Using a Task Modeling Formalism in the Design of Serious Games for Emergency Medical Procedures

Alberto Cabas Vidani and Luca Chittaro
HCI Lab, Dept. of Math and Computer Science
University of Udine
Udine, Italy
http://hcilab.uniud.it

***Abstract*—Lack of standard methodologies to guide and organize game design can result in longer and less predictable game production processes. Moreover, the need for interaction among domain experts (providing the instructional content) and game developers is a peculiar aspect of serious games that makes their development more difficult. This paper focuses on the design of games for procedural training, and proposes to adopt a task modeling technique (ConcurTaskTrees, CTT [15]) in the modeling of training scenarios. In particular, we show how CTT can be used to (i) analyze and structure pedagogical content about the procedures, (ii) support and monitor procedure execution in the game. We also describe how we employed CTT in the design of a serious game for training nurses in emergency medical procedures on disabled patients.**

Keywords-serious games; design; development; task modeling; emergency medical procedures.

I. INTRODUCTION

Several papers [1-7] describe the activities carried out to develop specific serious games. However, they do not report about specific methodologies and no methodological common ground emerges to guide and organize game design. In his recent introductory paper on serious games [8], Zyda describes the main challenges posed by serious games development. He stresses the importance of a research and development agenda from which serious as well as non-serious (entertainment) games could really benefit, allowing for shorter and more predictable production plans, reduced development time and a more effective educational or instructional effect of games. He claims that a cognitive approach to game design would be necessary to this end, i.e. the design process should take into account the player's internal information processing mechanisms, as well as the processes involved in perception. This would allow developers to model a more involving story and more believable characters as well as better understand players' behavior and requirements. In addition, it would help analyzing gameplay and integrate pedagogy with story, which is necessary in serious games. Another fundamental need of serious games development is the capability to automatically analyze and report the actions made by the player in the game and the consequences they had on the virtual environment. This feature would help assess users' learning achievements and also improve game quality (e.g., identify usability issues) by supporting offline analysis of player performance.

Unlike entertainment games, the development of a serious game requires a tight collaboration among domain experts (e.g, physicians for a medical game) and game designers and developers. This critical aspect of serious games design is highlighted by Kelly et al. [9]: 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 (the pedagogical content). To speed up serious games development and make it more reliable, one could benefit from specific tools 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.

Our general research goal is to define a framework to support serious games design, also allowing for partial automation through tools that could be used directly by domain experts. Given a *scenario* (i.e., an informal narrative describing events that should take place in the game, as well as locations, objects and characters involved in those events), the domain expert should be able to model it with a visual editor, producing a representation of it that could be automatically incorporated in the game.

In this paper, we concentrate on serious games for procedural training. Their main purpose is to teach the player the correct way to execute a procedure (e.g., casualty triage in a medical game or weapon shooting and reloading in a military game). In these games, the pedagogical content is part of the story itself, since some of the actions the player can perform belong to the procedure she has to learn. This paper focuses on two main problems: (i) finding a suitable modeling formalism for the analysis of the procedures and (ii) support and monitor procedure execution in the game through a formal representation which can be processed by a computer. In particular, we investigate the use of a task modeling technique - *ConcurTaskTrees* (hereinafter, *CTT*) [3], and

introduce a code module to support CTT in serious games and game building tools. More specifically, we developed (i) an XML parser which extracts information from a CTT model and exploits it to build a tree-like data structure that describes the possible paths that a scenario can take and the actions that compose them, (ii) the methods for accessing the data structure to support and monitor procedure execution in the game.

The paper is organized as follows. In Section II, we describe previous research that deals with serious game design and development. In particular, we consider proposals of general modeling and design techniques. Section III discusses our proposal for using CTT in the modeling of scenarios and in partial automation of game creation. Section IV presents the specific serious game in which we have employed the proposed techniques. Finally, Section V concludes the paper by illustrating future work.

II. Related Work

While serious games research has studied important topics such as realism and effectiveness of simulation (e.g., creating realistic smoke behavior for a firefighter training game [5]) or increasing immersion in the game (e.g. by developing specific props [10] or implementing advanced interaction with virtual objects [11]), the game production process is still "a handcrafted, labor-intensive effort" [8] which could take advantage of standard design methodologies.

