a button to close the window showing the message which will signal the game that the user has given the input to complete the effect. After these two common elements, there are elements peculiar to each effect type, i.e., the effect parameters.

The most complex effect type is the DialogueEffectType, exemplified in Figure 2.13. In our architecture, dialogs are sequences of lines alternatively said by the participating virtual characters, which can be at most two. The first interlocutor is the game entity performing the task or the user, the second interlocutor is the task target. Each dialog can contain one or more LinePair elements, grouped under a LinePairs element. Each of these elements must contain an Interlocutor1Line element, representing the line said by the first interlocutor, while the Interlocutor2Line, representing the line said by the second interlocutor, is optional. This way, effects of this type can also be used to make a character say a single line, e.g., an exclamation. The text of each line is stored in a Text element, which is mandatory, and one or more optional audio fragments can be associated to each line. This approach allows game creators to compose messages reusing audio samples.

Effects of a task and the order in which they have to be executed are defined when a task description is added to the task repository and can't be changed at runtime. However, depending on the effect type, some effect parameter values can be instantiated at runtime. To do so, game creators

```
<Effect  xsi:type="DialogEffectType">
    <LinePairs>
          <Linepair
                <Interlocutor1Line>
                      <Text>Buongiorno,  sono  @UserGenderJob  del  118,  cosa  sta
                            succedendo?
                      </Text>
                      <Audio>
                            <AudioFragment>@PresentationQuestion</AudioFragment>
                      </Audio>
                   </Interlocutor1Line>
                <Interlocutor2Line>
                      <Text>@Presentation</Text>
                      <Audio>
                            <AudioFragment>@PresentationAnswer</AudioFragment>
                      </Audio>
                   </Interlocutor2Line>
             </Linepair
    </Linepairs>
</Effect>
```

Figure 2.13. An effect of type DialogEffectType.

can use variables, which are then substituted at runtime with the appropriate values by the Variable Mapper module, as described in section 2.1.1.3. A variable is used for example in the effect of type TextMessageEffectType shown in Figure 2.12. This effect includes an Element named Message whose value is the message that will be shown to the user. In this example, the message to be shown is the result of a chest palpation in a medical serious game. At runtime, when this effect is executed, the Scenario Simulator finds the @ChestPalpation variable and uses the Variable Mapper to discover that it must be substituted with the text describing the result of the examination stored in the current state of the patient. The message string can then be provided to the game engine, through the Serious Game Coordinator, to be shown to the user.

2.1.3 Menu task hierarchies

In model-based user interface design, models are used to give a representation of tasks that can be performed through an interface, of the abstract structure and behavior of the interface and of its visual parts [48]. Model-based user interfaces bring several advantages [59], among which reusability and support for the creation of design tools, which can easily understand models. These are also purposes of the architecture we propose, so we applied some of the principles of model-based user interface design by coupling task descriptions, illustrated in section 2.1.2, with an XML based hierarchical description of the user interface.

As introduced at the beginning of this chapter, to proceed in the scenario, the user has to select the correct task among the available ones. Every game entity can be defined as target for a task. A specific menu must be associated to each task target, which will contain all the tasks the user can perform on that target. While the implementation of the interaction (e.g., mouse button click management and menus appearance) depends on the chosen game engine and thus cannot be generalized, we created an XML schema to define task hierarchies that support the creation of game engine-independent menus.

Menus based on our schema are organized following the well-known hierarchical structure in which each menu can contain several items grouped in nested submenus. We decided to adopt this structure for the menus for two main reasons:

- many well-known office applications make use of hierarchical menus, so many users, not only computer experts, would be familiar with similarly structured menus,
- grouping tasks reduces their number in each submenu, to avoid the user the inconvenience of scrolling down very long lists of tasks.

Each menu is described by a single XML file, of which an example can be seen in Figure 2.14. At the top, each file has to include a Language and a Country element specifying the language of the current menu, to allow menu localization. Then, a Target element indicates the target of the tasks included in the menu, which must be a string the game engine can use to retrieve the corresponding game entity. For example, for the Menu in Figure 2.14 to work, the virtual world must include a game entity to which the string “Relative” is associated (how this is implemented

depends on the specific game engine).The main element called Menu can include two kinds of elements:

- Menu elements identifying submenus,
- Action elements corresponding to tasks the user can perform.

Each Menu element must be followed by a MenuName and a Description element. Action elements must include a TaskName and a MenuItemName element:

- the former indicates the name of the task in the task repository to which the menu item corresponds,
- the latter indicates the name of the task shown to the user.

The depth of this recursive tree structure is not limited, that is, each menu can contain an

```
<ActionHierarchy xmlns="http://tempuri.org/ActionHierarchy.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <Language>it</Language>
   <Country>IT</Country>
   <Target>Relative</Target>
   <Menu>
      <MenuName>Tasks on character</MenuName>
      <Description>Menu showing tasks on a character</Description>
      <Menu>
         <MenuName>Communicate</MenuName>
         <Description>Communicate with the character</Description>
         <Action>
            <TaskName>Presentation to the character</TaskName>
            <MenuItemName>Presentation</MenuItemName>
         </Action>
         <Menu>
            <MenuName>Request information</MenuName>
            <Description>Request information from the caracter</Description>
            <Action>
               <TaskName>Ask character name</TaskName>
               <MenuItemName>Name</MenuItemName>
            </Action>
            <Action>
               <TaskName>Ask character age</TaskName>
               <MenuItemName>Age</MenuItemName>
            </Action>
         </Menu>
      </Menu>
      <Action>
         <TaskName>Give item to character</TaskName>
         <MenuItemName>Give item</MenuItemName>
      </Action>
   </Menu>
</ActionHierarchy>
```

Figure 2.14. Example of an XML file defining a menu.

arbitrary number of actions and/or submenus. By distinguishing between task name and menu item name, we made task definition independent from menu definition. This allows developers, for example, to use the same task repository while showing different task names tailored to the user. This can be useful when groups of people with different levels of expertise must be trained. For example, a serious game for emergency medical training could be used by both professional healthcare personnel and people with no medical formation. To the second group only a small subset of the tasks available to the first would be available. Professional healthcare personnel is used to use abbreviations or acronyms for many tasks, so they could expect to see the same abbreviations or acronyms in the menus. However, the second group would probably not know these abbreviations and need more descriptive names in the menus.

2.2 Implementation aspects

The architecture has been developed in C#. We defined two namespaces under which the classes of the architecture are grouped. First, the Logger namespace includes only the homonymous class which implements the Logger module (described in section 2.1). The operations performed by the Logger class are straightforward, so we do not deal with them in this section. Second, the GameLogic namespace gathers classes implementing the functions of the Scenario Simulator module and of the Serious Game Coordinator module. A class called Narrator is responsible for reading CTT models describing the scenario and executing them. It exploits the methods of a class called SpatialPreconditionChecker to know if task spatial conditions are satisfied before executing them and the methods of the EffectLauncher class to execute task effects.

The main functions of the SeriousGameCoordinator class are the following:

- it synchronizes the classes managing preconditions and effects with the game engine tick, a signal used to set the frequency of time-dependent events in a game (such as rendering). This is necessary to correctly execute effects with duration and to wait for preconditions to be satisfied;
- it provides public methods that the game engine can use to access data (e.g., tasks) and to initialize the Scenario Simulator module once the scenario has been chosen and the training session starts;
- it defines the events to which the game engine must subscribe to receive information from the modules of the architecture;
- it provides the methods and public fields that the game engine must call to send information to the modules of the architecture.

From the point of view of a serious game developer using the proposed architecture, there are six fundamental events to which the game engine must subscribe:

- the EffectFired event fires when an effect of a task is executed. By handling this event, the game engine shows the consequences of task selection to the user;

- the EffectCompleted event fires when the execution of an effect is completed. By handling this event, the game engine can, for example, go back to the default camera after having changed it because of an effect of type ChangeCameraEffectType;
- the SpatialConditionUnsatisfied event is fired when a spatial condition is not satisfied and it must be satisfied by the system (see section 2.1.2.1);
- the SpatialConditionMessageSent event is fired when an unsatisfied spatial condition must be satisfied by the user and carries the message to be shown to her;
- the SpatialConditionsDataRequested event is fired when data is needed to check if a spatial condition is satisfied. By handling this event, the game engine sends the necessary data to the Scenario Simulator module;
- the ScenarioCompleted event is fired when the scenario is completed. It can be used, for example, to trigger the visualization of debriefing information or to start a final cutscene.

3

Case study: a serious game for Advanced Life Support training

Based on the architecture described in chapter 2, we developed a serious game called EMSAVE, aimed at training EMS nurses (i.e., nurses working in ambulance services) in decision making during medical emergencies. We developed it in collaboration with nurses, instructors and medics working for the Italian EMS. In this chapter, we describe how we created it, in order to illustrate how the architecture is used to build serious games for scenario-based training. A similar process can be followed to create serious games in other domains.

3.1 Conventional training of EMS nurses

In order to be able to develop EMSAVE, we initially studied how EMS nurses work and train. We collected relevant information from interviews with nurses, instructors and medics working for the Italian EMS, who also collaborated during the entire production process. When EMS nurses face an emergency, they have to correctly analyze the situation, assess conditions of people involved, choose the appropriate procedures and perform them. Moreover, they typically need to operate under strict time constraints. EMS nurses training includes classroom lessons, case studies and live simulations. Emergency procedures are often described by flowcharts that have to be memorized by nurses. During live simulations, nurses operate on mannequins or actors under the supervision of one or more instructors, who guide them and evaluate their performance, based on a

predetermined scenario describing the initial situation, tasks that the trainee has to perform and how the scenario proceeds consequently. More rarely, larger live simulations (reproducing, for example, mass disasters) involving many people are organized, with actors playing patients and emergency situations recreated in every detail. Simulations are necessary for EMS nurses, to test concepts and procedures they learned before having to handle real emergencies, where errors or delays can have serious consequences.

Live simulations require time, money and organizational effort, each time they are carried out. On the contrary, the development costs of a serious game, if it also provides the tools for conveniently creating new content (e.g., creating new training scenarios for the game), must be sustained only once. Then, the serious game can always be available and can be used anywhere, including at home. Training sessions need to be periodically repeated during nurses' careers, mainly because: (i) emergency procedures are regularly updated, as a consequence of progress in medicine (e.g., new knowledge or changes in medical devices); (ii) some kinds of emergency situations occur rarely, and they have to be rehearsed through training, to maintain the ability to effectively handle them. Thus, nurses need training tools which can be frequently, conveniently and easily accessed and used, to reduce time needed and increase opportunities for training: serious games are really promising to this purpose.

We also had the possibility to observe conventional training sessions with live simulations. In particular, we observed simulations carried out for training in Advanced Life Support (ALS) procedures. These procedures are exclusively employed in cardiopulmonary emergencies. During ALS procedures, depending on the type of cardiac arrhythmia, defibrillation is applied, and medication is administered. Also oxygen is administered and sometimes endotracheal intubation must be performed to secure the airway. At regular intervals, the effect of the treatment on the heart rhythm, as well as the presence of cardiac output, is assessed on a cardiac monitor or through palpation and auscultation. In case of patients who suffered physical injuries, Advanced Trauma Life Support (ATLS) procedures are used instead. Despite the differences among procedures, training methods are very similar.

From our observations and the information collected during the interviews with the experts, we learned that:

- ALS procedures are always performed by a team, both during real emergencies and training,
- the trainee acts as the team leader, thus being the one making decisions, while the other team members only execute her orders,
- the instructor directs the simulation following a predetermined scenario,
- in most scenarios, besides the EMS team and the patient, characters also include a patient's relative, who can be interviewed to obtain more details about the patient (when the patient himself cannot talk),

- it is up to the instructor to decide when to help the trainee,
- at the end of the simulation, instructors evaluate trainee's performance.

These observations show how training sessions are carried out in a way very similar to a role playing game, with the training scenario as script. These points make our architecture, in which scenarios play a central role, very promising for developing serious games for training EMS nurses.

Every five years, ALS guidelines are internationally updated and medical personnel has to attend a retraining course to learn what changed. Full ALS courses take three days and combine classroom lessons with simulations involving mannequins. Smaller retraining sessions, with the purpose of refreshing already acquired knowledge, should be held during the period between guideline updates but, mostly due to budget shortage, they rarely take place. Serious games could thus be used to reduce retraining costs and make it more convenient. Instead of having to recruit a trainer and organize collective courses, trainees could individually practice on computers at work or at home on the assigned scenarios. Based on these considerations, we decided to develop EMSAVE as a serious game to be used for retraining purposes.

3.2 Application Overview

The user of EMSAVE plays the team leader of an EMS team that is called to rescue a patient outside of the hospital. The virtual rescue team also includes a nurse teammate and a patient's relative character can be involved in scenarios as well. Real-world emergencies are usually handled by teams made of more than two persons, but, in accordance with the experts, we decided to keep their number low in the serious game, to simplify development and lower the risk of unexpected situations during play, due to interactions between virtual character models. The reduced number of team mates did not affect the effectiveness of training, since team mates' unique purpose is to execute some tasks requiring an intervention on the patient, such as administration of medications and application of defibrillation. The trainee plays the role of the team leader, so her only concern is to take the correct decisions about what tasks to perform and when to perform them and this is not affected by the number of team mates.

