HIGH-LEVEL SCENARIO EDITING FOR SERIOUS GAMES

Casper van Est, Ronald Poelman and Rafael Bidarra
Serious Gaming Centre, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands

Keywords: Scenario editing, Simulation games, Serious games.

Abstract: Although simulation games provide a competetive and safe alternative to real-life training sessions, the flexibility of adjusting such training sessions to fit the needs of individual trainees is relatively low. The reason for this is that these games are often delivered as a static product with predefined scenarios that cannot easily be edited by an instructor. This paper proposes a solution to this dilemma in the form of scenario editing, which allows instructors to define and edit scenarios, using high-level actions and events and some basic logic. A prototype scenario editing application was developed and subsequently evaluated, featuring a graph-metaphor for easily editing scenarios and an interface that allows real-time editing. The evaluation shows that the chosen approach is applicable and a good starting point for further development.

1 INTRODUCTION

When video games originally came into existence, their purpose was solely to entertain. Nowadays, with video games becoming more accepted by the mainstream, and with more scientific research being done in this area, a subset of video games called serious games is being used for business and educational purposes as well (Smith 2007). One such application of a so called serious game is as a replacement of professional training sessions, where they are being used to educate and train safety supervisors, medical professionals, police officers, and other professionals. The advantages of using such a simulation game to replace real-life training sessions are numerous; simulation games arenot expensive, safer, less time consuming and can potentially offer better learning (Susi, Johannesson et al. 2007).

However, simulation games are not being used to their full potential yet. One important issue that simulation games are currently facing is that the flexibility of adjusting the virtual training session to an individual trainee's needs is relatively low, compared to a real-life training session.

The fundamental problem with the current development approach of simulation games is that a simulation game is handed to the instructor as a finalised product. In optimal conditions, the instructor is indirectly involved in the process of making the game, by defining the training program, but once the development of the game has finished, no additional changes can be made to the game, or to its training program. Some simulation games do have options for adjustability. However, these options are still very limited and restrictive in nature. With these games, the game developer has (perhaps in consultation with an instructor) prepared a few options for the instructor, which the instructor can use to alter the gameplay. While this allows to instructor to exercise some control over the game's scenario, the instructor can only adjust these predefined settings. Thus, the game is still delivered as a finalised, static product.

The approach described in this paper aims at making the training session more adaptable to the individual trainee's needs, by providing the instructor with a Scenario Editor. In this case, the game developer delivers not one, but two products to the instructor; the simulation game and an extensive collection of scenario building blocks (Van Est, 2010). Then, a separate application called the scenario editor can be used to arrange the scenario building blocks according to a training program, and combine them with the simulation game to create an individual training experience, specific to a certain trainee. Using the feedback from the training session, adjustments can then be made to the training program, by re-arranging the scenario building blocks. This way, another individual training experience can be created using the same scenario building blocks and simulation game. With a large

van Est C., Poelman R. and Bidarra R..
HIGH-LEVEL SCENARIO EDITING FOR SERIOUS GAMES.
DOI: 10.5220/0003374503390346
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2011), pages 339-346
ISBN: 978-989-8425-45-4

collection of building blocks, the variations in arrangements that can be made are endless. Thus, the flexibility of adjusting a training scenario is returned to the instructor. In the following sections, this paper will discuss this approach in more detail, and an application example will be presented.

2 PREVIOUS WORK

This section discusses current authoring methods. These methods can be divided into two categories; environment authoring and causality authoring. In environment authoring, the editing environment is similar to the game world; it provides the user with a view of the environment which is comparable to the world as it is presented to the player. In causality authoring, the instructor operates in a distinctly different environment than the game world, such as an abstract graph-based editor.

In environment authoring, the instructor can directly influence the game's environment. The instructor can place assets (objects, characters, triggers, markers, etc) and move them around. By placing a number of these assets, and assigning certain properties to them, the instructor can influence the course of the scenario. A real world example of environment authoring is a child playing in a sandbox: he builds an environment with perhaps buildings or foliage, places a handful of characters and then 'runs' the scenario. Environment authoring offers the instructor direct control over specific assets in a game environment. Thus, it offers the instructor great and precise power. However, it also requires the instructor to directly influence the game world, thus requiring the instructor to have a decent amount of knowledge about the game world. The instructor is required to know about game development concepts such as placing and moving objects around in a virtual 3D world, using triggers, materials, etc.