Kelly et al. [9] highlight the need for a careful design of serious games aimed at effectively conveying educational content. They describe the development of a game to learn about the immune system of living organisms. An educational advisory board was formed, including immunologists, instructional designers, game developers and experts in educational technology. Then, the board identified the learning objectives and defined the game features (e.g., level structure, characters, etc.) that best supported those objectives. For example, game levels were structured following the organization of basic biological concepts. Although the paper accurately describes some important design choices, it does not try to generalize them to other serious games and to distill a method that could be followed by other projects.

Nadolski et al. [12] focus instead on proposing a methodology (called EMERGO) and a toolkit for educational game design. This toolkit is mainly aimed at creating a learning environment (based on a set of static pictures with a 2D graphics user interface) rather than to create a compelling 3D gaming experience as many serious games (including ours) aim to.

Moreno-Ger et al. [13] suggest some design guidelines for online educational games. They especially concentrate on assessment of player's performance and adaptation to player's abilities. They propose to model games as finite state machines (FSMs) in which the actions of the player trigger state transitions and the sequences of actions eventually lead to one or many end states. This kind of model seems to suit those games where players have to go through a series of challenges (like quizzes) in a predefined order. For more complex games, FSMs can be limited and richer notations could prove more adequate to model them.

A slightly improved approach based on an augmented FSM is proposed by Ponder et al. [14]. Each state of the augmented FSM, called *scenario step*, contains one or more *decision sets*. Each decision set is connected to another scenario step. If a decision set contains more than one decision, all the decisions in the set must be taken by the player to move to the next scenario step. Although this model is richer than the one used in [13], it still lacks some important features such as temporal operators to describe the various possible temporal dependencies among decisions in a scenario.

It must also be noted that most [9,13,14] of the above mentioned approaches do not propose tools to automatically create parts of the game from the design specifications they produce.

III. Our Proposal

Task models are used in human-computer interaction (HCI) to analyze the logical activities that support the achievement of users' goals [21]. 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, 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 believe that task models can be used to structure and describe these tasks. Moreover, task models can be represented by a proper data structure, to be processed by the game engine or game building tools. Task models can thus support the game production process in different ways:

- When the design process requires to define a scenario or multiple scenarios (which might illustrate single stages of the game or the whole story), 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 pedagogical content in terms of task models.
- At runtime, the above mentioned data structure 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 procedures 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 learning progress of single players or groups of players.

A. *The CTT Formalism*

CTT is a formal notation originally proposed by Paternò et al [15] in the domain of interface design and development. It has been used to design and develop hypermedia [16] and safety critical applications [17], to support usability evaluation [18], and to analyze user interactions with web applications [19].

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;
- Concurrent notation. Tasks are linked with their siblings in the tree through temporal operators.

A CTT model is built in three phases:

- tasks are decomposed and arranged in a hierarchical tree-like structure;
- temporal relations among tasks are identified;
- objects associated with each task and actions which allow objects to communicate with each other are identified.

Different temporal operators allow designers 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 don't need to have the same duration, so they can overlap;
- Synchronization (T1 |[]| T2). Tasks T1 and T2 have to synchronize on some actions to exchange 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.

As explained in [15], some combinations of temporal operators can lead to ambiguity in the interpretation of a CTT model. For example, the tree in Fig. 1 can be interpreted both as (T1 [] T2) ||| T3 and T1 [] (T2 ||| T3). Ambiguity has to be solved either by introducing a priority order among temporal operators or by introducing an additional node to disambiguate, as shown in Fig. 2, whose interpretation is unique and corresponds to T1 [] (T2 ||| T3).

Creation and editing of CTT models can be performed using the freely available, cross-platform visual tool CTTE (ConcurTaskTrees Environment) [20].

B. *Motivations for Choosing CTT*

Several features of CTT make it suitable to model training procedures in serious games.

First, as a task modeling formalism, it is possible to employ CTT for the purposes we described at the beginning of this section.

