Scribe: A Tool for Authoring Event Driven Interactive Drama

Ben Medler and Brian Magerko

Games for Entertainment and Learning (GEL) Lab
Michigan State University
253 Comm. Arts Bldg., East Lansing, MI, USA 48824
medlerbe, magerko@msu.edu
http://gel.msu.edu

Abstract. Creating an interactive drama requires authors to produce large quantities of story content. A programmer or knowledge expert typically creates this content because they have experience with the story environment. By using an authoring tool someone with less experience with the story environment can organize and create story content. The Scribe Authoring Tool is an authoring tool that will be used to create interactive dramas. The tool will follow certain requirements to make it relevant to any story environment and will be usable enough for someone not familiar with the environment to author story content for interactive dramas.

Keywords: interactive drama, authoring tool, story representation.

1 Introduction

Authoring an interactive drama requires the author to generate a large amount of story content [9, 13]. However, the creative process of authoring content for an interactive drama typically requires an author that is both an artist (to generate the creative content) and a programmer (to encode that content in the logical representation being used, such as a partial-order plan [19], ABL [12], or plot points [9]). The creation of tools to aid in the authoring process is a necessity to make this task both more efficient and easier for authors who may not be computer programmers. Other authoring tools have been made for interactive dramas such as DraMachine [6], Art-E-Fact [8], and Scenejo [18]. These tools are lacking in encompassing everything needed for an interactive drama. Authoring tools must be robust and usable enough for them to encode their creative vision. It is for this purpose that we are creating a new authoring tool that will be independent of both game environments and story content and increase the scope of the available authoring functions for an interactive drama creator.

The Scribe Authoring Tool is the new authoring tool being used for the Interactive Storytelling Architecture for Training (ISAT). ISAT is a training architecture that combines elements of the interactive drama architecture IDA [10] and intelligent tutoring within a game environment [11]. The current instantiation of ISAT is being

developed to train field knowledge for combat medic students. ISAT uses a 3D game environment and takes the trainees through a story that requires them to perform soldier and medical duties while on active duty. Scribe will give subject matter experts (SMEs), who typically have no programming experience, the means of authoring relevant, meaningful interactive dramas that trainees will experience in this 3D game environment. We define requirements that are necessary for Scribe and discuss how this new authoring tool will function.

2 Requirements for an Interactive Drama Authoring Tool

To avoid creating a limited tool, we examined what makes a good authoring tool and how we could produce one that was more robust. We reviewed other authoring tools and programs to define a set of further requirements, in addition to content creation, that would have to be fulfilled.

2.1 Generality

Most authoring tools are made for a specific game environment or story setting [1, 2, 4, 5, and 15]. While these tools are efficient in their specific domain, they are not well suited for the varying and dynamic storytelling environments that are needed for interactive drama. A more *general* tool could be re-used across environments and story contexts rather than reinventing a new tool.

2.2 Enables Debugging

Debugging becomes extremely difficult when dealing with interactive dramas due to the large amount of content that they use. Interactive drama users are able to play through many different possible storylines [9, 13]. Content must be produced in large quantities with complex relationships between the various parts to allow for these variable storylines. With this larger structure comes an increase in the number of possible problems in both the system behavior and the authored content, such as poor decision making by the intelligent agent coordinating the story, redundancy in the storyline, continuity problems, or dead-ends that could affect the user experience.

It is difficult for the author to play through a large interactive drama fixing errors as they play each storyline possibility [9, 13]. Debugging of this content is typically done through playtesting the experience and observing if the correct behavior occurs. An authoring tool could help facilitate this process by decoupling the real-time experience of an interactive drama from the authored story content and agent behavior. By allowing debugging inside of an authoring tool, the author will save time by not having to switch into the story environment, and will be able to play through the story quicker, with various story navigation options.

2.3 Usability

Authoring tools are used to make creating stories easier. If the tool is hard to use and is not intuitive for a prospective author, then the tool becomes more of a hindrance than an aid. *Usability* includes ease of learning, efficient to use, tasks are easy to remember, and users make few errors while being overall pleased with using the tool [14]. An authoring tool's functions must allow the author to know what is happening by not having complex feature sets or mislabeled functions.

2.4 Environment Representation

Each story that is written for one type of environment can be similar to the others in many ways, but in different environments two stories can highly vary in game mechanics, narrative devices, or user interaction abilities. To protect against these varying environment definitions, an authoring tool will need an infrastructure to understand these definitions and have a relative representation of any environment defining these definitions. Examples of this representation include displaying map data, providing relative variables to manipulate, determining Non-Player Characters (NPCs) behaviors, etc.