Examples of authoring applications that use environment editing are UnrealEd, the level editor of the Unreal 3 engine; and e-Adventure (Moreno-Ger, Martinez-Ortiz et al. 2005) (Moreno-Ger, Blesius et al. 2007). UnrealEd was developed by and for professional game developers, and is as such very powerful, but also very complex. When using e-Adventure, on the other hand, creating a game in e-Adventure is made easier for the instructor by allowing him to author and execute a game without any background in programming. The instructor can author game scenarios and add content to them, such as objects, characters and conversations. The authoring application focuses on supporting those tasks that are specific to the educational domain. Among these are assessment and adaptation: the need to track and evaluate the activity of the trainee and the need to adapt the behaviour of the game to fit different ranges of trainees, respectively. A noteworthy feature of e-Adventure is the possibility to link to other sources of information, to be accessible during the game.

The functionality offered by e-Adventure is too limited to be suited for professional game developers; only one type of game can be created, that game has to follow certain specific guidelines and there are little options for customizing the game. At the same time, the actions required to create a game using e-Adventure are too detailed to be suited for non-professional game developers. The user has to concern himself with technical issues such as foreground masks, layers, inventory item icons, etc. Aside from causing the creation process of a game to take an unnecessarily long time, these options are overwhelming to a didactic expert with no game development experience.

The second authoring method is called causality authoring. This method lets the instructor edit the causality processes of a scenario, usually by presenting a graph metaphor. Using this authoring method, authors can specify causalities such as 'when the user opens that box, he will receive this object'. Editing a graph is easier than editing a game environment, since it requires less technical knowledge of the author.

Examples of authoring applications that use causality authoring are Unreal 3's Kismet editor, Scribe (Medler and Magerko, 2006), Façade (Mateas and Stern, 2000), (Mateas and Stern, 2003), Scenejo (Weiss, Muller et al., 2005), (Spierling, Weiss et al., 2006), Art-E-Fact (Iurgel, 2004) and SAVEace (Holm, Stauder et al., 2002). For a discussion on the strengths and weaknesses of these applications, see (Van Est, 2010).

Limitations in current authoring methods provided by game development tools are found to be

- Authoring requires knowledge of gaming concepts
- Authoring requires too much work
- Authoring systems are designed non-generically
- Authoring systems offer unfriendly user interfaces
- Graphs can become too complex

One disadvantage of all current authoring methods is that none of them offers a generic solution; no standalone tool exists that allows scenario authoring to function with any other game development tool.

This would improve the effectiveness of such a tool, as it can then be applied in multiple projects.

The graph metaphor used in several high level scenario authoring applications seems to be a good fit, since it corresponds well with the user's concept of a scenario. However, the interface that is usually provided can be very complex, especially for non technical users. An authoring application would benefit from using more graphical metaphors, such as icons for pre- and post conditions, as can be found in tools aimed at children. The main issue with current authoring methods for the use in simulation games is that they have not been explicitly designed for use by a field expert or instructor. These experts typically have little knowledge of the technical concepts required in current authoring methods. For maximum usability, a scenario authoring application should offer functionality that allows people with limited experience to easily create or edit a scenario. Of course, since the scenario editor also needs to be powerful and support creativity, the tool should offer considerable depth, allowing the user to create complex scenarios as well. Perhaps a separation between basic and advanced features could offer some improvements to the usability.

3 BASIC APPROACH

The goal of the approach proposed in this paper is to give an instructor more control over the scenario of a simulation game. Basically, this can be achieved in three steps:

1. Represent the scenario of a simulation game at a more abstract level of scripting.
2. Make the abstracted scenario editable by an instructor.
3. Communicate the adjustments made by an instructor to a simulation game.

By seamlessly supporting these steps, one can enable instructors to make adjustments to abstracted scenarios, and subsequently communicate these adjustments to a simulation game.

3.1 Abstraction

First, two levels of scripting were identified. The first, lowest, level is called the programming level. The scripts in this level deal with low level concepts, such as objects, vectors, math functions, etc. The language used to write a script in this programming level could be any programming language such as C++ or UnrealScript.