Second, CTT, as some other task model formalisms, supports hierarchical modeling. This can make it easier to understand models, also when they are shared among different persons or groups, and also allows for reuse of models or parts of them. The low percentage of reuse of code modules is a problem in current game development [8] and reuse of task models would help speeding up game design in situations where, for example, several training scenarios that share common procedures have to be modeled. We believe that the hierarchical structure can also help when complex scenarios that contain many paths are used. In this case, the designer could first analyze the scenario at a more abstract level, creating a tree with a limited number of nodes to roughly describe the procedures. Then, she could deepen the analysis by attaching subtrees to the nodes that need a more detailed description. This kind of reasoning can be iterated in a recursive way, until all the paths have been analyzed.

Third, the graphical syntax (see Figures 1 through 5 for examples) of CTT can make modeling more practical, because it can be easier to interpret and to edit than a text-based syntax [21].

Fourth, CTT allows one to model concurrent processes, a feature not easily found in other task modeling formalisms [21]. This can be useful in serious games design, because it allows one to model interactions among several players in multiplayer games or among computer-controlled characters and players.

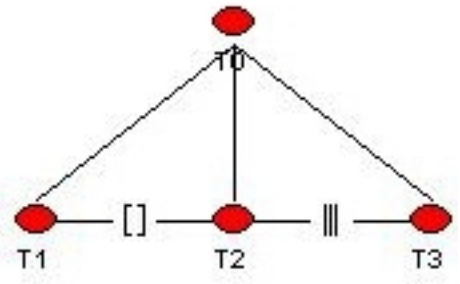

Figure 1 An example of the ambiguity problem.

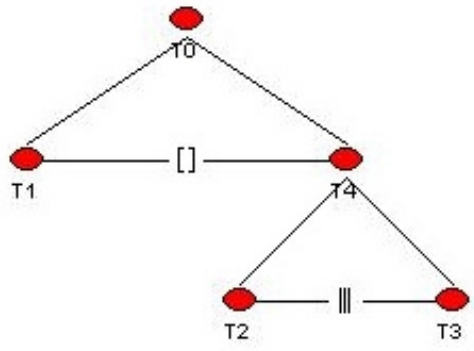

Figure 2 Eliminating ambiguity by inserting a new node.

Fifth, among task modeling techniques, the choice of temporal operators of CTT is the largest available [21], providing more expressive power.

Finally, the availability of the above mentioned CTTE tool is particularly important. Besides supporting a relatively easy and quick editing of CTT models, CTTE provides some additional features that can prove useful in our case. The first one is the *Informal to formal* description tool. It allows to load a natural language description of a scenario and interact with the tool to extract task descriptions from it. This way, users of CTTE can create a task list from a scenario and then arrange the tasks in a CTT tree-like structure. Scenarios are easily understood by people who have no experience in HCI or software engineering. Therefore, they offer a good way to start designing new games with experts of different areas, as it often happens when developing serious games. The second useful feature of CTTE is the XML export of CTT models. This makes the exploitation of task models by other applications easier: a developer can create a parser for CTT XML syntax that extracts the needed information from the exported model.

C. Integration of CTT in the Game Development Process

To exploit CTT for the purposes described at the beginning of this section, we first needed to make CTT models accessible by any application. We thus defined a suitable data structure that can be automatically built (using a parser we developed) from CTT models in XML format. The proposed data structure has been implemented in C# and is a tree-like recursive structure, with nodes representing tasks. Leaf nodes in the tree represent the actions players can perform in the game, while internal nodes only serve the purpose of grouping their child nodes. This means that the actions described in the game scenario will be represented by the leaf nodes of the CTT model, while internal nodes will be used by designers to group player's actions into a logical structure. In the following, we describe:

- the basic properties and methods to access the data structure,
- a method which simulates the dynamic behavior of the model represented by the data structure,
- a difficulty property to support different levels of game difficulty with the data structure.

1) Accessing the Data Structure

Each node in the structure has a *Name* and a *TemporalOperator* property as in the CTT formalism. Currently, the data structure only supports the enabling, choice and order independence temporal operators. The following node properties support access to the structure:

- an array of references to child nodes;
- references to the closest right and left siblings (i.e., the ones directly connected to the current node by a temporal operator, if any);
- a reference to the parent node.

Direct access to nodes is provided by a method that searches a node given its name. Otherwise, nodes can be accessed starting from the root node, which is exposed by a property, and recursively visiting the tree.