2.5 Pacing and Timing

The *pacing and timing* of a story helps create dramatic feelings and allows the story to have a greater appeal to the reader by timing when information is given to them [17]. *Pacing and timing* can be crucial to dramatic development and encoding these effects is important to story representation. An authoring tool should allow an author to create timelines that bring captivating effects to their stories, which as been done in other narrative media, such as computer games, film, and literature. The tool should provide the author with ways in which to specify their stories *pacing and timing* that also adheres to the requirement of *usability*.

2.6 Scope

Interactive drama content includes: character behavior, story representation definitions, dialogue scripts, etc. and an authoring tool must cover this wide *scope* with authoring functions. This ensures that the pieces of an interactive drama can be manipulated and understood by the author. This gives one centralized tool in which these authoring functions take place.

3 Related Work

Authoring tools are used in many software developing situations as a means to generalize the software building process to make building applications easier.

Authoring tools are found in PC game titles as a way for users to make their own content for the game with little or no coding [1, 2, 4]. Game genres such as Real Time Strategy (RTS) and Role Playing Game (RPG) generally have authoring tools that accompany them. There have also been tools, such as Redux [15], that show how authoring story content can be achieved for an environment such as the one in ISAT. Besides related tools, other interactive drama architectures such as Façade [13], MIMESIS [18] and IN-TALE [16] would benefit from using an authoring tool such as Scribe.

Authoring tools used for RTS and RPG games are fine examples of how typical authoring tools work in the game industry. Games like Starcraft [2], Warcraft 3 [4], or Neverwinter Nights [1] have functional authoring tools that allow users to build maps/levels, design gameplay effects, and create storylines. However, these tools are domain specific and do not have the *generality*, or *enables debugging*, for environment actions and story creation abilities that are needed for interactive drama.

Another authoring tool, ScriptEase [5], was made for a commercial game but was produced from an academic perspective. ScriptEase is used to build gameplay and storylines for an RPG called Neverwinter Nights (NWN). Again, this tool is domain specific, but ScriptEase fulfills two of the requirements discussed above, *generality* and *usability*, by generalizing actions and commands while using a simple interface.

Redux [15], an authoring tool that has influenced the design of Scribe, is used to create Human Behavior Models (HBMs) and helps SMEs create annotated diagrams that represent close quarter combat inside buildings. These models are then displayed inside a 3D environment for trainees to study. Redux however only presents linear diagrams of various tactics and models them. Scribe allows the creation of story content that will be presented to the trainee in a nonlinear way, based on the director agent's decisions (described in the following section) and trainee interaction.

Other interactive drama authoring tools, DraMachine [6], Art-E-Fact 8], and Scenejo [18], look into representing different parts of interactive dramas. DraMachine separates different story parts so that authors can work a story piece by piece. Art-E-Fact uses a directed graph approach to its stories and focuses on an NPC dialogue system. Scenejo uses a similar dialogue system, which allows the author to set different variables related to NPC actors. These variables use a dialogue database to communicate with other NPC actors. While each of these authoring tools help with certain pieces of interactive dramas, none of them bring together all of the requirements for an interactive drama authoring tool.

A current interactive drama architecture used for training, IN-TALE [16], is attempting interactive drama goals similar to our ISAT project. IN-TALE is an architecture that, like ISAT, allows the story and environment to change accordingly to how the trainee is acting. How ISAT differs from IN-TALE is that IN-TALE focuses more on the behaviors of intelligent agents and how these behaviors will make the story dynamic. ISAT focuses more on story representation and the trainee's specific action in the environment to influence a relevant storyline for that trainee. ISAT is also including Scribe as the key to help produce this story representation and manipulate how a trainee's actions will affect the story. Currently IN-TALE lacks an authoring tool for its architecture.

The field of interactive drama is sorely lacking in authoring tools for the myriad of approaches out there, much less a tool that is *general* enough to be used with different

approaches. Other interactive drama works [13, 16, and 18] have architectures that would greatly benefit from an authoring tool such as Scribe. Programmers, or other knowledge experts, add time and cost to these other architectures. Having an authoring tool that adheres to the above stated requirements will work well for all of these story representation programs.

4 Overview of Scribe

Scribe is being created at the Games for Entertainment and Learning (GEL) Lab. Scribe's design is an attempt to fulfill the necessary requirements that are stated in Section 2. It is being built in parallel with ISAT and will be using a simulation environment called the Tactical Combat Casualty Care (TC3) trainer [7]; however, Scribe is designed with the intent of being more *general* than use only within ISAT. TC3 is a 3D game environment where Army medic trainees will be able to experience and interact with the content that is created in Scribe. The TC3 environment allows trainees to perform duties that will correspond to actions that a medic must perform out in the field. Work has been completed on an interface prototype of Scribe using Adobe Director. We are currently working on the core Java development of the tool for integration with ISAT and the TC3 trainer.