At the beginning of each training session, the user has to choose a user profile or create a new one. The profile stores the user's gender, so that the user character model in the virtual world can be adapted to it, and associates logs to the right user. After the user profile, scenario and location have to be chosen among the possible ones. To this date, we included in EMSAVE three scenarios. The first one is shorter and simpler to complete, because it is intended as a tool to familiarize with the serious game. In this first scenario, the team leader CTT model contains 125 tasks. In the other two scenarios, instead, the team leader CTT models include respectively 210 and 200 tasks. These two scenarios are more complex, since they are intended to be used for retraining and thus they must represent a challenge for the trainee. We then included two different locations, a private house and a

train station, as possible settings for scenarios. We chose these locations after consulting the experts and receiving from them a list of the places in which emergencies most frequently take place, patient's home and crowded spaces, represented in our case by the train station, were the first two. Users are allowed to play any combination of scenarios and locations. Once scenario and location have been selected, the training session starts. The user is placed inside a virtual world in which the emergency described by the scenario is taking place and is shown a message giving some initial information about the emergency.

EMSAVE has been developed with NeoAxis [76], a game engine programmable in C# and built on the Ogre rendering engine [77]. Exploiting NeoAxis features we developed user navigation in the virtual world, interaction with the virtual world and the graphical user interface (described in section 3.2.1). We also developed the classes controlling NPCs behavior. We employed NeoAxis editing tools to create maps for the two locations mentioned in section 3.2. Finally, as introduced in section 2.1, we implemented the necessary events subscriptions to make the game engine able to send messages to the Serious Game Coordinator module.

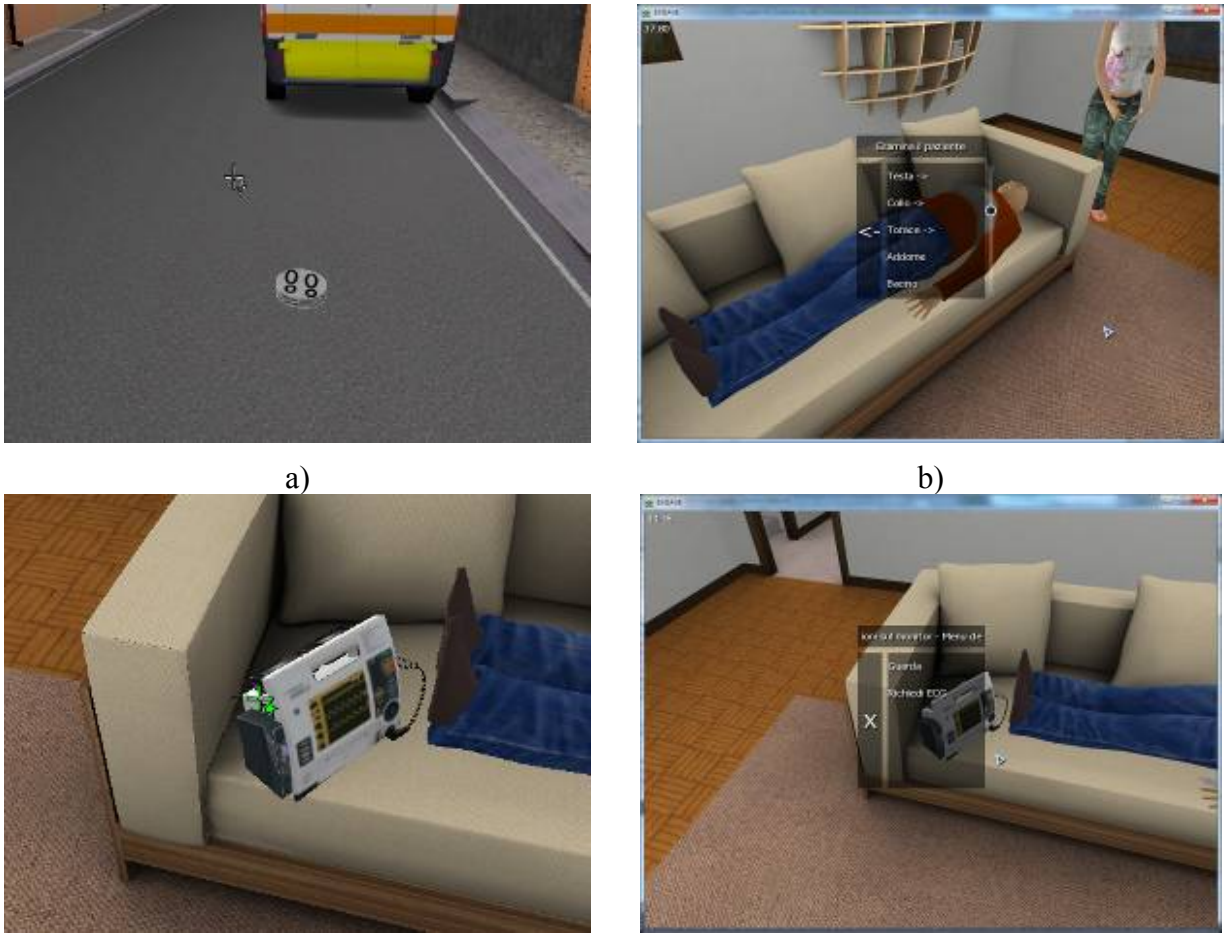
3.2.1 User interaction

The user sees the virtual world in first person. Movement through the virtual world and interaction with objects in it are carried out only through the mouse:

- mouse movement controls rotation of the user viewpoint,
- a left click on the ground makes the user character move towards the clicked position (Figure 3.1a),
- a left click on an interactive object (Figure 3.1c) or character opens a contextual menu (Figure 3.1b and Figure 3.1d), presenting tasks from the task hierarchy (introduced in section 2.1.3) which has that game entity as target.

For example, the contextual menu associated to the patient contains all the tasks that the user can perform on the patient, such as talking to him and carrying out examinations (Figure 3.1b). The user can also interact with the patient's relative character to ask information about the emergency or about the patient. The teammate character, instead, takes part in the execution of the emergency procedure by executing user's orders. Orders are issued by selecting the appropriate tasks from the contextual menu associated to the teammate character. The task hierarchy for each menu was defined together with the experts, also to ease its comprehension by the potential users.

By limiting interaction devices to the mouse only, we aimed at making user interaction as simple as possible (point-and-click interface). As we verified in our user studies (see chapter 4 for more details), most of the potential users of EMSAVE are not computer experts and employ computers only for office work, internet browsing and similar activities. They are used to mouse-based interaction and benefit from an interface which requires to use only the mouse, instead of the typically more complex video game interfaces



a) b)

c) d)

Figure 3.1. Details of the user interface: a) a 3D animated icon indicates the destination of user's avatar movement after a click, b) the contextual menu related to the patient, c) appearance of the mouse pointer hovering over an interactive object (in this case, a medical device), d) the contextual menu related to the medical device.

Figure 3.2. The debriefing window: a red circle denotes a wrong task selection, a green circle denotes a correct one.

To perform the emergency procedure, the user has to select a task at a time. When the user selects a correct task (i.e., a task which satisfies the temporal constraints imposed by the CTT model representing the current scenario, as described in section 2.1.1.1), its spatial conditions are checked. If they are not satisfied, the game engine executes the appropriate instructions or the user is asked to take the necessary actions to satisfy them, depending on what the task specifies. When spatial conditions are satisfied, the effects of that task, defined in the task repository, are executed. Otherwise, if a wrong task is selected, an explanation suggesting how to find the correct task is shown to the user (as described in section 2.1.1.1).

During the training session, the user can access a non-contextual menu by pressing the right mouse button. This menu allows the user to get help in the form of a suggestion of the next correct tasks, to see the history of previously selected tasks or the results of the exams previously executed on the patient.

When the user has performed all of the tasks necessary to handle the emergency, the scenario is completed and a debriefing window is shown (Figure 3.2). This window includes a list of all the tasks the user selected during the training session in the order they were executed and shows which ones were correct or wrong.

```
<Task>
   <TaskName>Chest palpation</TaskName>
   <RoleLabel>TeamLeader</RoleLabel>
   <TargetLabel>Patient</TargetLabel>
   <Conditions>
      <Condition xsi:type="PositionConditionType">
         <Solver>System</Solver>
         <PositionLabel>GenericAnimationOnPatientTorso</PositionLabel>
         <ThresholdDistance>0</ThresholdDistance>
      </Condition>
      <Condition xsi:type="OrientationConditionType">
         <Solver>System</Solver>
         <OrientationLabel>GenericAnimationOnPatientTorso</OrientationLabel>
         <ThresholdRotation>0</ThresholdRotation>
      </Condition>
   </Conditions>
   <Effects>
      <Effect  xsi:type="SerialEffectsType">
         <Effect xsi:type="ParallelEffectsType">
            <Effect xsi:type="ChangeCameraEffectType">
               <UserStoppable>1</UserStoppable>
               <CameraLabel>AtPatientChest</CameraLabel>
            </Effect>
            <Effect xsi:type="SerialEffectsType">
               <Effect xsi:type="AnimationEffectType">
                  <Duration>4</Duration>
                  <GameEntityLabel>TeamLeader</GameEntityLabel>
                  <AnimationName>GenericActionChest</AnimationName>
                  <Loop>true</Loop>
                  <Speed>1</Speed>
               </Effect>
               <Effect xsi:type="TextMessageEffectType">
                  <UserStoppable>1</UserStoppable>
                  <Message>@ChestPalpation</Message>
               </Effect>
            </Effect>
         </Effect>
         <Effect  xsi:type="PositionEffectType">
            <UserStoppable>true</UserStoppable>
            <GameEntityLabel>TeamLeader</GameEntityLabel>
            <PositionLabel>TeamLeaderSafe</PositionLabel>
         </Effect>
         <Effect xsi:type="OrientationEffectType">
            <Duration>0</Duration>
            <GameEntityLabel>TeamLeader</GameEntityLabel>
            <OrientationLabel>TeamLeaderSafe</OrientationLabel>
         </Effect>
      </Effect>
   </Effects>
</Task>
```

Figure 3.3. The XML code for the Chest palpation task in the task repository.

3.3 The task repository for EMSAVE

As described in section 2.1.1, scenarios are made of tasks taken from the task repository. The task repository schema is defined in the architecture and is domain independent, but tasks are domain specific. So, before creating scenarios for EMSAVE, we had to create a task repository suitable for ALS training. To this purpose, we worked together with the experts. First they gave us a small set of the most common tasks which have to be performed in almost every emergency. Then, through various revisions, this set was expanded with other less frequent tasks. This incremental approach allowed us to quickly create a small, but plausible, testing scenario, without having to populate a huge task repository with all the possible tasks necessary for ALS training before being able to conduct preliminary tests of the application.

With the information gathered from the experts, we created the task repository following the schema we defined. The most challenging part of this activity was the definition of the effects for each task. For some tasks, effects were obvious. For example, the effect of task "Put oxygen mask on patient's face" obviously involves having the mask on the patient face. However, to correctly assign effects to some tasks we had to obtain additional information from the experts. For example, drugs are sometimes administered by injection, other times orally, and this detail determines which animation has to be used as an effect of the drug administration task.

Another difficulty in defining task effects was the decision on how they had to be combined, whether in sequence or in parallel. In some cases, to create a believable sequence of effects, we put together complex combinations of nested parallel and serial sequences. For example, Figure 3.3 shows the "Chest Palpation" task in the task repository. The task has several effects, which are shown to the user in the following order:

1. the camera is moved to focus on patient's chest,
2. the team leader character (the one controlled by the user) performs the palpation animation,
3. after some seconds of palpation, a text message communicates the result of the diagnostic task,
4. the team leader character moves back to its default position,
5. the team leader character turns towards the patient.

Effect 1 in the list above must start together with effect 2, so it is put inside an effect of type ParallelEffectType with the effect of type SerialEffectType in which effect 2 and 3 are nested. This nesting is used to make the text message appear only after the animation has played for 4 seconds, as specified by the value of the Duration element inside the effect of type AnimationEffectType. Finally, the effect of type ParallelEffectType including effects 1, 2 and 3 is nested inside an effect of type SerialEffectType and followed by effects 4 and 5. This way, after completing the palpation

animation (effect 2) and receiving the corresponding feedback (effect 3), the team mate character goes back to its initial position and turns to the patient. Figure 3.4 illustrates the execution sequence of these effects.

This task definition also shows an example of how variables are used. More precisely, the Message parameter in the text message effect contains the string "@ChestPalpation". Obviously, the result of a chest palpation is not necessarily the same every time the chest palpation task is performed, it depends on current patient state. When the text message effect is executed, "@ChestPalpation" is translated by the Variable Mapper sub-module (described in section 2.1.1.3) and the message is substituted by a phrase reporting a diagnostic result retrieved from current patient state (as can be seen in Figure 3.6a).