2) Simulating the Dynamic Behavior of the Model

Since we wanted to support and monitor procedure execution through the CTT model, we needed a way to simulate its dynamic behavior. In particular, we needed a way to let the game engine, at any time, know the list of enabled tasks according to the constraints specified in the task model. For example, if a task T1 is connected to its left sibling (task T2) by the >> operator, it means that T1 cannot be executed before T2 (i.e., T1 is disabled until T2 is executed). Or, if two nodes are connected by the |=| operator, the corresponding tasks can be enabled at the same time.

The list of enabled tasks contains only tasks corresponding to leaf nodes. At the beginning of the simulation, the list only includes tasks that do not have neither enabling siblings nor ancestors with enabling siblings. A step of the simulation consists in updating the list as if one of the enabled tasks had been executed. In our implementation, the update is carried out by a method which receives the name of a task as a parameter. If the task is not enabled, no update is performed, otherwise the method performs the following operations:

- removes the task from the list of enabled tasks;
- inspects the hierarchy to check if the task was preventing other tasks from becoming active;
- if such tasks are found, adds them to the list of enabled tasks.

Simulation of the dynamic behavior of the model provides support to and monitoring of procedure execution in the game. For example, a game engine could retrieve task names for enabled tasks and show them to the player. Then, the player could select one of them, triggering the execution of a simulation step. As a result, some event will happen in the game (depending on the choices of the game designer) and the task list proposed to the player will be updated.

3) Supporting Different Difficulty Levels

A typical player's challenge in games for procedural training is to choose the right task to perform at the right time among the various tasks into which the procedure is organized. Using the above described method for simulation of the dynamic behavior of the model, a game can only allow the player to execute tasks which satisfy the constraints imposed by the temporal operators in the CTT model. In this sense, task selection would not be a challenge to players because any of the proposed choices would be correct.

Allowing for possibly wrong choices would make the game more challenging and also give more learning opportunities to the player. We thus allow to include in possible player's choices also tasks that do not satisfy the temporal constraints. The number of proposed tasks is determined according to:

- *difficulty level* of the game, identified by an integer ranging from 0 (easiest) to a given level k (hardest), which is decided at design time and corresponds to the maximum number of difficulty levels of the game.
- *choice difficulty* of the nodes, i.e. an integer value (greater than 0), which must be assigned at design time to the parents of leaf nodes (for example, there are 7 such parents in the structure shown in Fig. 3). Firstly, the game designer has to consider nodes which only have leaf children, and she has to assign a value to the choice difficulty of each of those nodes based on how difficult she deems it

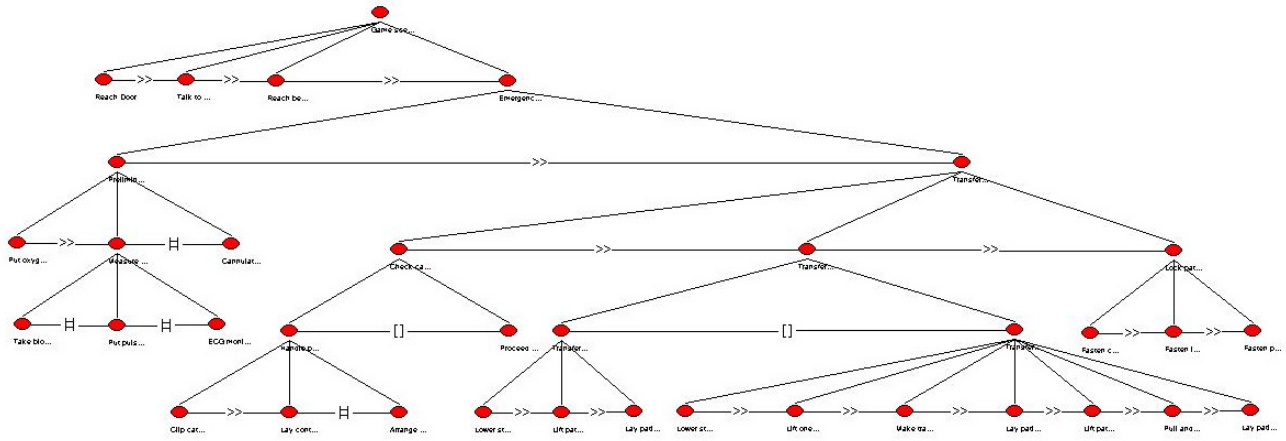

Figure 3 The structure of a complete model for one full medical procedure of our game

will be for the player to choose a task among the children of the considered node. Then, she has to consider nodes with both leaf and non-leaf child nodes: for each such node, the choice difficulty is the maximum among choice difficulties of its non-leaf children and the value of the difficulty of choosing among its leaf children. The difficulty value for nodes with no leaf children is derived automatically and is equal to the maximum among choice difficulty values of their child nodes.