The second level of scripting is called the gameplay scripting level, in which the scripts deal with the objects in a game level. The language used in the gameplay scripting level is easier to use than programming languages and could for instance be UnrealEd's visual scripting language Kismet (UnrealEd, 2011).

These two scripting levels, programming and gameplay scripting, are commonly used to develop games. In current game development teams, programmers operate in the lowest scripting level and write all kinds of scripts on how the engine should simulate the playing world. Then, gameplay scripters define what can be called the game's behaviour, e.g. how it interacts with the player, in the gameplay scripting level, as in Fig. 1.

What we propose in this approach is to add a third level of scripting above the previously mentioned levels, in which the global scenario of a game is scripted. We call this third level the scenario scripting level, and it deals with the scenario of a game. This high level of scripting is useful because it allows the instructor to focus on the scenario itself, without worrying about unnecessary programming or gameplay issues. For example, an instructor does not want to be bothered by issues such as which truck model should be used, how it moves or which colour it has. All these issues are dealt with in lower scripting levels, and are defined by one of the game developers.

Achieving this higher abstracted scripting level can be done by taking the same abstraction step that is taken from the programming level to the gameplay level, but now by applying it from the gameplay level to the scenario level, as in Fig. 2. In our approach, we've chosen to let the game developer decide on how to abstract the contents of the gameplay scripting level to the level of scenario scripting, just as the programmer decides how to abstract his code into the gameplay scripting level. In this way, the game developer is responsible for creating the content that can be used by the instructor. In that sense, it is up to the game developer to choose what information, or meta-data, is supplied to the instructor, and thus at which level of abstraction the instructor operates.
Therefore, there is no exact, strict, definition of the boundary between the scenario scripting level and the gameplay scripting level, and it can be precisely defined on a case-by-case basis.

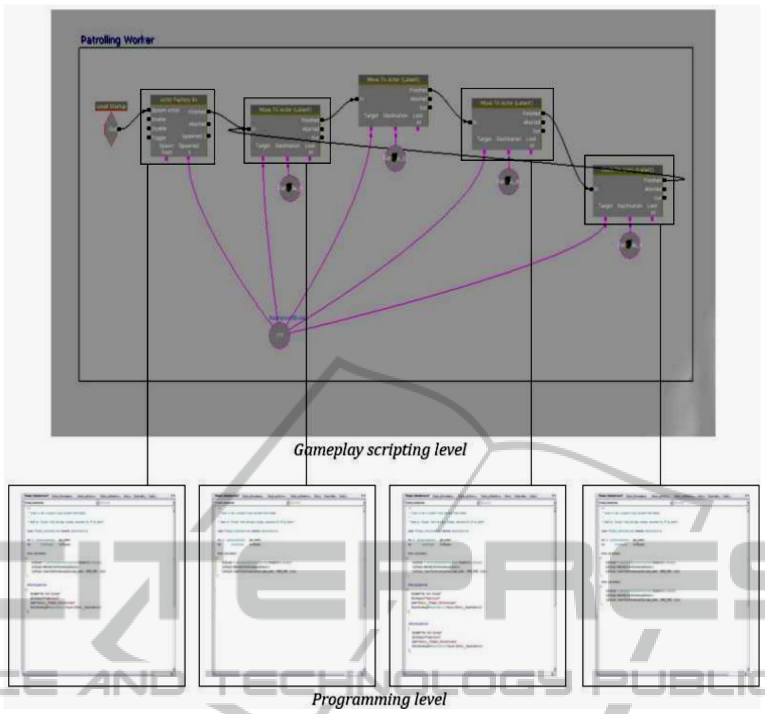


Figure 1: Nodes at the gameplay scripting level are abstractions of scripts at the programming level.

3.2 Editing

The editing actions of the scenario scripting language are called building blocks, which the instructor should have at his disposal in the scenario scripting level. Above, we showed that sections of gameplay script can together form scenario building blocks. But which forms can such a building block take? We start by identifying two entities that are the most basic building blocks of a scenario script; actions and events. An action is something that is performed by the player, while an event is something that is performed by the game. Together, these two building blocks allow an instructor to script a basic scenario. However, with just these two entities, the instructor is fairly limited in his expression of a scenario, as he can only create linear scenarios. In order to create non-linear scenarios, the instructor also needs to be able to apply some logic at the level of scenario scripting, such as if-then statements or other flow control strategies.