Further work was needed to produce and collect resources necessary to fully implement effects. As described in section 2.1.2.2, effects can involve many types of media: video, pictures, audio. Some audio clips and some pictures were found on the internet and validated by the experts, while others were created specifically for EMSAVE. Some tasks required showing nurses maneuvers as effects. In some cases, they did not need to be shown in detail, so we created generic animations that could approximate them (Figure 3.6b). In other cases, however, it was necessary to show the exact movements that were part of a maneuver. Creating a 3D animation precisely showing such movements would have required a big effort both in terms of time and money. So, we shot videos of actual nurses performing those procedures on an actor playing the patient (Figure 3.6c) under the supervision of the experts. Funds available for serious game development are often too limited to create detailed animations of all the tasks that characters have to perform in the game. When this happens, shooting videos could be a more viable way to show certain tasks in a more detailed way.

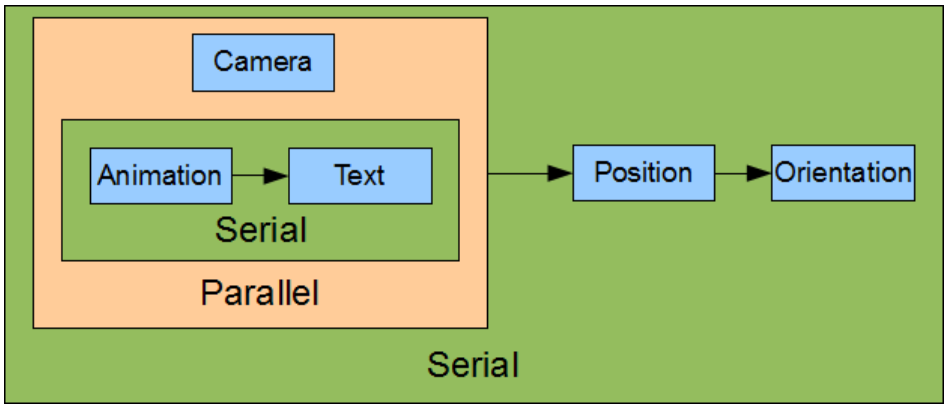


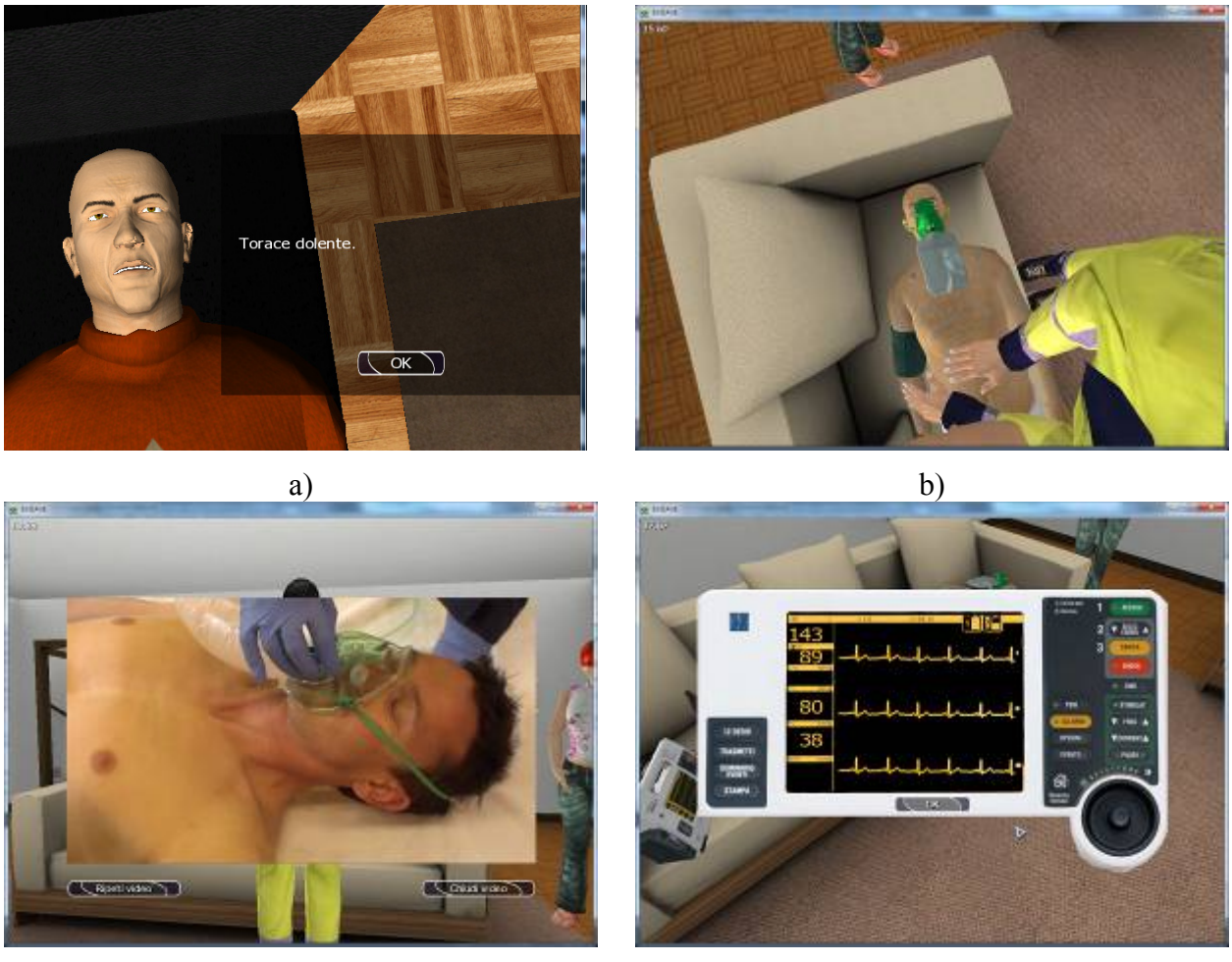
Figure 3.4. Chart illustrating the execution sequence of effects described in Figure 3.3.

Figure 3.5. Yellow flags indicating placeholders in a map.

Spatial conditions have been easy to define. For example, the chest palpation task shown in Figure 3.3 has a position and an orientation spatial condition. As described in section 2.1.2.1, in both spatial conditions we didn't hardwire the exact coordinates. This way, the spatial condition would have required the user character model (since this is a task performed by the user) to be, for example, at the same 3D coordinates regardless of the map used for a specific scenario. Since different scenarios can take place in different locations (i.e., in different virtual worlds), such a solution would have been very limiting. Instead, we use labels which name placeholders that game developers put in each map. Such a solution supports any combination of scenarios and maps, provided game developers correctly place placeholders in each map (see Figure 3.5 for an example of placeholders put in a map using the NeoAxis map editor).

3.3.1 Dialogs in EMSAVE

During an emergency, there are several questions that a nurse must ask to the patient or to other characters, to gather the information necessary for successfully treating the patient. Together with the experts, we identified what these dialogues were and added them in the EMSAVE dialogue repository. We used variables, as described in section 2.1.1.3, that were translated at runtime, based on data retrieved from the user and patient models.



a) b)

c) d)

Figure 3.6. Examples of task effects: a) a message box showing the outcome of the Chest Palpation task, b) generic animation of the user's avatar moving the hands on patient's chest, c) video showing how to put the oxygen mask on patient's face, d) a 2D overlay showing the display of a medical device with data taken from the patient state model.

```
<Effect  xsi:type="DialogEffectType">
<LinePairs>
       <Linepair
              <Interlocutor1Line>
                     <Text>
                     Buongiorno, sono @UserGenderJob del 118, cosa sta succedendo?
                     </Text>
                     <Audio>
                            <AudioFragment>@PresentationTeamLeaderAudio</AudioFragment>
                     </Audio>
                     </Interlocutor1Line>
              <Interlocutor2Line>
                     <Text>@Presentation</Text>
                     <Audio>
                            <AudioFragment>@PatientNameAnswerAudio</AudioFragment>
                            <AudioFragment>@PresentationPatientAudio</AudioFragment>
                     </Audio>
                     </Interlocutor2Line>
              </Linepair
</Linepairs>
</Effect>
```

Figure 3.7. The effect of type DialogEffectType of the task called Presentation.

As regards the audio samples, after writing down the text for each dialogue, we recorded them with human actors. We had to record both the female and the male version for the team leader character since the user has the possibility to choose gender when creating a new user profile.

As an example, consider the dialog added as an effect for the task called Presentation. This is the first task carried out by the nurse after reaching the patient. In it, the nurse presents herself, asks the patient's name and what is going on, while the patient answers with his or her name and some initial details on what's happened and how he or she feels. In Figure 3.7, you can see the XML code for the corresponding effect of type DialogEffectType.

This dialogue is a good example of how and why variables are used in dialogues. Both text and audio for team leader lines must change depending on the job indicated by the user for her character, when the user profile was created. When the user character presents itself, it says it is a nurse. Since in Italian different words are used to indicate a male or female nurse, the @UserGenderJob variable is substituted at runtime with the correct word, specified in the user model. The same holds for the audio part of the question. The @PresentationQuestionAudio variable is translated at runtime with the audio filename specified in the user model.

Several audio fragments can also be combined in a single line, as can be seen in the Interlocutor2Line element in Figure 3.7. While variables corresponding to text fragments are simply substituted by text inside the Text element, to combine different audio fragments, they have to be specified using separate AudioFragment elements, appropriately ordered inside the Audio element. So, in our example, the patient character would first say its name, retrieving the file whose path is

obtained retrieving the value of the @PatientNameAnswerAudio variable, and then explain its situation, retrieving the file whose path is obtained translating the @PresentationPatientAudio variable.

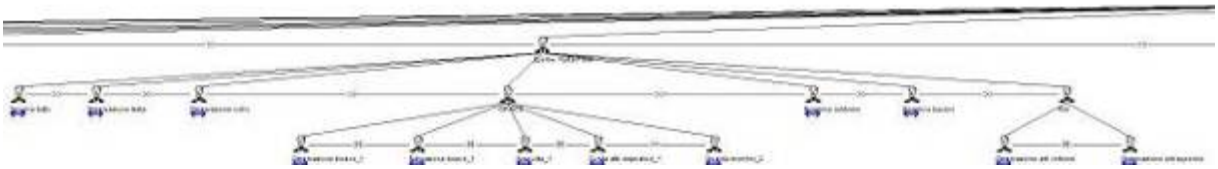
3.4 Scenario CTT models in EMSAVE

In EMSAVE, each scenario includes four CTT models:

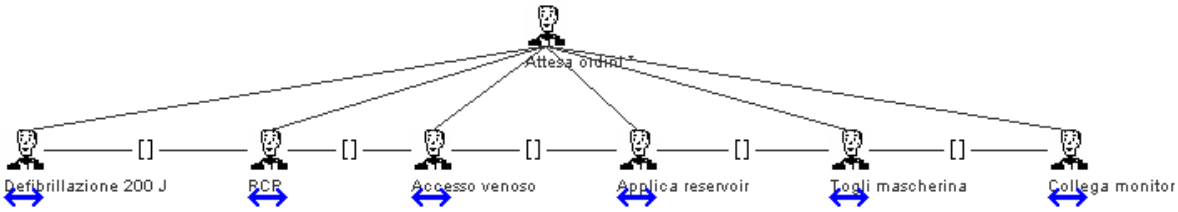
- a team leader CTT model, describing the correct sequence of tasks the user has to perform to successfully complete the scenario (Figure 3.8a),
- a teammate CTT model, describing tasks performed by the teammate NPC, that is a simple two-level tree, in which all leaves are connected through the "Choice" temporal operator and the root node is an iterative task. This way, after a teammate task has been executed, all leaf tasks are available again (Figure 3.8b),
- a patient CTT model, that is another two-level tree, in which all leaves are "change state" tasks (i.e. tasks that tell the scenario simulator to change the current patient state) and are connected one another through the "Enabling" temporal operator, forming a straight sequence of tasks (Figure 3.8c),
- a concurrent CTT model (Figure 3.8d reports a fragment of a concurrent CTT model), describing temporal relations among tasks belonging to the above models (as defined in section 2.1.1).