The game difficulty level and choice difficulty of the nodes are used during simulation of the dynamic behavior of the model to determine what tasks can be proposed to the user. When game difficulty is lowest (0), only tasks that satisfy temporal operators are proposed to the player (i.e., the simulation works as described above). As the difficulty level increases, the number of considered tasks increases as well, until all the tasks belonging to the procedure are proposed when the difficulty becomes highest (k). To determine what tasks can be proposed, at each simulation step, when the list of enabled tasks is updated, the choice difficulty value of the parents of non-leaf nodes is compared to the game difficulty level:

- if the choice difficulty of the parent is greater than the game difficulty level, only correct tasks in its subtree are proposed;
- if the choice difficulty of the parent is less or equal than the game difficulty level, then all the tasks in the subtree are proposed to the player (this includes the possibly wrong tasks).

IV. Case Study: Emergency Medical Procedures on Disabled Patients

We have employed CTT in the design of a medical serious game, using the previously described data structure to model the game scenario. The target players of the game are nurses who work in ambulance services, and the purpose of the game is to train them in some emergency medical procedures concerning disabled persons. It allows nurses to explore a 3D virtual environment, where they can choose actions to respond to the medical emergency at hand.

A. Motivations

There are several reasons that motivate our focus on disabled patients. First, emergencies concerning the disabled are often more difficult to handle with respect to non-disabled persons. For example, deaf persons could not hear or understand emergency nurses' words, cognitive disabled persons could not be able to communicate correctly what they feel, motor disabled persons could not move (and feel pain in) some parts of their bodies, rescuing a blind person could require nurses to deal properly with a guide dog, and so on. Moreover, some disabled person could already have attached medical devices (e.g. catheters) the nurse should deal with. Some disabled persons could be using devices (e.g. a wheelchair) that are not taken into account by procedures for the non-disabled (which are the ones carried out most frequently in the nurses' regular practice). As a consequence, medical emergency procedures must often be tailored to the specific needs of disabled persons. However, since ambulance runs concern disabled persons much more rarely than non-disabled ones, and there are many kinds of disabilities, exposure of a nurse to real cases for all disabilities is unlikely. For these reasons, specific training could make nurses more familiar with and more effective in responding to these rarer situations.

Second, although serious games research has already focused on training applications for medical first responders (see for example [22], [11] and [4]), no specific work concerns emergency procedures for disabled persons.

Third, a game, especially if in the form of a 3D virtual environment (as the one we are developing), can generally create an immersive experience, which helps players in familiarizing with infrequent situations. Some training techniques currently employed for emergency nurses are based on live simulations, with actors acting as patients. These simulations require time, money and organizational effort, each time they are carried out. On the contrary, a serious game, once the development costs are sustained, is always available and can be used anywhere, including at home. Moreover, live simulations usually focus on the most frequently needed procedures which are aimed at the non-disabled. So, the game could be a more practical means to experiment with the considered scenarios.

B. Technical Details of the Implementation

We developed the game using NeoAxis [23], a game engine based on the Ogre rendering engine [24]. Neoaxis provides a set of tools for resource editing (3D models, level maps, graphical interfaces, …), basic AI algorithms

for character movement, physics simulation, cut scene and controller management. Since the game engine is developed in C#, we were able to easily integrate the previously discussed data structure in the game, and directly it access it from the game code.

C. Application of CTT to Game Scenario Modeling

The game has three main training objectives:

- given an emergency medical procedure organized into tasks, teach the trainee what tasks to perform, in what order and timing, under which conditions, to correctly carry out the procedure;
- show the trainee how the tasks are performed, through realistic animations of game characters and objects;
- allow the trainee to experience and familiarize with infrequent emergency situations.