The hypothesis behind the ISAT approach is that interactive drama techniques can be used to provide more effective and engaging training experiences. ISAT's main component is an intelligent director agent that employs story-mediation techniques to manage the story world in response to authored story content and trainee actions. The director makes decisions based on the trainee's actions in the world and the state of the trainee skill model, which is the director's hypothesis of the trainee's aptitude in the set of skills being trained. The director can alter the environment and selects plot content to maximize the trainee's learning experience. The author can create story content and will provide information for the director to use and follow as it gives each trainee their own experience in the environment.

5 Authoring Modes

The structure of Scribe is split into three different authoring modes, each with its own functionality for producing story content:

- **Element Placement** will allow the author to view a 2.5D (shows a 2D figure but annotates height similar to a topographic view) version of the environment map. Here the author will place element pieces (defined below) on the map. These element pieces will be the dynamic content that will interact with the trainee.

- **Story Creation** is where the Author will create the storyline structure of what will occur as the trainee plays through the story. Varying levels of story detail allow the author to control different aspects of the story and its

elements. Story Creation makes use of the configuration of the element pieces created by the author during Element Placement to create logical story statements from the visual map representation.

- **Debugging** will allow the author to interact with the director agent inside of Scribe. Here, the storyline can be easily navigated by the author and will be allowed to query the director as to how it will handle various storyline situations, which serves to both ensure that the story content and the director's behavior is correct.

Though this structure is comprised of three parts, Scribe allows the author to flip between these parts freely, providing a larger *scope* for an iterative design and debugging process. Changes to the story content areas are global in nature, giving the Author the ability to change the content and have it reflected in all of the areas.

6 Scribe Authoring Example

6.1 Element Placement

Fig. 1. Screen shot of the Element Placement window for the Scribe prototype.

This example illustrates the features of Scribe and how they relate to the overall requirements described in section 2. The example involves three plot points (i.e. scenes from the story) to explain the story modes. The first plot point gives the trainee four wounded soldiers to treat. The plot point also requires eight friendly NPC

soldiers (the trainee's squad) to be placed on the map as well. The plot point will have the friendly soldiers standing guard while the trainee treats the wounded soldiers.

In this example, the author starts by creating a new project in Scribe and chooses a map where the author would like their story to take place. In terms of the ISAT project, a map is a large 3D model that is designed to look like realistic places a military medic could find out in the field. The author chooses a map and a representation of it is shown in Element Placement mode [Figure 1]. As part of the *generality* requirement, we created a XML protocol that can be used to represent 3D maps. This allows Scribe to understand any other map (necessarily ISAT specific) that is encoded in the same XML protocol. Scribe can read a map's XML file and create a 2.5D representation that will show an overhead shot of the map along with some height information, allowing the author to set elements on the map.

A Scribe element is anything tangible in the game environment that interacts with the trainee (e.g. NPCs, environment objects, or invisible elements like spawn points, which are described below). The goal of the Element Placement is to place these elements and configure them for the plot point(s) with which they are associated. There are three types of elements: objects, points, and zones. Objects are anything that is visible to the trainee (e.g. NPCs, cars, buildings, trees, etc.). Points are invisible coordinates that exist in the game environment that can be annotated for reference by the director agent or for some use in the story. For example, a point can be used to spawn enemy soldiers at its location or to denote a point of interest to the synthetic characters. Zones are invisible rectangular prisms with dimensions that can be referenced by the director agent or used in the story. For instance, an event could be triggered when the player walks through a certain zone.

These element types fulfill part of the *generality* and *environment representation* requirements. We find that many stories taking place in a 3D environment will be able to use these three types of elements. Points and zones will allow the author to organize sections and specific locations in a 3D world. Objects will represent the tangible items that are seen in the environment. Examples of these element types have been used before in 3D game engines such as the Unreal Engine, so these representations have been generalized to fit most 3D environments [3, 9].

With the map now chosen and displayed, the author can start placing elements. For this story example eight friendly soldiers and four injured soldiers must be placed on the map according to the first plot point. Scribe allows the author to specify which type of element is being placed. To place the 12 NPCs the author turns on "object elements" and clicks on the map in the 12 spots they wish to place the NPCs. Each time an element is placed, a temporary element is created, allowing the author to annotate it with relevant information.