As described in section 2.1.1, the first and last CTT models are the ones that every serious game based on our architecture must have and always have the same purpose, regardless of the domain. The other two, instead, the ones corresponding to the NPCs, have a purpose specific to EMSAVE. The patient CTT model represents a linear sequence of state changes to resemble scenarios written for real world training. These scenarios always feature an initial patient state at the beginning and several state changes happening during the scenario, sometimes following a task performed by the trainee, sometimes corresponding to a spontaneous evolution of patient conditions. The teammate CTT model, instead, works as a set of always available tasks. It represents the behavior of the teammate character, which does nothing other than wait for orders by the team leader. To correctly describe the sequence of a team leader's order and the execution by the teammate, the concurrent CTT model must include a task corresponding to the order, taken from the team leader CTT model, followed by the appropriate teammate task, connected by an "Enabling" temporal operator. For example, in Figure 3.8d, the team leader task "Richiedi accesso venoso" (which means "request vein cannulation") is followed by the team mate task "Accesso venoso" (which means "Vein

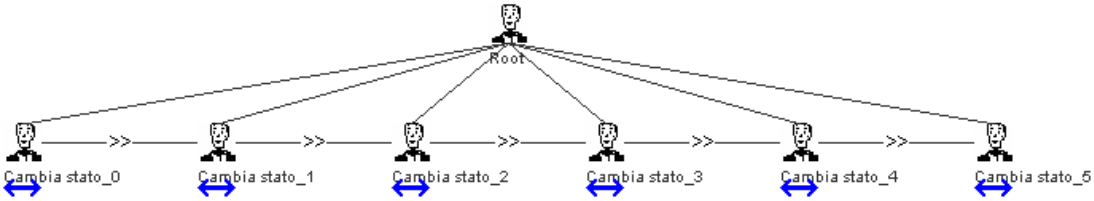
cannulation") and the two are connected by an "Enabling" temporal operator. After executing the team leader task, the scenario simulator finds the teammate task and makes the teammate character reach the patient character and perform an animation indicating that the patient character is being cannulated.



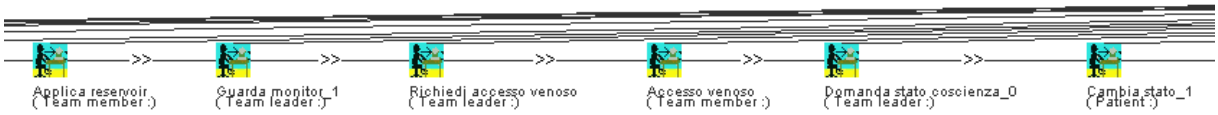
a)



b)



c)



d)

Figure 3.8. Screenshots of a) the team leader CTT model, b) the teammate CTT model, c) the patient CTT model, d) a fragment of the concurrent CTT model

3.5 Patient states

To create the patient state model we extended the generic state model XML schema (as described in section 2.1.1.2), which includes information about the patient and the evolution of his vital signs and of the symptoms he shows throughout the scenario.

Each scenario needs its specific patient state model. The initial part of the XML file includes some static data, regarding personal data about the patient, which are common to every state. They are:

- patient gender,
- answers to questions that the nurse asks during the SAMPLE (which stands for Symptoms, Allergies, Medications, Prior History, Last Meal, Event) interview, that the user has to carry out at a certain point during the scenario,
- the type of cardiac arrest the patient suffered (since ALS scenarios always involve a cardiac arrest).

The remaining part of the patient model contains one or more patient states. We worked together with the experts to identify the symptoms and vital signs to be included in the patient state model, based on the diagnostic tasks a nurse can perform during an ALS scenario, and the possible outcomes for each one of them. For example, as shown in Figure 3.9, the skin on the chest can be normal, pale or red. When the user selects the "Chest observation" task, one among these three values is returned as a result.

```
<xs:element name="ChestObservation">
   <xs:complexType>
      <xs:all>
         <xs:element name="Skin">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="Normal"></xs:enumeration>
                  <xs:enumeration value="Pale"></xs:enumeration>
                  <xs:enumeration value="Red"></xs:enumeration>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
```

Figure 3.9. The XML schema definition of the element describing possible outcomes for the chest observation task

4

User Evaluation

The evaluation of EMSAVE was conducted in two stages. First, we carried out a pilot study on a prototypical version of the serious game, with the purpose of evaluating acceptance of the serious game as a training tool by the potential users (i.e., healthcare professionals employing ALS procedures) and its perceived usability and usefulness. The pilot study also allowed us to collect suggestions and comments which helped to further develop the serious game. Second, on the more advanced version of the serious game described in chapter 3 we conducted another evaluation with the purpose of assessing the effectiveness of EMSAVE as a training tool. We evaluated the effectiveness in terms of knowledge gains:

- by comparing subjects' knowledge of ALS procedures before and after training with the serious game, through a questionnaire created by professional ALS instructors,
- by assessing knowledge retention, through a new administration of the same questionnaire, three months after the training session took place.

In this chapter, we describe the experimental design of both evaluations and the results they produced.

4.1 Pilot study

In the early stages of development, we conducted a study to evaluate nurses' acceptance of the serious game, its usability and its perceived usefulness. Since this kind of training tool was completely new to its potential users, it was necessary for us to know if it would have been well

received and if they found it useful and effective. We also took advantage of this study to collect suggestions about possible new features, which were kept in consideration in later development stages.

4.1.1 EMSAVE prototype

The evaluation was performed on a prototypical version of EMSAVE, which differed from the version described in chapter 3. The most evident differences concern the interface, interaction with the virtual world and task selection.

Camera orientation and user's avatar movement were controlled in the same way as in the later version, except for the fact that a third-person over-the-shoulders view was used instead of the first-person view and that the right mouse button was used to indicate movement destination. There were no interactive 3D objects in the virtual world, information from the virtual medical devices was instead integrated in the user interface (Figure 4.1, lower left) and dialogues with NPCs started automatically at predetermined points in the scenario. A list of the available tasks, presented in random order so as not to suggest their correct sequence, was continuously visible to the user, in a frame on the left side of the screen (Figure 4.1, upper left). To select a task, the user had to press the Enter key on the keyboard, so that the mouse was no more used to move the user's avatar but to

Figure 4.1. The interface of the EMSAVE prototype used in the first user study.

select tasks from the list. After the user double-clicked on the name of the selected task or pressed the Enter key again, the mouse again controlled user's avatar movement. In the later version, instead, tasks were selected from contextual menus which opened after a click on a character or on an object in the virtual world, thus there was no need to switch the mouse from movement control to task selection.

We exploited the prototype to also assess subjects' preference on the methods employed to show the manual procedures performed by nurses' avatars in the game. In the scenario a domain expert created for the evaluation, there were three manual procedures, which we reproduced with animations of the team mate character, while in the later version of EMSAVE videos were used to this purpose. The animation of the first manual procedure was created using motion capture data, while for the other two keyframing was employed. Moreover, the second animation was presented in the form of a rendered video overlaid on the 3D scene, while the other two were actually performed by characters in the virtual world. Motion capture requires more effort than keyframing, but should produce more realistic animations. Similarly, making virtual characters perform actions in the virtual world is more difficult than overlaying a rendered video on the scene, but should provide a more compelling experience. We included all these conditions in the serious game, to test if nurses had strong preferences for one technique over the others and thus receive indications for the future development of the application. Further detail about the prototype is provided in [13].

4.1.2 Experimental design

The pilot study had the following purposes:

- to investigate nurses' acceptance of the serious game in terms of learnability and usability of controls,
- to assess the perceived usefulness and effectiveness of the serious game as a training tool (EMS nurses' training only involves non computer-based activities, so we don't know how they perceive them),
- to collect nurses' judgment about specific details, such as the way we reproduced some patient's symptoms or the different animation realization and presentation techniques we illustrated in section 4.1.1,
- to gather suggestions about possible improvements to the serious game.

We carried out the user study with nurses working for the EMS center of Udine Hospital, Italy. The emergency situation we presented in the evaluation is based on a scenario concerning a tetraplegic person with high fever and respiratory difficulties in his bed at home, assisted by a relative. Patient's respiratory difficulties in the simulation are indicated by a cyanotic complexion, the sound of heavy breathing and the animation of the patient abnormally bending his chest. Emergency nurses must initially concentrate on patient's breathing, putting an oxygen mask on his face. Then, they must measure some vital signs, like heart rate, blood pressure, etc. Finally, the patient must be transferred to a stretcher.

The evaluation took place at the EMS center, the usual workplace of the subjects. Each nurse carried out the test during her work shift change or during a planned break. This way, it was possible for nurses to devote their full attention to testing the application without being subjected to time-pressure or interruptions and distractions. Nurses were initially asked to fill in an anonymous questionnaire containing demographic questions (age, sex, computer experience, videogame experience, etc.). The computer use questions (Table 4.1) were taken from the computer use survey proposed in [25]. From that survey, we did not use questions about how long the subject had been using the Internet and how often she used it, since they were not relevant to our purposes. For the same reason we excluded the question about cell phone usage frequency and about self-assessment of computer skills. After having been instructed about the navigation controls, nurses went through a familiarization phase, in which they were allowed to spend unlimited time in a generic non-emergency virtual world until they felt familiar with the controls. This virtual world was made of two rooms connected by a corridor. Nurses started in the corridor and were asked to reach the room in front of them, which had many boxes in its centre, forcing the player to move along the walls to explore the room. Nurses were asked to fully explore the room around the boxes and to go back to

1. How long have you been using a computer?					
I have never used a computer	Less than a year	About a year	1-2 years	2-5 years	More than 5 years
2. How often do you use a computer?					
I do not use a computer	About once a month	A few times a month	A few times a week	Every day, for less than 1 hour	Every day, for 1-3 hours	Every day, for more than 3 hours
3. What is the main thing you do when you use a computer?					
I do not use a computer	e-mail	Web browsing	Videogames	Document editing (Word, Excel, etc.)	Other (describe)
4. How much do you like using a computer?					
Not very much	A little	Very much			
5. How often do you play videogames?					
I do not play videogames	About once a month	A few times a month	A few times a week	Every dat, but for less than 1 hour	Every day, for 1-3 hours	Every day, for more than 3 hours
6. Did you ever play 3D videogames?					
Yes	No				

Table 4.1 The computer use questionnaire

the corridor. Then, if they were not yet familiar with the controls, they were asked to reach the other room at the opposite end of the corridor. This room was larger and contained several big fantasy objects. Nurses were free of moving around the room as long as they liked.

When nurses felt acquainted with the controls, the evaluation phase, in which nurses faced the emergency scenario written by the experts, started. First, nurses were told that they had to move in the virtual world and choose the appropriate tasks from the proposed list to handle the presented emergency situation. They were instructed about how to interact with the list of tasks and about the fact that tasks in the list would appear in random order. This way, we forced nurses to think about what task to select, preventing them from blindly double clicking the first task on the list. No other information was given, but nurses were invited to ask any question in case of difficulties or make any comment about the whole experience (following a think-aloud protocol), since we were interested in evaluating how quickly all the aspects of the application could be understood. Then, interaction with the application started.

Initially, a text window, introducing the emergency situation and the main goal, is shown to the nurses. After they have read the introduction, they can click on the OK button and start the test. The virtual experience initially places nurses on the front door of the patient's house where they are met by a relative of the patient, who engages them in a brief conversation about what happened. Then, the relative invites nurses to follow her to the patient's bedroom. Once they reach the room, the relative informs nurses that the patient is tetraplegic and they can also see the wheelchair on the patient's bedside. Then, nurses can choose tasks from the proposed list to perform the procedure. When a correct task (i.e. a task satisfying the constraints imposed by the CTT model of the procedure) is chosen, an event is triggered in the virtual world (e.g. an animation). After executing all the proper tasks, when the patient is transferred to the stretcher to take him to the ambulance, a message informs the nurse that she successfully handled the emergency.

After the test, nurses were administered a questionnaire (Table 4.2), with questions about overall player satisfaction, specific application details, such as the patient character or animations, and the perceived usefulness of this kind of applications in training EMS nurses. Some questions were taken from questionnaires that can be found in the literature. Questions 2, 3, 5, 6, 15, 16 and 17 were taken from the Presence Questionnaire proposed in [65]. Question 12 was instead taken from the EGameFlow [19] questionnaire, developed to evaluate games for e-learning, while question 1 comes from a questionnaire to evaluate navigation control in VEs [56]. Most questions were presented in the form of statements to which nurses had to indicate their level of agreement on a five-item Likert scale, ranging from "Totally agree" (1) to "Totally disagree" (5). To have more opportunities to collect subjects' opinions on the experience, we read the statements together with them, discussing and delving deeper into their answers. The number of items in the questionnaire was intentionally kept as low as possible, since we wanted to avoid nurses getting tired or annoyed by the evaluation. The evaluator also interviewed nurses about their ratings and their answers to the open questions, to collect nurses' impressions and feedback thoroughly.

		Mean	Std. Dev.
Interaction			
1	Moving through the virtual world was simple	2.08	0.95
2	I quickly learned how to move through the virtual world	1.83	0.7
3	I could always concentrate on the actions in the game rather than on the commands I had to issue to activate them	2.08	0.86
4	I quickly learned how to select tasks to perform the procedure	1.58	0.64
View			
5	I had a complete view of the virtual world	1.42	0.49
6	I could observe well objects and characters in the virtual world from various points of view	1.42	0.49
Specific application details			
7	The sound of the shortness of breath of the patient is realistic	1.42	0.64
8	The cyanotic complexion of the patient is realistic	1.33	1.31
9	List virtual patient's aspects you liked or disliked (open question)		
10	The application shows three animations, listed below, related to actions belonging to the procedure. Order them from 1 to 3 based on your preference (1 best, 3 worst). Oxygen mask application. Vein cannulation. Patient's transfer to the stretcher.		
11	The characters helped me understand what was happening in the virtual world and what I was supposed to do	2.25	0.83
Usefulness and effectiveness as a training tool			
12	This kind of application could increase my knowledge	1.67	0.85
13	This application would integrate well with the training methods currently employed to train EMS nurses	1.5	0.5
14	In what phase of the training should it be employed? (open question)		
Global satisfaction			
15	I felt involved in the experience	1.75	0.6
16	I liked the graphics of the application	1.67	0.47
17	I liked the audio of the applications	1.67	0.47
18	I will be willing to use this kind of application	1.25	0.43
19	Additional comments (open question)		

Table 4.2. Application evaluation questionnaire (ratings ranged from 1 to 5, lower values correspond to better ratings,). Scores indicate subjects' level of agreement to each statement. Results for questions 9, 10, 14 and 19 are reported and discussed in sections 4.1.4.3, 4.1.4.4, 4.1.4.6.