Additionally, in some types of games the instructor might want more detailed control over a scenario by using variables. At the level of scenario scripting, a variable can be used for logical decisions. The use of variables widens the options an instructor has for creating non-linearity in his scripts. Whereas without using variables, the instructor can only base the logic in his script on whether an action has been performed, with the use of variables the instructor can also write scripts that base their logic on how well an action was performed. Finally, a variety of settings have been introduced, including the scenario relevant properties of a game, e.g. the number of pedestrians in a driving simulation.

3.3 Interfacing

The arrangement and properties of the building blocks defined above can now be applied in a simulation game. The actual process of scripting a scenario can potentially happen in two different contexts and stages; within a game development environment, as is the case in gameplay scripting , or outside of the game development environment, by utilising a standalone scenario authoring application. Our approach is based on offering a standalone authoring application. That way, the process of scripting a scenario is independent of the specific game in which the scenario will be used, so the instructor can learn how to write scripts for one game and apply this knowledge to other games as well.

Considering that scenario building blocks are created from sections of gameplay scripts, we use an event-based communication between the scenario

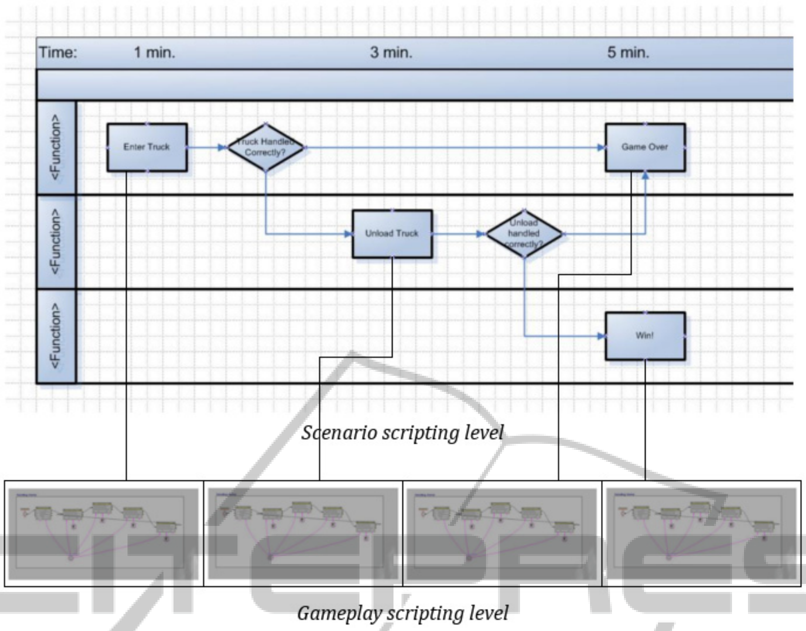


Figure 2: Nodes at the scenario scripting level are abstractions of scripts at the gameplay scripting level.

authoring application and the simulation game. This system sends messages back and forth when a building block, in the form of an action or event, needs to be executed. The game itself is then responsible for handling all the scripts at the gameplay scripting and the programming level, while the scenario authoring application is responsible for handling the scripts at the scenario scripting level, including the handling of scenario logic. By sending messages between the editor and the game, all communication occurs in real-time. This allows the instructor to make adjustments to the scenario, as long as these changes do not corrupt the scenario.

In conclusion, our approach allows instructors to exercise control over a scenario by interactively editing a visual script at the abstracted level of scenario scripting, using the language of scenario building blocks. Furthermore, additional control is given to instructors by allowing them to make real-time adjustments to a scenario as it is running.