To teach a procedure, the game proceeds as follows. During the game, players can explore the virtual environment and choose what task to perform among a list of possible ones. The selection of a task by the player triggers specific events in the game, for example the execution of some animation (e.g., the nurse character performing an action) or some visible effect on the patient (e.g., turning pale). The list is populated with tasks taken from the CTT model. CTT models are produced during the design phase, starting from scenarios written together with an emergency medicine expert, describing the details of an emergency situation and of the medical emergency procedure to be performed in that situation.

The scenario we consider as an example in this paper concerns a tetraplegic person with high fever and respiratory difficulties in his bed at home, assisted by relatives. 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 and immobilized. The scenario ends with nurses moving the patient out of his home.

From the scenario, we created the CTT model shown in Fig. 3. All leaf nodes in the tree are tasks that players can perform at some time in the game. There are two kinds of leaf nodes: (i) nodes which correspond to medical tasks (e.g. cannulating a vein) or (ii) nodes which correspond to generic actions, not strictly part of the medical procedure, but essential to the progress of the game (e.g., reaching the patient's bed after entering the house).

As an example of how the CTT model in Fig.3 is used, we consider the subtree rooted at the node labeled *Preliminary operations* (Fig. 4). The three children of this

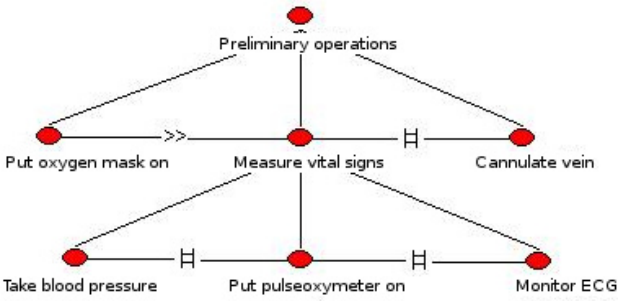

Figure 4 Subtree modeling preliminary operations.

node are two leaves and an internal node, which, in turn, has three leaf children). The first temporal operator on the left is an enabling operator, while the others are all order independence operators. This subtree specifies that the preliminary operations include putting the oxygen mask on the patient, measuring vital signs and cannulating the patient's vein. In turn, measuring vital signs includes taking blood pressure, putting the pulseoxymeter on the patient and monitor ECG. The first task to be performed is to put the oxygen mask on, then the other tasks can be carried out in any order.

The subtree rooted in the node labeled *Transfer to stretcher* (Fig. 5) is an example of how choices and alternatives are modeled. When the procedure reaches this subtree, the player must transfer the patient to the stretcher, to subsequently move him out of his home. This can be done in two ways, depending on the means employed to lift the patient up: the bed sheets on which the patient lies or a specific transfer sheet that is usually available in the ambulance. As explained in section III, nodes corresponding to the two alternatives (*Transfer using bed sheet* or *Transfer using transfer sheet*) are grouped under a single node (*Transfer to stretcher*, in our case) to avoid ambiguities. For each alternative, a corresponding subtree groups the tasks to be performed.

D. In-Game Use of the Model

As described in the previous subsection, to perform a task belonging to the emergency medical procedure, players have to select it from a list. We included in the graphical user interface of the game a *task selection panel* showing a list of tasks loaded at runtime from the CTT model. Players double click a task to perform it (Fig.6).

When the difficulty value is more than 0, the list can also include tasks which do not satisfy constraints imposed by the temporal operators. When players select a wrong task the error is recorded by the game for further review.