4.1.3 Participants

We circulated a call for volunteers among the nurses working for the EMS center of Udine Hospital: 12 of them, 8 female and 4 male, answered positively. Subjects did not receive any compensation for participating in the test. Moreover, their superiors and supervisors were not involved in the recruitment, to avoid them feeling forced to participate in the study. The average age was 38.6 years (youngest 28, oldest 50). Answers to question 2 in the computer use questionnaire (Table 4.1) were almost equally distributed among the last three. This means that all subjects use computers daily, although the numbers of hours differ. No nurse indicated playing videogames as her most frequent computer task: the main computer tasks were almost equally distributed among e-mail, web browsing, editing documents and using computers at work. Most subjects said that they like to use computers very much. Use of videogames was rare: 7 nurses played videogames a few times a month, 4 nurses played about once a month, only one nurse never played videogames. Finally, we asked whether subjects ever played 3D videogames, making sure that they understood we referred to videogames in which the action took place inside a 3D virtual world, by giving some examples of well-known videogames. It turned out that 7 subjects never played 3D videogames.

4.1.4 Results

In this section, we report the results of the evaluation , together with some observations. These results were also presented at the "2nd International conference on Games and Virtual Worlds for Serious Applications" [14].

4.1.4.1 Familiarization Phase

The purpose of this phase was to allow nurses to familiarize with navigation controls. We measured the duration of this phase, to assess its difficulty (the longer the familiarization phase, the harder to learn navigation controls): the average training time was about 2 minutes. However, seven nurses took 90 seconds or less to get used to the navigation controls. The two slowest nurses took about 5 minutes to familiarize with the controls. They both did not have any experience with videogames and one of them did not like to use computers while the other one liked it only a little. A particular case is represented by a nurse having experience with some console games, who took nearly 4 minutes to familiarize with the controls. Because of the control system she was used to, she would have preferred to use arrow keys to navigate the virtual world rather than the mouse. Overall, the control system we adopted turned out to be easily learnable by users with no or very little experience with 3D videogames, as confirmed by questionnaire results. The first 3 statements, concerning the difficulty of familiarizing with the controls, received positive ratings as shown in Table 4.2. The best rating (1.83) was the one related to the learnability of the control system.

4.1.4.2 Evaluation Phase

The evaluation phase was completed in an average time of nearly 7 minutes (mean: 419 seconds, standard deviation: 83.38 seconds). During this phase, nurses did not show particular difficulties with the controls or the interface. Ratings of statement 4 ("I quickly learned how to select tasks to perform to carry out the procedure") averaged a very good result (mean: 1.58, standard deviation: 0.64). This result was predictable, considering that computer use questions pointed out that the main computer tasks carried out by the subjects usually involve traditional 2D interfaces with menus.

4.1.4.3 Specific Application Details

Statements 5 to 11 in Table 4.2 investigated the satisfaction of the subjects about some specific aspects of the application. The provided view proved adequate to the task, allowing nurses to completely survey the virtual world and objects and characters in it (statements 5 and 6, both with mean 1.42 and standard deviation 0.49). During the interviews we had while designing and developing EMSAVE, clinicians were very demanding when it came to the simulation of the patient, so we used statements 7 to 9 to collect nurses' opinions about the patient character. Aside from the particular observations of each single nurse, it is worth noting that almost all nurses paid considerable attention to the symptoms shown by the patient. The number and the level of detail of their comments stress how a credible simulation of the symptoms and of the reactions of the patient to the procedure is crucial for this kind of applications. To assess nurses' subjective preference about the three animations used to show manual procedures, we asked them to put them in order of preference, from the best to the worst (question 10). We assigned a score of 1, 2, 3 respectively to the first, second and third condition. The average scores for the three animations were 1.75 (standard deviation 0.72) for the animation created with motion capture, 1.92 (standard deviation 0.96) for the rendered animation, 1.83 (standard deviation 0.8) for the animation created through keyframing. Friedman's test pointed out that the small differences among these averages are not statistically significant. The ratings obtained highlight how there was no clear preference among the considered animations, although we expected the rendered video to perform worse than the other two, since the overlaid video could have created a break in the sense of immersion.

4.1.4.4 Usefulness and Effectiveness as a Training Tool

There was a substantial agreement about the usefulness of the game as a tool for training nurses. Asked if this kind of application could increase their knowledge (statement 12), nurses answered with an average score of 1.67 (standard deviation 0.85). This result is encouraging, indicating that training with serious games could be well accepted by nurses. Nurses thought also that this kind of training could be easily integrated with current training procedures for EMS nurses (statement 13, mean 1.5, standard deviation 0.5). In particular, by interviewing them in more depth, we got the

following comments and suggestions (the first two were given by almost all nurses, while each one of the others were suggested by a single nurse only):

- the application could be used to train novice EMS nurses who still have to learn all the emergency procedures, even the simplest ones;
- the application could be used during retraining, which occurs periodically during a nurse's career to refresh previously acquired knowledge;
- the need to concentrate on the patient during real emergencies could prevent nurses from paying attention to each possible detail of the procedure, so the application could be used to analyze emergency situations and the corresponding procedures more carefully;
- the application could always be available to nurses at the workplace to refresh procedures when there is some time available (e.g., waiting for an emergency call), and it could also be used at home;
- the application could be used to teach procedure updates;
- the application could be used to test nurses' knowledge, substituting written exams;
- a multi-user experience could improve collaboration among the different responders involved in the emergency, such as nurses, ambulance drivers and possible volunteer rescuers, by allowing them to communicate and cooperate applying their specific skills.

4.1.4.5 Overall Satisfaction

Statements about realism of graphics and audio (16 and 17) received the same high score (mean: 1.67, standard deviation: 0.47). In particular, subjects appreciated the fact that uniforms used for the 3D models of nurses in the virtual world and medical devices had been modeled after the real ones. This brought the simulation closer to their everyday life experience. Although the graphical and audio quality of this first prototype did not reach the level of commercial videogames, the high level of satisfaction from the nurses' side indicates that it was sufficient for the specific training application.

Almost all nurses reported a high or very high involvement (statement 15, mean: 1.75, standard deviation: 0.6). This happened despite the fact that no particular interaction devices for immersive VR were used and an observer was present.

Finally, statement 18 tested if nurses would willingly use this kind of application for training. The average score was the highest of the whole questionnaire: 1.25 (standard deviation: 0.43). In particular, some nurses were really enthusiastic about the possibility of having such applications available everyday for training.

4.1.4.6 Nurses' Feedback

During the evaluation, we invited nurses to share their thoughts about the application, whether they were questions, critiques or proposals. Most comments were related to minor improvements

we should implement, but some identified new requirements for the application or important desired features. In the following, we report the most significant comments.

In the virtual world, users can see an assistant nurse character which remains passive most of the time. It was included into the virtual world for two reasons: (i) to make the presented situation more realistic, since actual ambulance runs do not involve less than two nurses; (ii) to take part in moving the patient to the stretcher, an action required by the considered emergency medical procedure. One of the nurses wished it was possible to issue orders to the other nurse, since in real situations different nurses deal at the same time with different aspects of the emergency. This is an improvement which could increase the complexity of possible actions in the virtual world and pose more challenges to the user, providing more learning opportunities.

We were expecting nurses to generally require more verbal interactions with characters than we provided in the application. However, only one of them said it would have been good if he could ask questions to the other characters. Similarly, another nurse pointed out that questions made by the main character to the relative during conversations were not the ones she would have made in a real situation, so the possibility of choosing questions would have made conversations more useful.

Making the application adaptive to different professional levels was suggested by one of the nurses. Since, in real emergency situations, people with different skills and goals are involved (e.g. nurses and volunteer rescuers), the application should be aimed at training all these different responders and automatically adapt to different users. For example, the set of tasks a user can choose should change with respect to the role she has.

Some nurses had experimented other multimedia applications for training, in the form of hypertexts, integrating videos and animations, which presented emergency situations and asked trainees to select the correct procedure or just showed it. Nurses compared the serious game with these applications or with the current non computer-based training methods. In each case the application was seen as a better way to convey knowledge, mainly because of the immersive elements it introduces.

Finally, by observing nurses while they navigated the virtual world, we noticed that several of them tended to collide with walls or other obstacles, forcing the camera to positions that made it difficult to move away from those obstacles. By adopting a first person viewpoint in the later version of EMSAVE, this problem was removed.

4.2 Evaluation of EMSAVE in ALS retraining

We designed and carried out a user evaluation with healthcare professionals on the version of EMSAVE described in chapter 3. The main objective of the evaluation was to assess the effectiveness of EMSAVE as an ALS retraining tool, by measuring knowledge gains after training with the serious game. Moreover, we also investigated subjects' perceived usefulness and effectiveness of EMSAVE as a retraining tool. As during the entire development process, ALS training experts were involved in designing the experimental evaluation. Every detail of the evaluation procedure was discussed with the experts, to ensure its effectiveness in testing the impact of EMSAVE on trainees' knowledge.

4.2.1 Procedure

The user evaluation was composed of a training event and a retention test. To propose a training experience familiar to the ones to which subjects were used, we worked with the "SOS formazione" (the center for hospital training) and the experts on the organization of the evaluation. It was not possible to integrate the evaluation in a full ALS training course, because they have a structure and methodologies that cannot be modified, require the presence of several instructors and last 3 days. So, we agreed on proposing an ALS retraining course lasting only half a work day, that is three hours and a half to four hours, since we had more freedom to structure it as we needed.

The experiment followed a within-subject design. The procedure we followed was composed of the following steps (with the corresponding durations):

1. subjects filled out a questionnaire about demographics, computer use (Table 4.3), use of 3D videogames and 3D applications (Table 4.4), subjects' initial level of preparation(Table 4.5), feelings towards ALS training and expectations from virtual reality for training (Table 4.6). The computer use questions were the same used for the pilot study (Table 4.1) with slight variations: i) an initial question was added, which excluded all the following in case of a negative answer, ii) answers to question 2 were changed to cover a longer period of time, iii) answers to question 4 were grouped in more coherent categories iii) for the answers to question 5 a 5-item scale instead of a 3-item scale was used, to uniform it to the rest of the questionnaire. Similar questions were used to survey 3D videogames and 3D applications use, with the addition of a question asking about the devices used for 3D applications. If subjects would have found the mouse-based interaction we proposed too difficult, we could have investigated possible influences by the devices they were used to. The last part of the questionnaire was proposed by the ALS instructors and included a set of questions they usually employ to assess courses outcomes (30 minutes together with the following step);
2. subjects were administered a multiple choice test (that we call pre-test), made of 38 questions about ALS procedures, each one with one correct answer out of a possible

four, to assess their initial knowledge. The questionnaire was created by professional ALS instructors (who also wrote the scenarios used for the evaluation), in order to test the knowledge necessary to successfully complete the testing scenarios. In other words, they made sure that the questionnaire only included questions which could test knowledge gains caused by playing the scenarios. Our hypothesis was that there would have been a statistically significant difference between the number of correct questions before and after the testing scenarios;

3. subjects were instructed on how to use EMSAVE, playing along a brief demonstration held by one of the developers (15 minutes);
4. subjects went through a simple familiarization scenario, with the purpose of getting acquainted to the application, its controls and its interface (30 minutes);
5. subjects played two testing scenarios, which they were required to complete, without any help from the experimenter, except for unexpected situations, such as application crashes. Scenarios were written by professional ALS instructors, who were asked to create scenarios of similar difficulty but regarding different emergency situations, which would have allowed to cover various ALS concepts. Both scenarios included a patient with cardiac issues which resulted in a cardiac arrest, but the causes of these issues differed, so subjects had to handle the emergencies in distinct ways. Each scenario also included a team mate executing user's orders and a relative of the patient to answer questions in case the patient lost consciousness. The setting for the first scenario was the patient's house, while for the second scenario the setting was a train station. In both scenarios users had to reach the patient and execute the necessary tasks to handle the emergency.(60 minutes);
6. the same 38 questions of the pre-test were administered as a post-test to measure knowledge gains (30 minutes, together with the following step);
7. subjects were administered a questionnaire about perceived usefulness, effectiveness, usability and validity of EMSAVE and their opinion about the application and the whole experience (Table 4.7);
8. subjects were invited to share their thoughts about the application, how they thought its employment would affect training and any remarks or proposals they had (30 minutes).