The author places 12 temporary object elements to represent the other NPCs on the map. By clicking on any of the temporary objects a list of variables is displayed in the Element Placement window. Each element has a set of variables that are associated with it and are set by the author. These variables are determined outside of the tool by encoding the list of variables, which the director agent understands, into a XML protocol that Scribe uses, allowing for *generality*. This *environment representation* ensures that the content is relevant to the environment.

Having set the 12 objects to eight soldiers and four casualties, the author has now completed the setup for the first plot points. As explained in the next section, each

plot point has a start and end states that the elements of that plot point should be in for the plot point to begin. To set these states, the author selects a plot point and designates the current configuration as one of the plot point's states. At this stage, there are no plot points and the author must create one in Story Creation mode.

6.2 Story Creation

This section describes the design of the Story Creation mode [Figure 2], which was implemented in our interface prototype and is part of our current implementation efforts. Scribe uses plot points to describe major parts of the overall storyline, similar to scenes in a play [10]. Each plot point consists of three parts: preconditions, events, and actions, which were inspired by work done on IDA's story representation [9]. These three parts help with the *generality* of plot points because the statements in each of these parts can be relative to whatever environment Scribe is used for, as shown below.

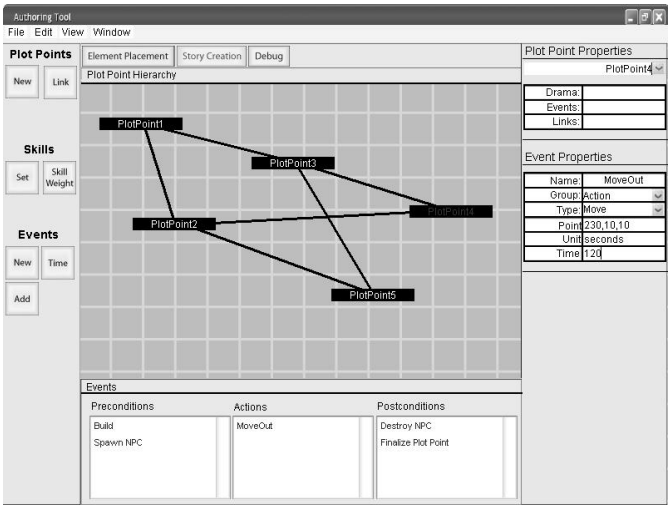

Fig. 2. Screen shot of the prototype Story Creation window for Scribe.

Preconditions are logical statements that must be true for the plot point to occur. For example, a plot point precondition may state that the trainee must be in the marketplace (i.e. Location(User, Marketplace)). If the trainee is in the market place, then that precondition is fulfilled. With the preconditions and other ordering constraints fulfilled, the plot point can occur. Actions are logical statements that describe the changes in the world caused by a plot point's completion. For example, if a plot point had the trainee meet a NPC, an action could state that they have met. Finally, events are a set of statements that dictate what changes in the world occur during a given plot point. This is similar to systems such as Façade and IN-TALE that

use dramatic beats [12, 16] which represent important occurrences that happen through out a story and happen over a fairly short period of time (about a minute).

Scribe events are used to temporally describe scripted, possibly temporally overlapping, changes in the environment. An event could call for an NPC to move to a new location, spawn a new enemy unit down the road, or start a dialogue that involves the trainee. Scribe allows for events to overlap one another, something IDA's story representation lacked [9], and intermixes NPC behaviors or goals in with environment commands. This is different from using ABL, such as in Façade and IN-TALE, as it focuses on NPC behavior to achieve story goals [12, 16]. However, Scribe currently assumes that NPCs are goal-based and directable. Note that Scribe has the ability to work with NPC behaviors but does not create them. The ABL language or Redux could be used to create NPC behaviors and could be imported into Scribe.

When the author switches to Story Creation mode, the map is taken away and a graph area to lay out plot points replaces it. The plot point graph area allows for the visual organization of plot points and events. Plot points can then be created or selected by the author and annotated (i.e. adding preconditions, events, and actions). For this example, the author creates a plot point and it appears on the graph area. The author then flips back to Element Placement mode and sets the initial position for the elements appearing in this plot point.