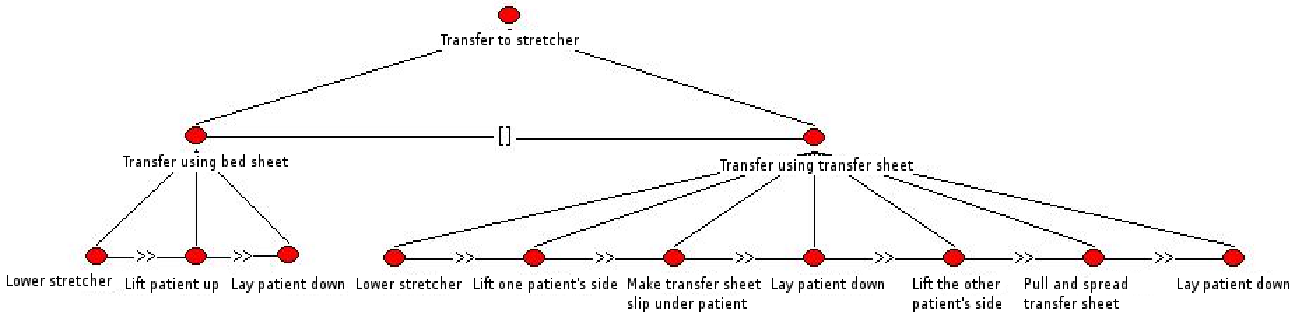

Figure 5 Subtree modeling a choice task.

Figure 6 A screenshot of the game with the task selection panel on the right.

Figure 7 A screenshot of the game with the panel for selection of alternatives.

The task selection panel generally shows only tasks corresponding to leaf nodes. It shows internal nodes only when they have child nodes connected by a choice operator. When the player double clicks on the name of one of the parent nodes a new panel opens, showing the possible alternatives (the child nodes) as in Fig. 7. After the player has selected an alternative, game execution continues as usual. For example, in the case depicted in Fig. 3, when task labeled *Transfer to stretcher* is in the task selection panel and gets double clicked, a new panel pops up. It shows the list of possible alternatives, which, in this case, are *Transfer using bed sheet* and *Transfer using transfer sheet*.

V. Conclusions and Future Work

In this section, we summarize some of the limitations of our current work and how we intend to extend the current approach.

First, at this stage of the work we have focused on single-player games and have not implemented the concurrency part of CTT. When we will include the concurrent part, we will be able to handle multiplayer games, e.g. supporting different task selection panels for each player and including tasks requiring collaboration among players. Moreover, we could use concurrency to model the behavior of autonomous characters interacting with players. The multiple characters we currently support are not autonomous: their pre-defined behavior can be modeled only as a consequence of tasks chosen by the player.

Second, an interesting extension we are investigating is the integration of CTT with error modeling techniques. More precisely, our goal is to incorporate explicit error modeling in the CTT-based scenario modeling to:

- make the game capable of analyzing players' errors (e.g. gathering more detailed and precise statistics on player performance),
- allow the game to propose erroneous actions in the task list that is presented to players by exploiting the association between tasks and errors described in the error model.

SHERPA (Systematic Human Error Reduction and Prediction Approach [25]) is the specific error modeling technique we are considering. SHERPA has been defined to model errors users could commit while interacting with a system. After carrying out a task analysis of normal activities, SHERPA requires to associate each task to a task type among the five it defines (e.g., the task of checking if a catheter is present would be classified as a checking task). SHERPA task types associate then a possible error mode to each task. Finally, consequences of the error, possibility for recovery, probability of the error, its criticality and potential remedies are determined by designers.

Third, the way we have developed scenarios so far has been to use CTT ourselves during and after expert interviews. Another important step would be to evaluate how successfully the experts could use it on their own. The simple notion of game difficulty we proposed will also need to be tested with experts to assess how easy it is for them to provide the required numbers and how effective the resulting difficulty levels could be. Finally, we plan to carry out user evaluations of the game with nurses.

Acknowledgment

We are particularly indebted to dr. Elio Carchietti (118 Emergency Medical Services, Udine Hospital, Italy) for sharing his experience with us.

Our research is partially supported by the Friuli Venezia Giulia region under the project "Servizi avanzati per il soccorso sanitario al disabile basati su tecnologie ICT innovative" ("Advanced emergency medical services for the disabled based on innovative ICT technologies").

References

[1] A. Sliney and D. Murphy, "JDoc: a serious game for medical learning," Advances in Computer-Human Interaction, 2008 First International Conference on, 2008, pp. 131-136.

[2] R.L. Sanders and G.S. Rhodes, "A simulation learning approach to training first responders for radiological emergencies," Proceedings of the 2007 summer computer simulation conference, Society for Computer Simulation International, 2007, pp. 1-3.

[3] R.L. Sanders and J.E. Lake, "Training first responders to nuclear facilities using 3-D visualization technology," Proceedings of the 37th conference on Winter simulation, Winter Simulation Conference, 2005, pp. 914-918.