Three months after the retraining course, a retention test was carried out. Participants to the training received the same ALS questionnaire used for the pre and post-test and were asked to compile it and send it back. By comparing results from the three tests, we were able to assess how much knowledge had been retained over time.

4.2.2 Participants and setting

Since we proposed a retraining course, we involved only subjects which had already received ALS training. We only allowed subjects who knew the most recent guidelines update, which, at the

1. Do you use computers?				
Yes	No			
2. How long have you been using computers?				
Less than a year	1-2 years	2-5 years	5-10 years	More than 10 years
3. How often do you use computers?				
Once a month or less	A few times a month	A few times a week	Every day for 1 to 3 hours	Every day for more than 3 hours
4. What is the main thing you do when you use a computer? (Check all that apply)				
Applications related to my profession	Office applications, email and Internet	Multimedia applications	Videogames	Other: _____
5. How much do you like using a computer?				
Not at all	A little	Quite like it	A lot	Really like it

Table 4.3. The computer use questionnaire.

time we organized the evaluation, dated back to 2005. Otherwise, subjects could have made errors in the ALS questionnaire or in scenarios following the wrong guidelines.

Subjects were recruited through several channels. First, the training event was advertised in the "Foglio Informativo", a monthly publication reporting all courses available to medical personnel, which is distributed in all departments of the Udine hospital and also downloadable from the hospital website. These courses award their participants CME (abbreviation for Continuing Medical Education) credits. This credit system is used internationally (e.g., in Europe, Great Britain, Canada and the United States) to help those in the medical field maintain competence and learn about new and developing areas of their field. In particular, all people working for the Italian public healthcare have to obtain a fixed amount of CME credits each year (as happens, for example, in several of the United States). Second, an information leaflet specifically dedicated to the event was sent to all the hospital departments dealing with emergencies and critical patients. Finally, the nurse coordinators of those departments were directly informed of the event through email and phone. 40 people answered positively to the call. As a compensation for their participation they received 6 CME credits.

4.2.2.1 Demographics, computer use, 3D videogames and 3D applications use

Of the 40 participants, 25 were female (62,5% of the total) and 15 male (37,5% of the total). This gender distribution and in particular the predominance of women, is close to the actual gender distribution of Italian professional nurses: 78% of the 395491 members of IPASVI (the federation of Italian nurses) are female. The average age was 35.52, the youngest participant was 24, while the oldest 49. From the computer use questionnaire (Table 4.3) we learned that all subjects use computers and only 2 of them have been using a computer for less than 1 year. The rest of them have been using computers for at least 2 years, with 33 people using computers for 5 years or more.

1. Do you use 3D videogames or other 3D applications?				
Yes	No			
2. How long have you been using them?				
Less than a year	1-2 years	2-5 years	5-10 years	More than 10 years
3. How often do you use them?				
Once a month or less	A few times a month	A few times a week	Every day, for 1 to 3 hours	Every day, for more than 3 hours
4. What devices do you employ when you use them? (Check all that apply)				
Mouse	Keyboard	Game devices (e.g., joypad)	Movement sensor (e.g., Wii)	Touchscrenn (e.g., on a mobile phone)
5. How much do you like using them?				
Not at all	A little	Quite like it	A lot	Really like it

Table 4.4. The 3D videogames and 3D applications use questionnaire.

Only 2 subjects use computers less than once a week, while 26 subjects use it at least once a day. On a scale from 1 to 5, subjects answered 3.575 on average (standard deviation 0.78) to the question on how much they liked using computers. Only 6 people indicated that they use videogames, thus the majority of the subjects were probably not used to the kind of experience we proposed.

Only 5 subjects used 3D videogames or 3D applications before (Table 4.4), and no one of them had more than 5 years of experience. Also, the frequency of use was quite low: no subject uses them daily. As regards the control devices , all 5 subjects are used to mouse and keyboard. Finally, to the question "How much do you like using them?" the 5 subjects answered with an average score of 3.4 out of 5 (standard deviation 0.548) on a 5-item Likert scale, with higher values corresponding to better ratings.

4.2.2.2 Initial level of preparation

Questions in Table 4.5 helped assessing the subjects' initial level of preparation. 16 subjects worked in the emergency room, 12 subjects in the emergency services (i.e., on ambulances): these are the two departments were ALS procedures are needed more often and more frequently applied. 11 people worked at the Intensive Care Unit while the remaining one was an anesthesiologist. Moreover, 28 people indicated that they employed ALS procedures often or always. However, 24 subjects declared their knowledge of such procedures was only sufficient, while 15 subjects deemed it passable. No one reported a very good knowledge in the field. Only 9 subjects ever attended an ALS retraining course, despite 23 of them thought that they should be held every 1 or 2 years and 14 of them thought they should be repeated twice a year. This significant discrepancy between the number of people that never received ALS retraining and the widespread need for it underline the importance of a tool, such as serious games, to make retraining convenient, cheap and feasible.

1. Which department do you work for?				
ER	EMS	ICU	Cardiology	Other:_____
2. How often are you involved in the ALS team in your department?				
Never	Rarely	Sometimes	Often	Always
3. How would you rate your ALS knowledge level?				
Very bad	Poor	Sufficient	Passable	Very good
4. Which agency provided the last ALS course you attended?				
IRC/ERC	AHA	AISACE	Hospital agency	Other:_____
5. Have you ever attended an ALS retraining course?				
Yes	No			
6. How often do you think you should attend an ALS retraining course?				
Once every 5 years or less	Once every 1-2 years	Once every 6 months	Once a month	Once a week or more
7.Have you ever participated in a course exploiting virtual reality simulations?				
Yes	No			
8. Did you specifically prepare to attend this course?				
Yes	No			

Table 4.5. Questions about working department, ALS use and knowledge.

4.2.2.3 Feelings towards ALS training and expectations from virtual reality for training

The final questions (Table 4.6) were presented in the form of statements to which subjects had to indicate their level of agreement on a five-item Likert scale, ranging from "Totally disagree" (1) to "Totally agree" (5). Regarding the first aspect, that is feelings towards ALS training:

- subjects strongly felt the need to refresh their knowledge of ALS procedures (average score of 4.125, standard deviation 0.757),
- they generally felt neither comfortable nor uncomfortable during ALS simulations (average score of 3.125, standard deviation 0.9),
- they also generally did not feel under stress during ALS simulations (average score of 2.975, standard deviation 1.07).

At the same time, they thought virtual reality could really be useful for ALS retraining (average score of 4.075, standard deviation 0.888) and that it will become a widespread training tool in the future (average score of 4, standard deviation 0.716). Although positive, these results could have been influenced by the voluntary nature of participation to the evaluation: subjects could already be well-disposed towards training in virtual reality.

1. I feel the need to refresh ALS procedures				
Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
2. I feel uncomfortable when filling out an ALS questionnaire				
3. I feel under stress when facing an ALS simulation				
4. I think that a virtual reality simulation tool, like the one that will be used for this course, can be useful to refresh and keep updated my ALS knowledge				
5. I think that in the future virtual reality simulations will become a widespread retraining tool				

Table 4.6. Questions about feeling towards ALS and the usefulness of virtual reality for it (all questions have the same set of possible answers).

4.2.3 Results

In this section, we report and comment results from the ALS and the acceptance questionnaire, participants' remarks and suggestions about EMSAVE and results from the retention test.

4.2.3.1 ALS questionnaire results.

As mentioned above, we administered the same set of questions about ALS before and after the two testing scenarios. Answers to the pre-test showed that, on average, subjects had some gaps in their knowledge of ALS procedures (Figure 4.2). The total number of correct answers was 943 out of 1512 (38 questions times 40 participants), with an average of 23.575 correct answers per subject (standard deviation 4.26). The maximum number of correct answers given by a subject was 31, while the minimum was 13. With an average score too close to 38, we would have had a ceiling effect, which would have left too little space for improvement and made it difficult to measure statistically significant knowledge gains. Besides, it was also confirmed that participants needed

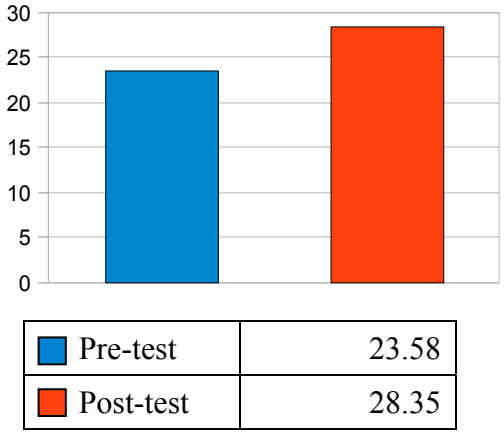


Figure 4.2. Chart showing the average number of correct answers to the pre-test and the post-test.

ALS retraining.

The post-test questionnaire received 1134 positive answers, 191 more than the pre-test, with an average of 28.35 correct answers per subject (standard deviation of 3.676). We analyzed the results using the Wilcoxon matched pairs test, which can be employed to evaluate the difference between two observations of the same group, before and after an experiment. It applied to our case, since we had a single group of subjects who went through the same experiment (i.e., the training session) and were observed before and after it through the pre-test and post-test questionnaires. The computed difference between the two averages of correct answers was 4.8, which is a statistically significant difference ($T = 8$, $p < 0.05$). The total of the ranks where performance after the post-test was higher was 8 and the total of the ranks where performance was lower was 812. The effect size was large, $r = 0.85$. This means that completing scenarios with EMSAVE improved subjects' knowledge of ALS procedures.

4.2.3.2 Perceived usefulness, effectiveness, usability and validity of EMSAVE and opinions about the application and the whole experience

All the answers to this questionnaire proved statistically significant, since p-value, calculated through the chi-square goodness of fit test, was less than 0.001 for each of them. The first question

1. How often do you think the virtual reality simulator should be used for training?				
Once every 5 years or less	Once every 1-2 years	Once every 6 months	Once a month	Once a week or more
2. I think this is a valid self-training tool to refresh ALS knowledge and keep it up to date				
Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
3. After completing the simulated scenarios, I feel more confident about ALS procedures				
4. The proposed scenarios are credible				
5. The proposed scenarios are in line with the ones proposed by instructors during ALS courses				
6. I think that in the future virtual reality simulations will become a widespread retraining tool				
7. I will gladly use this tool for retraining if it was available on my home computer				
8. I will gladly use this tool for retraining if it was available at my working department				
9. I'd like that every month one hour of my work was dedicated to training with virtual reality				
10. It was easy to refresh procedures using this tool				
11. I felt involved while performing procedures with this tool				
12. Overall, I am satisfied by this retraining tool				
13. I positively value the training experience had in this course				

Table 4.7. Evaluation and acceptance of EMSAVE as a training tool (All questions but the first one have the same possible answers).

asked how frequently subjects thought the virtual reality simulator should be used. Nineteen of them answered "once every 6 months" and other 11 answered "once a month". Six people indicated once or more a week as the best frequency. Considering that virtual reality is not part of current training methods and the almost complete lack of experience subjects have with virtual reality applications, these are very encouraging answers: they both reveal that healthcare professionals see virtual reality simulation (at least in the form implemented in EMSAVE) as a viable training tool. Moreover, this question was a slight modification of question number 6 in Table 4.5: "How often do you think you should attend an ALS retraining course?". While the previous question asked about the desired frequency of ALS retraining courses, this one asked about the desired frequency of ALS retraining in virtual reality. Possible choices for the answer were the same, so we could compare answers to the two questions using the Wilcoxon matched pairs test. A significant difference came out in favor of the question asked after the training ($T = 0$, $p < 0.05$). 13 subjects showed no change in agreement after the training and were discarded from the analysis. The total of the ranks where agreement was stronger before the training was 378 and the total of the ranks where agreement was stronger after the training was 0. The effect size was large, $r = 0.87$. This means that subjects would like to attend retraining courses more often if they involved virtual reality simulations.

The remaining questions were presented in the form of statements to which subjects had to indicate their level of agreement on a five-item Likert scale, ranging from "Totally disagree" (1) to "Totally agree" (5). Statements about the effectiveness of EMSAVE received good scores: statement 2, about the validity of EMSAVE as an ALS self training tool, got an average score of 4.375 (standard deviation 0.54). This statement repeated statement number 4 in Table 4.6, we analyzed the difference between the average scores (which was 0.2) using the Wilcoxon matched pairs test, to see if subjects' opinions about the validity of EMSAVE to refresh knowledge had changed after using it, but it was not significant ($T = 11$, $p=0.56$). 29 subjects showed no change in agreement after the training and were discarded from the analysis. The total of the ranks where agreement was stronger before the training was 55 and the total of the ranks where agreement was stronger after the training was 11. The effect size was large, $r = 0.59$.The following statement, about increased confidence in ALS procedures after the simulation, got an average of 3.925 (standard deviation 0.61). Thus, to the considered sample EMSAVE proved very effective in increasing confidence about ALS procedures.