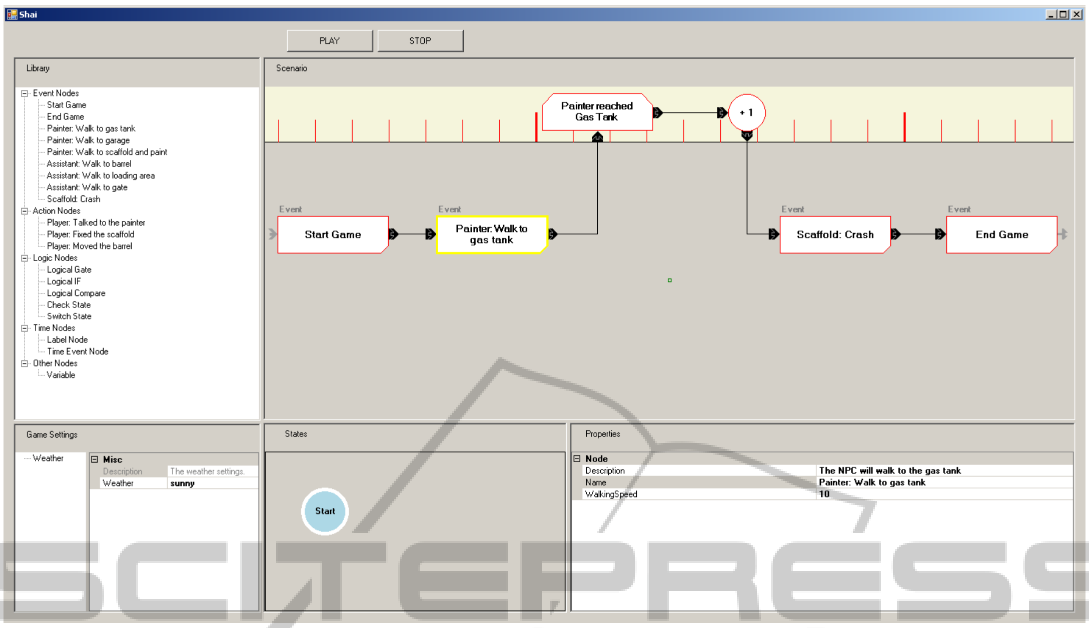
4 PROTOTYPE SCENARIO EDITOR: SHAI

To evaluate the approach discussed in Section 3, a prototype scenario editing application was developed. This Section briefly discusses the design and implementation of that prototype, called Shai.

The Library panel on the left presents the variety of nodes that are available to the user. These nodes can be dragged from the Library and dropped onto the large Scenario panel on the right. Nodes in the Library are grouped by their type; event nodes, action nodes, logic nodes, time-based nodes and miscellaneous nodes. Event and action nodes are specific to a certain game, while the other nodes types can be applied in any game. The node types are corresponding to the entities discussed in Section 3.

The main panel of the editor is the Scenario Panel. Here, nodes can be linked together to form a scenario. Starting the playback of a scenario is straightforward; by simply pressing the play button, the application boots up the game and the scenario begins at the (requisite) ‘start game’ node. While a scenario is being played, the instructor is free to move nodes around and add or remove new nodes, as long as this does not invalidate the scenario (such an invalidation could be detected by the application, but is not available in the current implementation).

Communication between Shai and the game is performed using a separate application, called the Communicator. Messages sent from Shai to the game are first handled by the Communicator. The Communicator can read these messages and decide what to do with them. For instance, it can simply forward the messages to any or all connected game engines, or it can write to a debug log, perhaps by forwarding them to a database, etc.



- Event Nodes
  - Start Game
  - End Game
  - Painter: Walk to gas tank
  - Painter: Walk to garage
  - Painter: Walk to scaffold and paint
  - Assistant: Walk to barrel
  - Assistant: Walk to loading area
  - Assistant: Walk to gate
  - Scaffold: Crash
- Action Nodes
  - Player: Talked to the painter
  - Player: Fixed the scaffold
  - Player: Moved the barrel
- Logic Nodes
  - Logical Gate
  - Logical IF
  - Logical Compare
  - Check State
  - Switch State
- Time Nodes
  - Label Node
  - Time Event Node
- Other Nodes
  - Variable

Weather

Misc	
Description	The weather settings.
Weather	sunny

Node	
Description	The NPC will walk to the gas tank
Name	Painter: Walk to gas tank
WalkingSpeed	10

Figure 3: Interface of the prototype scenario authoring system Shai.

The Communicator can also forward messages from the game engine to Shai. The Communicator can handle multiple connections, both from Shai and (any type of) game engines, using a TCP/IP connection.

By separating the communicator and the scenario authoring application, the author application is shielded off from the game engine. This makes the functionality of the authoring application independent of the type of game engine used, which improves the applicability of Shai.

5 APPLICATION EXAMPLE: SUPERVISOR

This section demonstrates the power of our approach, using Shai in combination with a concrete simulation game called Supervisor. We also discuss how that game needed to be set up properly by the game developer, in order for it to be compatible with Shai.