[4] D. McGrath and D. Hill, "UnrealTriage: a game-based simulation for emergency response," The Huntsville simulation conference, 2004.

[5] T.U.S. Julien and C.D. Shaw, “Firefighter command training virtual environment,” Proceedings of the 2003 conference on Diversity in computing, ACM, 2003, pp. 30-33.

[6] S. Jain and C.R. McLean, “A concept prototype for integrated gaming and simulation for incident management,” Proceedings of the 38th conference on Winter simulation, Winter Simulation Conference, 2006, pp. 493-500.

[7] M. Hogan, H. Sabri, and B. Kapralos, “Interactive community simulation environment for community health nursing,” Proceedings of the 2007 conference on Future Play, ACM, 2007, pp. 237-240.

[8] M. Zyda, “From visual simulation to virtual reality to games,” Computer, vol. 38, 2005, pp. 25-32.

[9] H. Kelly, K. Howell, E. Glinert, L. Holding, C. Swain, A. Burrowbridge, and M. Roper, “How to build serious games,” Commun. ACM, vol. 50, 2007, pp. 44-49.

[10] P. Backlund, H. Engstrom, C. Hammar, M. Johannesson, and M. Lebram, “Sidh - a game based firefighter training simulation,” Proceedings of the 11th International Conference Information Visualization, IEEE Computer Society, 2007, pp. 899-907.

[11] S. Stansfield, D. Shawver, A. Sobel, M. Prasad, and L. Tapia, “Design and implementation of a virtual reality system and its application to training medical first responders,” Presence: Teleoper. Virtual Environ., vol. 9, 2000, pp. 524-556.

[12] R. Nadolski, H. Hummel, H. Van den Brink, R. Hoefakker, A. Slootmaker, H. Kurvers, and J. Storm, “EMERGO: methodology and toolkit for efficient development of serious games in higher education,” Simulation & Gaming, 2008.

[13] P. Moreno-Ger, D. Burgos, I. Martínez-Ortiz, J.L. Sierra, and B. Fernández-Manjón, “Educational game design for online education,” Comput. Hum. Behav., vol. 24, 2008, pp. 2530-2540.

[14] M. Ponder, B. Herbelin, T. Molet, S. Schertenlieb, B. Ulicny, G. Papagiannakis, N. Magnenat-Thalmann, and D. Thalmann, “Immersive VR decision training: telling interactive stories featuring advanced virtual human simulation technologies,” Proceedings of the workshop on Virtual environments 2003, ACM, 2003, pp. 97-106.

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

[16] F. Paternò and C. Mancini, “Engineering the design of usable hypermedia,” Empirical Softw. Engg., vol. 4, 1999, pp. 11-42.

[17] F. Paternò, C. Santoro, and S. Tahmassebi, “Formal models for cooperative tasks: concepts and an application for en-route air traffic control,” Proceedings of DSV-IS '98, 1998, pp. 71-86.

[18] A. Lecerof and F. Paternò, “Automatic support for usability evaluation,” IEEE Trans. Softw. Eng., vol. 24, 1998, pp. 863-888.

[19] L. Paganelli and F. Paternò, “Intelligent analysis of user interactions with web applications,” Proceedings of the 7th international conference on Intelligent user interfaces, ACM, 2002, pp. 111-118.

[20] G. Mori, F. Paternò, and C. Santoro, “CTTE: support for developing and analyzing task models for interactive system design,” IEEE Trans. Softw. Eng, vol. 28, 2002, pp. 797-813.

[21] F. Paternò, “ConcurTaskTrees: an engineered approach to model-based design of interactive systems,” The Handbook of Analysis for Human-Computer Interaction, Lawrence Erlbaum Associates, 2002, pp. 483-500.

[22] P. Kizakevich, R. Furberg, R. Hubal, and G. Frank, “Virtual reality simulation for multicasualty triage training,” Proceedings of the 2006 I/ITSEC Conference, 2006.

[23] NeoAxis Engine, www.neoaxisgroup.com

[24] Ogre 3D graphics engine, www.ogre3d.org

[25] D.E. Embrey, “SHERPA: a systematic human error reduction and prediction approach,” Contemporary Ergonomics: Proceedings of the Ergonomics Society's Annual Conference, Taylor & Francis, 1987.