The two following questions tested whether the proposed scenarios were appropriate for ALS training. Scenarios proved both realistic (average score 3.925, standard deviation 0.572) and in line with the ones usually proposed by ALS instructors (average score 4.125, standard deviation 0.404). These results help ensure the validity of the experiment.

Subjects showed confident that virtual reality will become a widespread retraining tool (average score 4.225, standard deviation 0.66). Their opinion did not change with respect to the same question asked before the training (Table 4.6, statement 5), since the difference between the two average scores (0.23), analyzed using the Wilcoxon matched pairs test was not significant ($T = 36$,

p=0.1). 24 subjects showed no change in agreement after the training and were discarded from the analysis. The total of the ranks where agreement was stronger before the training was 100 and the total of the ranks where agreement was stronger after the training was 36. The effect size was medium, r = 0.41. They also proved willing to use EMSAVE for their training, both at work (average score 4.525, standard deviation 0.554) and at home (average score 4.225, standard deviation 0.675). They also would like to dedicate one hour each month to training with EMSAVE: average score 4.3, standard deviation 0.723.

We dedicated the last questions to a brief evaluation of EMSAVE and of the overall experience. Subjects agreed that executing the procedures with our application was not difficult (average score 3.6, standard deviation 0.778). This is a positive result, since, as reported above, almost all subjects did not have any experience with 3D applications. Very important is the fact they felt involved in the experience, which is one of the primary objectives of EMSAVE and of virtual reality training tools in general (average score 4.225, standard deviation 0.62). Overall, the application and the experience had a good impact: subjects were really satisfied by the retraining tool (average score 4.425, standard deviation 0.549) and very positively valued the training experience (average score 4.525, standard deviation 0.506).

4.2.3.3 Miscellaneous comments

In this section, we report the most interesting among comments and suggestions subjects gave at the end of the course.

Comments confirmed the very positive, in some cases enthusiastic, response of the subjects to the whole experience we could already infer from the answers to the last questionnaire. They liked the alternative and innovative approach to training and were overall satisfied by the usability and immersiveness of the application.

As regards ease of use, several subjects reported some difficulties in the beginning, which quickly disappeared during the second and third scenario. The biggest obstacle was the large number of options in the menus that made it difficult to find the desired task. However, after some minutes of practice, all subjects got used to the menu structure and found the needed options more quickly. The choice of using only the mouse as an interaction device and to assign the left mouse button to interact with objects and to navigate and the right mouse button to bring up the menu was appreciated. No subject reported problems with navigation in the virtual world, despite their very limited experience with 3D applications. However, this could be due to the fact that the proposed scenarios required very little exploration of the virtual world. Some subjects highlighted that action was really slower in the simulation than in real emergencies. This was expected and in a certain measure inevitable, because of the fact that users had to select tasks from menus and not directly act on the patient. However, it also forced subjects to carefully consider each and every step of the procedure, while in real emergencies some steps are carried out almost automatically. Many subjects appreciated this, because it helped them reconsider their knowledge and reassess every detail.

The last recurring topic among the comments we received regarded the need for some additional help features during the simulation. Single subjects proposed the following features:

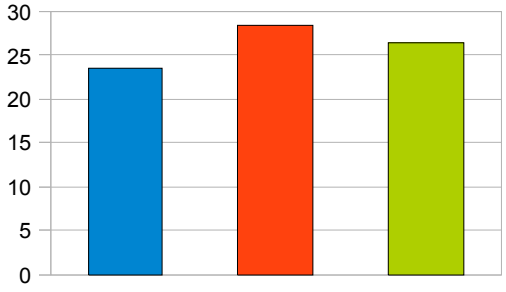
- vital signs and the list of devices attached to the patient should be always shown by the GUI,
- a flowchart for the procedure should be available, both to show the history of performed tasks and advice about the next step,
- guidelines should be shown before each scenario, to refresh them in the user's memory,
- since sometimes the same emergency could be handled in slightly different ways, ambiguities can arise in some scenarios, where the scenario requires the trainee to execute a task while another one could also be correct. In these cases a rationale for the task inserted in the scenario should be given.

The first two points had already been discussed with experts during development. We integrated the task history in form of a list, since procedure execution in EMSAVE takes far more than in real emergencies, thus users could forget some tasks they already carried out. We agreed, instead, not to provide the flowchart suggesting next tasks, as well as panels continuously showing vital signs and the list of attached devices, since they could have made procedure execution too easy.

4.2.3.4 Retention test results

Three months after the evaluation had taken place, 39 participants (one had sadly deceased) received the same ALS questionnaire used for the pre and post-test and were asked to compile it and send it back. All 39 subjects compiled and returned the questionnaire, then we compared the new results to the ones obtained from the pre and post-test questionnaires (Figure 4.3), to measure how much the knowledge gained during the evaluation had been retained.

A one-way analysis of variance (ANOVA) pointed out that there was a significant effect



Pre-test	23.58
Post-test	28.35
Retention test	26.36

Figure 4.3. Chart showing the average number of correct answers to the pre-test, the post-test and the retention test.

($F(2,117)=15.527$, $p<0.05$). We then compared the pre-test and the post-test with the retention-test using the Wilcoxon matched pairs test. As expected, there was a decrease in the number of correct answers to the retention test compared to the post-test: on average, each subject gave 1.9 less correct answers to the retention questionnaire and the difference was statistically significant ($T = 131$, $p < 0.05$). Two subjects showed no change in performance after the training and were discarded from the analysis (together with the deceased one). The total of the ranks where performance after the post-test was higher was 572 and the total of the ranks where performance was lower was 131. The effect size was large, $r = 0.55$. However, the average performance was still better than the pre-test: on average, each subject gave 2.9 more correct answers to the retention questionnaire than to the pre-test one and the difference was statistically significant ($T = 85.5$, $p < 0.05$). Four subjects showed no change in performance after the training and were discarded from the analysis (together with the deceased one). The total of the ranks where performance after the post-test was higher was 85.5 and the total of the ranks where performance was lower was 544.5. The effect size was large, $r = 0.64$.

Although all subjects participating to our study received ALS training, pre-test results showed that they suffered some knowledge loss due to the passage of time. The results of the retention test suggest that part of the knowledge acquired through the use of the serious game was retained by participants to the evaluation after a period of three months. The negative effect of time can be reduced through the repetition of training sessions, which the adoption of a serious game makes more convenient and less expensive than traditional training methods. The only limitation to the retention test is that subjects received questionnaires by e-mail and we cannot be sure that they compiled it without any help or in an environment that favored a level of concentration comparable to the one they had during the evaluation.

Conclusions

In this thesis, we have presented our architecture to support the creation of serious games for scenario-based training. The architecture aims at making serious games production more efficient, by allowing code reuse among different projects and reducing the need for game developers' effort, and at facilitating the involvement of domain experts and their collaboration with game developers.

The architecture exploits the CTT [45] task modeling formalism to represent scenarios with tree-like structures in which nodes represent tasks, which the user has to perform to successfully complete a scenario, and are connected by temporal operators which describe temporal relations between tasks. We also introduced a representation allowing to specify effects, and their execution order, for each task. Through effects, task consequences can be shown in the virtual world. To describe state changes of game entities (e.g., heart rate change in a patient after cardiac massage) we provided a representation that can be extended by game developers based on the domain and on the purpose of the serious game to be developed. State models can be used to store information that changes as the scenario progresses. To refer to information stored in state models when defining task effects, we allow game developers to use their own variables, that are substituted at runtime with values taken from the current state. Game developers only need to define the desired variable mappings, together with task descriptions and state models, at design time and then the defined variables can be used to define any number of scenarios. All the models we use are stored in XML format, which has the advantage of being standard and at the same time processable by computers and readable by humans. This way, game developers and domain experts can work together to define tasks and scenarios for serious games.

The architecture is also game engine-independent. This means that, to exploit the architecture, game developers are not restricted to a single game engine whose features could not meet their requirements. To adapt the architecture to a game engine, game developers only have to implement an event-based communication between the module we called Serious Game Coordinator and the game engine of choice.

By exploiting the XML format, we facilitate the creation of authoring tools that allow domain experts to edit scenarios without the help of game developers. As shown by Johnson and Valente [27], such tools can really determine the success of a serious game, by significantly speeding up the creation of pedagogical content (i.e., training scenarios in our case). We intend to proceed in this direction and work together with domain experts towards the creation of authoring tools they could find easy to use. They will allow the composition of scenarios by dragging and dropping tasks and setting up the appropriate temporal relations among them, in a similar way to the editors proposed by van Est et al. [17] or by Protopsaltis et al. [50]. The usability of these authoring tools will then be

evaluated, by asking experts from several domains in which scenario-based training is practiced to create training scenarios to be used in serious games.

We also plan to test game engine independence. Binsubaih et al. [10] have demonstrated the game engine independence of their architecture by interfacing it both with a simulation engine they created and a commercial game engine. To develop EMSAVE, we used the NeoAxis [76] game engine and to this purpose we implemented the communication between this game engine and the architecture. To test and demonstrate game engine independence, we should port EMSAVE to a different game engine. This test would help evaluate whether the porting procedure is simple enough or it needs to be improved, to ease the work of game developers.

Another direction in which we are moving is the adaptive tailoring of scenarios. Our purpose is to have the serious game tailoring scenarios based on the characteristics of the user or on instructor's indications. Currently, EMSAVE offers the possibility to create scenarios involving disabled patients, based on scenarios which originally involved only non disabled patients. Dealing with a disabled patient requires additional care by the nurse. For example, when talking to a deaf patient the nurse is required to get close in order for him to read her lips. In terms of task descriptions inside our architecture, this implies adding a spatial condition, requiring the user to be within a certain distance from the patient, to tasks corresponding to dialogs. We extended the architecture with a module which changes scenario CTT models and task descriptions, based on the information about the kind of patient's disability. At the beginning of a training session, after a scenario and a disability have been chosen, this module adapts the scenario in several ways. First, it can add new tasks to the scenario. For example, a guide dog is often present close to a blind patient and, until the dog has its harness on, it does not move and could stand in the way of the nurse. So, nurses are required to remove the harness at the beginning of the procedure. When the chosen disability is blindness, the task "Remove harness from guide dog" is added to the scenario in the right place. Second, spatial conditions or effects of existing tasks can be changed. For example, deaf patients have to be able to see the rescuer's face, in order to read her lips while she talks. When the chosen disability is deafness, a spatial condition requiring the user's avatar to be turned towards the patient is added to each task with an effect of type DialogEffectType. Third, the content of the virtual world can be changed. For example, the guide dog mentioned above is added to the virtual world only when the chosen disability is blindness. This feature can be extended by supporting changes based on user characteristics stored in the user model. For example, user skill level could be used to simplify a scenario for beginners or add more tasks for a more experienced user. Moreover, the adaptation mechanism must be tested to verify its applicability on other domains besides medical emergencies.

Finally, we intend to develop more advanced tools to guide and help the user in the training process and to support both assessment of user's performance by the serious game (e.g., through a scoring system, as presented in [36]) and by instructors (e.g., through tools for after-action review, as proposed in [29]). From our work with ALS instructors during the development of EMSAVE, we have seen that the way assessment is carried out strongly depends on the domain: both the assessed

aspects of the performance and how they are evaluated may radically change between domains. A careful and in depth analysis of the assessment process in several domains is necessary, to be able to further extend the architecture while maintaining generality.

Bibliography

[1] D. Alexander. Scenario methodology for teaching principles of emergency management. *Disaster Prevention and Management*, 9(2):89-97, 2000.

[2] G. Alinier. Developing High-Fidelity Health Care Simulation Scenarios: A Guide for Educators and Professionals. *Simulation & Gaming*, 42(1):9 -26, 2011.

[3] P. Backlund, H. Engstrom, C. Hammar, M. Johannesson, and M. Lebram. Sidh - a Game Based Firefighter Training Simulation. In *Proceedings of the 11th International Conference Information Visualization*, IEEE Computer Society Press, pages 899-907, 2007.

[4] T. Baranowski, R. Buday, D. I. Thompson, and J. Baranowski. Playing for Real : Video Games and Stories for Health-Related Behavior Change. *American Journal of Preventive Medicine*, 34(1):74-82, 2008.

[5] K. Becker and J. R. Parker. Serious games + computer science = serious CS. *J. Comput. Small Coll.*, 23(2):40-46, 2007.

[6] F. Bellotti, R. Berta, A. De Gloria, and L. Primavera. A task annotation model for Sandbox Serious Games. In *IEEE Symposium on Computational Intelligence and Games, 2009. CIG 2009*, IEEE Computer Society Press, pages 233-240, 2009.

[7] F. Bellotti, R. Berta, A. De Gloria, and L. Primavera. A task annotation model for Sandbox Serious Games. In *2009 IEEE Symposium on Computational Intelligence and Games*, IEEE Computer Society Press, pages 233-240, 2009.

[8] A. BinSubaih, S. Maddock, and D. Romano. An Architecture for Portable Serious Games. In *Doctoral Symposium hosted at the 20th European Conference on Object Oriented Programming*, 2006.