5.1 Supervisor

The Supervisor simulation game was commissioned by Shell and developed at TU Delft. It is designed to be used as a virtual alternative to
parts of real-life training sessions. In this game, the player (or trainee) assumes, in first person perspective, the role of a safety supervisor at an oil drilling site. The trainee is expected to handle hazardous situations, watch personnel and take care of health, safety and environment regulations. The instructor, or sometimes called the facilitator, is responsible for deploying the game to teach trainees how to become a competent safety supervisor.

Supervisor represents a prime example of the issues that can be found in simulation games. The instructors at Shell had limited influence on the development of the game; they provided information on what type of scenarios should be developed, but once the game was finished, its use was limited to whatever scenarios the game developers had implemented. The instructor, who then used the game to perform training sessions, had only the choice of a handful of different scenarios. Therefore, the use of the game was very limited.

Therefore, we made Shai use Supervisor as a test case, as Shai can be used to improve the use of such as game by allowing the instructor to make changes to the game's scenario, thus expanding the range of possible training sessions.

5.2 Implementation

Before Shai can be used in combination with Supervisor, the game needs to be able to properly communicate with Shai. In the case of Supervisor, which was developed using the Unreal 3 Engine, this required two steps. First, a programmer needed to write code for Kismet nodes, so they could be used in the second step by the gameplay scripter.

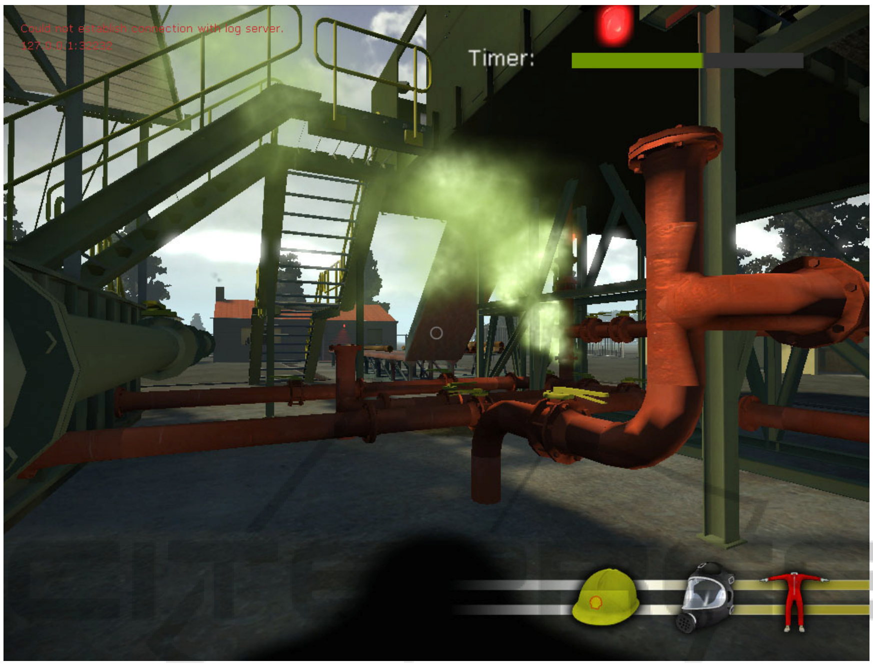

Could not establish connection with log server.
127.0.0.1:32232

Figure 4: A screenshot of Supervisor, as seen through the viewport of the player.

The second step is taken when the programmer has finished writing the code for the Kismet nodes; the nodes can then be placed in Kismet. This is done by the gameplay scripter. He will decide at which points in the game's logic messages will be sent back and forth between Shai and the game.

Thus, the slight overhead required to make Supervisor communicate with Shai is relatively little. On the programming side, there are only a handful of classes that need to be written, and their content is trivial. On the Kismet side, however, placing the extra nodes can be a bit tiresome. However, when this is performed while the level is being created (as opposed to afterwards, as was the case in developing this prototype), the extra work is bearable. Moreover, it helps the designer in keeping the Kismet sequences organized, and guides him into using modular design, which is always beneficial.

6 EVALUATION AND RESULTS

As discussed before, an application as Shai is aimed at non-programmers, such as instructors. The prototype, therefore, needs to be evaluated by its users. For this purpose, an evaluation plan was developed and executed featuring a tutorial for users to follow and a questionnaire to fill in.

The evaluation was performed using a combination of several evaluation methods. Mainly, the user was asked to fulfil a tutorial, in which he was asked to perform several tasks using the prototype. Secondly, the user was interviewed, using both a questionnaire, and a discussion, to find out about his experience with the prototype.

The prototype was tested by several field experts, including experts from Shell, who are familiar with the Supervisor game, and serious game industry experts, who are familiar with scenario editing challenges. The goal of this session was to evaluate feature completeness, adequacy of scenario representation and usability. For more details on the evaluation process, see (Van Est, 2010).

The general consensus amongst the testers was that this scenario editor presents significant advantages in helping instructors develop scenarios. However, the domain experts who already had some experience in scenario development noted that this application could best be used in the preparation phase of using scenarios, because the real-time adjustment options seemed to be too complex in the current form. When large, complex scenarios are involved, it can be very difficult for the user to fully comprehend the long-term effects of changes he is making in the scenario, especially under the pressures of a running game session.

While the current features offered in the prototype, such as the use of a visual node-based causality chain, were well received by the domain experts, they still had some suggestions on possible improvements, which can be considered as valuable recommendations. These included suggestions for conversation nodes, nodes that can retrieve information from the game, sub-graphing options and 3D editing with object placing. Finally, Shai's usability was rated poorly, which implies that significant improvements should be made in this area, if the prototype is considered for practical use.

7 CONCLUSIONS

This project had as its main goal to let the instructor exercise control over the scenarios of serious games by writing a script at the abstracted level of scenario scripting, using the language of scenario building blocks. More control is given to the instructor by allowing him to make real-time adjustments to the scenario as well. A prototype has shown that this approach is applicable and promising.

Now, we hope to see this project functioning as a starting point for future research on developing approaches that help instructors and other didactic or creative experts gain expressive power in the exciting and developing field of games. The current implementation of Shai offers the basics of such an approach, and with the right improvements, it could very well be used to give instructors more control over scenarios in simulation games.

REFERENCES

Holm, R., E. Stauder, et al. (2002). "A Combined Immersive and Desktop Authoring Tool for Virtual Environments." *IEEE Virtual Reality Conference 2002 (VR 2002).*

Iurgel, I. (2004). "From Another Point of View: Art-E-Fact." *Proceedings of the Second International Conference, TIDSE 2004.*

Mateas, M. and A. Stern (2000). "Towards Integrating Plot and Character for Interactive Drama." Socially Intelligent Agents: The Human in the Loop. Papers from the 2000 *AAAI Fall Symposium.*

Mateas, M. and A. Stern (2003). "Façade: An Experiment in Building a Fully-Realized Interactive Drama." Game Developers Conference (GDC '03).

Medler, B. and B. Magerko (2006). "Scribe: A Tool for Authoring Event Driven Interactive Drama." Proceedings of the Third International Conference, TIDSE 2006.

Moreno-Ger, P., C. Blesius, et al. (2007). "Rapid Development of Game-like Interactive Simulations for Learning Clinical Procedures." Proceedings of the Fifth International Game Design and Technology *Workshop and Conference (GDTW2007).*

Moreno-Ger, P., I. Martinez-Ortiz, et al. (2005). "The <E-Game> Project: Facilitating the Development of Educational Adventure Games." Proceedings of the *5th International Conference on Entertainment Computing (ICEC 2005).*

Smith, R. (2007). "The Five Forces That Are Driving the Adoption of Game Technologies within Multiple Established Industries." Games and Society Yearbook.

Spierling, U., S. A. Weiss, et al. (2006). "Towards Accessible Authoring Tools for Interactive Storytelling." *Proceedings of the Third International Conference, TIDSE 2006.*

Susi, T., M. Johannesson, et al. (2007). *"Serious Games - An Overview."*

UnrealEd (2011). Epic Games, http://udk.com/ (last accessed 06/01/2011)

Van Est, C. (2010) *"Shai: Scenario Editing for Simulation Games",* MSc Thesis.

Weiss, S., W. Muller, et al. (2005). "Scenejo – An Interactive Storytelling Platform." *Proceedings of the Third International Conference, ICVS 2005.*