[9] A. BinSubaih, S. Maddock, and D. Romano. A Domain-Independent Multiplayer Architecture for Training. Presented at the International Workshop in Computer Game Design and Technology, pages 144-151, 2004.

[10] A. BinSubaih, S. Maddock, and D. Romano. Game logic portability. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACM, pages 458–461, 2005.

[11] S. Blackman. Serious games...and less! *SIGGRAPH Comput. Graph.*, 39(1):12-16, 2005.

[12] P. Brusilovskiy. The construction and application of student models in intelligent tutoring systems. *Journal of computer and systems sciences international*, 32(1):70–89, 1994.

[13] A. Cabas Vidani and L. Chittaro. Using a Task Modeling Formalism in the Design of Serious Games for Emergency Medical Procedures. In *Proceedings of the IEEE VS-GAMES International Conference on Games and Virtual Worlds for Serious Applications*, IEEE Computer Society Press, pages 95-102, 2009.

[14] A. Cabas Vidani, L. Chittaro, and E. Carchietti. Assessing Nurses' Acceptance of a Serious Game for Emergency Medical Services. In *2010 Second International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, IEEE Computer Society Press, pages 101-108, 2010.
[15] R. E. Chatham. Games for training. *Commun. ACM*, 50(7):36-43, 2007.
[16] M. Crichton and R. Flin. Training for emergency management: tactical decision games. *Journal of Hazardous Materials*, 88(2-3):255-266, 2001.
[17] C. van Est, R. Poelman, and R. Bidarra. High-Level Scenario Editing for Serious Games. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, Springer Verlag, 2011.
[18] S. de Freitas and S. Jarvis. A Framework for developing serious games to meet learner needs. In *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, 2006.
[19] F. L. Fu, R. C. Su, and S. C. Yu. EGameFlow: A scale to measure learners' enjoyment of e-learning games. *Computers & Education*, 52(1):101-112, 2009.
[20] C. Gatzidis, K. Parry, E. Kavanagh, A. Wilding, and D. Gibson. Towards the Development of an Interactive 3D Coach Training Serious Game. In *Games and Virtual Worlds for Serious Applications, Conference in*, IEEE Computer Society Press, pages 186-189, 2009.
[21] S. Göbel, F. Mehm, S. Radke, and R. Steinmetz. 80days: Adaptive digital storytelling for digital educational games. In *Proceedings of the 2nd International Workshop on StoryTelling and Educational Games STEG'09*, CEUR Workshop Proceedings, 2009.
[22] S. Gobel, L. Salvatore, and R. Konrad. StoryTec: A Digital Storytelling Platform for the Authoring and Experiencing of Interactive and Non-Linear Stories. In *International Conference on Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS '08*, IEEE Computer Society Press, pages 103-110, 2008.
[23] M. Hogan, H. Sabri, and B. Kapralos. Interactive community simulation environment for community health nursing. In *Proceedings of the 2007 conference on Future Play*, ACM, pages 237-240, 2007.
[24] K. Hullett and M. Mateas. Scenario generation for emergency rescue training games. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, ACM, pages 99-106, 2009.
[25] L. A. Jackson et al. Culture, gender and information technology use: A comparison of Chinese and US children. *Computers in Human Behavior*, 24(6):2817-2829, 2008.
[26] S. Jarvis and S. de Freitas. Evaluation of an Immersive Learning Programme to Support Triage Training. In *Proceedings of the IEEE VS-GAMES International Conference on Games and Virtual Worlds for Serious Applications*, IEEE Computer Society Press, pages 117-122, 2009.
[27] W. L. Johnson and A. Valente. Collaborative Authoring Of Serious Games For Language And Culture. Presented at the SimTecT 2008, 2008.
[28] H. Kelly et al. How to build serious games. *Commun. ACM*, 50(7):44-49, 2007.

[29] P. Kizakevich, R. Furberg, R. Hubal, and G. Frank. Virtual Reality Simulation for Multicasualty Triage Training. In *Proceedings of the 2006 I/ITSEC Conference*, 2006.

[30] A. Lecerof and F. Paternò. Automatic Support for Usability Evaluation. *IEEE Trans. Softw. Eng.*, 24(10):863-888, 1998.

[31] M. D. Lynch. Developing a Scenario-Based Training Program: Giving Officers a Tactical Advantage. *FBI Law Enforcement Bulletin*, 74(10):8, 2005.

[32] D. McGrath and D. Hill. UnrealTriage: A Game-Based Simulation for Emergency Response. In *Proceedings of The Huntsville Simulation Conference*, 2004.

[33] A. McKenzie and G. McCalla. Serious Games for Professional Ethics: An Architecture to Support Personalization. In *AIED 2009: 14th International Conference on Artificial Intelligence in Education Workshops Proceedings*, IOS Press, pages 69-78, 2009.

[34] I. C. Moon, M. Schneider, and K. M. Carley. Evolution of Player Skill in the America's Army Game. *Simulation*, 82(11):703-718, 2006.

[35] P. Moreno-Ger, C. Blesius, P. Currier, J. L. Sierra, and B. Fernández-Manjón. Rapid development of game-like interactive simulations for learning clinical procedures. In *Proceedings of Game Design and Technology Workshop and Conference*, 2007.

[36] P. Moreno-Ger, C. Blesius, P. Currier, J. L. Sierra, and B. Fernández-Manjón. Online learning and clinical procedures: rapid development and effective deployment of game-like interactive simulations. In *Transactions on edutainment I*, Springer-Verlag, pages 288-304, 2008.

[37] P. Moreno-Ger, D. Burgos, I. Martínez-Ortiz, J. L. Sierra, and B. Fernández-Manjón. Educational game design for online education. *Comput. Hum. Behav.*, 24(6):2530-2540, 2008.

[38] P. Moreno-Ger, J. L. Sierra, I. Martínez-Ortiz, and B. Fernández-Manjón. A documental approach to adventure game development. *Science of Computer Programming*, 67(1):3-31, 2007.

[39] G. Mori, F. Paternò, and C. Santoro. CTTE: Support for Developing and Analyzing Task Models for Interactive System Design. *IEEE Transactions on Software Engineering*, 28(8):797-813, 2002.

[40] F. La Mura et al. Collaborative Virtual Environments as Research and Teaching Instruments in the Field of Disaster Medicine: the "e-DISTRICT CiPro" Simulator. In *Proceedings of the 11th International Conference Information Visualization*, IEEE Computer Society Press, pages 892-898, 2007.

[41] B. M. Nagle, J. M. McHale, G. A. Alexander, and B. M. French. Incorporating scenario-based simulation into a hospital nursing education program. *Journal of Continuing Education in Nursing*, 40(1):18-25, 2009.

[42] L. Paganelli and F. Paternò. Intelligent analysis of user interactions with web applications. In *Proceedings of the 7th international conference on Intelligent user interfaces*, ACM, pages 111-118, 2002.

[43] S. Pangoli and F. Paternó. Automatic generation of task-oriented help. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, ACM, pages 181–187, 1995.

[44] F. Paternò and C. Mancini. Engineering the Design of Usable Hypermedia. *Empirical Softw. Eng.*, 4(1):11-42, 1999.

[45] F. Paternò. ConcurTaskTrees: An Engineered Approach to Model-based Design of Interactive Systems. In *The Handbook of Analysis for Human-Computer Interaction*, Lawrence Erlbaum Associates, pages 483-500, 2002.

[46] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'97)*, Chapman & Hall, Ltd., pages 362-369, 1997.

[47] F. Paternò and C. Santoro. Preventing user errors by systematic analysis of deviations from the system task model. *Int. J. Hum.-Comput. Stud.*, 56(2):225-245, 2002.

[48] P. Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. In *Interactive Systems Design, Specification, and Verification*, vol. 1946, P. Palanque and F. Paternò, Eds. Springer Berlin Heidelberg, pages 207-226.

[49] M. Prensky. Digital game-based learning. *Comput. Entertain.*, 1(1):21-21, 2003.

[50] A. Protopsaltis et al. Scenario-based serious games repurposing. In *Proceedings of the 29th ACM international conference on Design of communication*, ACM, pages 37–44, 2011.

[51] P. Rooney, K. C. O'Rourke, G. Burke, B. MacNamee, and C. Igbrude. Cross-Disciplinary Approaches for Developing Serious Games in Higher Education. In *Games and Virtual Worlds for Serious Applications, Conference in*, IEEE Computer Society Press, pages 161-165, 2009.

[52] E. Salas, K. A. Wilson, C. S. Burke, and H. A. Priest. Using Simulation-Based Training to Improve Patient Safety: What Does It Take? *Joint Commission Journal on Quality and Patient Safety*, 31(7):363-371, 2005.

[53] Schaafstal A.M., Johnston J.H., and Oser R.L. Training teams for emergency management. *Computers in Human Behavior*, 17(5):615-626, 2001.

[54] B. G. Silverman et al. Modeling emotion and behavior in animated personas to facilitate human behavior change: the case of the HEART-SENSE game. *Health Care Management Science*, 4(3):213–228, 2001.

[55] B. G. Silverman, M. Johns, R. Weaver, and J. Mosley. Authoring Edutainment Stories for Online Players (AESOP): Introducing Gameplay into Interactive Dramas. In *Virtual Storytelling. Using Virtual RealityTechnologies for Storytelling*, vol. 2897, O. Balet, G. Subsol, and P. Torguet, Eds. Springer Berlin Heidelberg, pages 65-73, 2003.

[56] M. Slater, M. Usoh, and A. Steed. Taking steps: the influence of a walking technique on presence in virtual reality. *ACM Trans. Comput.-Hum. Interact.*, 2(3):201-219, 1995.

[57] A. Sliney and D. Murphy. JDoc: A Serious Game for Medical Learning. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, IEEE Computer Society Press, pages 131-136, 2008.
[58] T. Susi, M. Johannesson, and P. Backlund. Serious Games – An Overview. 2007.
[59] P. A. Szekely, P. N. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative interface models for user interface construction tools: the MASTERMIND approach. In *Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction*, Chapman & Hall, Ltd., pages 120–150, 1996.
[60] J. Torrente, A. del Blanco, E. J. Marchiori, P. Moreno-Ger, and B. Fernández-Manjón. <e-Adventure>: Introducing educational games in the learning process. In *2010 IEEE Education Engineering (EDUCON)*, IEEE Computer Society Press, pages 1121-1126, 2010.
[61] J. Torrente, P. Moreno-Ger, B. Fernàndez-Manjòn, and J. L. Sierra. Instructor-Oriented Authoring Tools for Educational Videogames. In *Advanced Learning Technologies, IEEE International Conference on*, IEEE Computer Society Press, pages 516-518, 2008.
[62] M. Turoff, M. Chumer, S. R. Hiltz, A. Hendela, J. Konopka, and X. Yao. Gaming Emergency Preparedness. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 02*, IEEE Computer Society Press, page 38, 2006.
[63] D. S. Vincent, A. Sherstyuk, L. Burgess, and K. K. Connolly. Teaching mass casualty triage skills using immersive three-dimensional virtual reality. *Academic Emergency Medicine: Official Journal of the Society for Academic Emergency Medicine*, 15(11):1160-1165, 2008.
[64] M. Virvou, C. Manos, G. Katsionis, and K. Tourtoglou. Incorporating the culture of virtual reality games into educational software via an authoring tool. In *2002 IEEE International Conference on Systems, Man and Cybernetics*, IEEE Computer Society Press, pages 326- 331, 2002.
[65] B. G. Witmer and M. J. Singer. Measuring Presence in Virtual Environments: A Presence Questionnaire. *Presence: Teleoperators & Virtual Environments*, 7(3):225-240, 1998.
[66] A. Yusoff. A Conceptual Framework for Serious Games and its Validation. PhD thesis. 2010.
[67] M. Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9):25-32, 2005.
[68] America's Army. Available: http://www.americasarmy.com/. [Accessed: 01-Nov-2011].
[69] Sealund. Available: http://www.sealund.com/. [Accessed: 01-Nov-2011].
[70] OLIVE - On-Line Interactive Virtual Environment. Available: http://www.saic.com/products/simulation/olive/. [Accessed: 02-Nov-2011].
[71] VBS2. Available: http://www.bisimulations.com/node/22/vbs2/rte. [Accessed: 02-Nov-2011].
[72] Thinking Worlds. Available: http://www.thinkingworlds.com/. [Accessed: 02-Nov-2011].
[73] Grand Theft Auto IV. Available: http://en.wikipedia.org/wiki/Grand_Theft_Auto_IV. [Accessed: 03-Nov-2011].
[74] Jess, the Rule Engine for the Java Platform. Available: http://jessrules.com/. [Accessed: 04-Nov-2011].

[75] Monkey Island (series). Available: http://en.wikipedia.org/wiki/Monkey_Island_(series). [Accessed: 06-Nov-2011].
[76] NeoAxis Engine. Available: http://www.neoaxisgroup.com/. [Accessed: 27-Dec-2008].
[77] OGRE. Available: http://www.ogre3d.org/. [Accessed: 07-Jan-2009].