Scribe is capable of making use of a planning algorithm or relying on the author to causally connect each plot point to form a story plan. When the author sets a plot point's preconditions and actions for use in the story plan ordering and casual links are created to connect plot points. Ordering links restrict plot points from occurring until all order connected plot points occur. Casual links connect the actions of plot points to the preconditions they fulfill. This representation allows for the possibility of replanning when threats to the plan occur, as IN-TALE [16] or MIMESIS [19] would do, but will also allow the author to create the plan by hand. The representation also differs from typical narrative planning languages by incorporating the overlapping temporal events within in plot point. The representation will be evaluated after the completion of the tool for *usability*. Another option that could represent the story graph would be to use a representation similar to ABL or IDA's that has no explicit causal representation [12, 10].

All statements (preconditions, events or actions) can be set in the Story Creation window. Like elements, statements are defined outside of the tool, using a XML protocol, and are defined to how the director agent will be able to understand them. For this example the author creates two event statements, a dialogue and a conditional statement. The author can then set the variables of each event, with each event having its variables listed as part of the XML protocol.

The author adds a condition stating that after 300 seconds a new plot point will be triggered to run. The trainee may be in the middle of treating a casualty, and a new plot point could start. This does not mean that whatever the trainee was doing immediately stops, what this means is that now a new set of events will start running and may change the environment, forcing the trainee to react to these events. The author then adds the dialogue event to the example plot point, which occurs after the plot point starts and before the condition event starts. In this example an event is placed that has a soldier telling the trainee to treat the wounded soldiers.

Events need a start and end time to facilitate the *pacing and timing* requirement. There are three different types of time functions: fixed, random, and relative. Fixed time is any static amount of time, for example, 300 seconds. Random time allows an author to set a time range, zero to given number, in which a number will be randomly selected at runtime. The last type is relative time, or time relative to other events. An event could be set to start after another event ends, for example. These types provide *generality* to how time functions for events. For *usability*, each event's start and end time are placed on a linear layered timeline that is shown in a similar way to how movie editing software displays video clips.

The last thing that the author can use in Story Creation mode is the skill model. ISAT is currently used for training medic skills, such as how to apply a tourniquet. Performing skills correctly means that the trainee has a higher score in the corresponding skills. Story statements can take this skill information into account, such as setting how high a trainee's skill has to be before a certain plot point can begin. Again, skills are part of a XML protocol, for *generality*, that is read in by Scribe. In this example the author does not deal with the skill values.

So the author now has a functional plot point in this example. The trainee will be asked to treat the four casualties and after 300 seconds, a new plot point will begin. As the complexity of the story increases, the possibility of more errors occurring increases (e.g. director error, content error, etc.). In order to protect against these errors, functionality for debugging the story is included in Scribe.

6.3 Debugging Story Content

Debugging the story in Scribe will be able to save a large amount of time that would have been spent inside the environment going over the many different ways of working through the storyline [15]. What Debugging mode will allow is a direct communication link to the director, simulating how the environment will act for the trainee. The director can be queried with a given situation in the tool and gives an answer, with an explanation of why this decision was made. This allows the author to determine both the correctness of the story content and the director's behavior.

The example thus far has had only one plot point. Adding two more plot points will give a better example of how debugging will potentially work inside of Scribe. These next two plot points will introduce enemy units, requiring the trainee to react to their presence. The difference between these plot points will be based on how well the trainee is performing. If the trainee is easily treating the injuries then plot point 2 will be chosen, otherwise plot point 3 will be selected.

In Debugging mode, the map is once again presented to the author. The map will be used to display a rough version of how the elements on the map will look throughout the plot points. The trainee is also simulated, both as a unit on the map and as a skill model. This allows the author to move the trainee around on the map and manipulate the skill variables to test how different positions and skill values affects the director's decisions. The author selects a plot point to test and then queries the director about the story's events. When the director makes story decisions it will take into account: element setup on the map, the trainee's location, the trainee's skill model, etc. The author may also query the director to choose the next plot point.

For this example, the author wishes to test which new plot point will be selected next based on the trainee's skill model. The author sets the simulated skill model to show high percentages. The director should decide on plot point 2. The director, however, chooses plot point 3 stating the trainee's Apply Tourniquet skill was too low (this explanation feature is still under consideration). This could mean the author needs to set the apply tourniquet's skill value even higher, or the director may be interpreting the previous plot point's events wrong. With the director's actions known the author can go back and make the changes needed to get the desired results.

Enables debugging is one of the harder requirements to fulfill because of all the storyline possibilities that are part of interactive dramas. Debugging is scheduled to be included in Scribe but currently is still being designed. However, this feature of Scribe's design is a novel one in the field of authoring tools and is potentially an extremely useful one.

7 Discussion and Future Work

As stated before, Scribe has existed as a user interface prototype and is currently under Java development. Currently, Scribe works with one environment and one director agent. Additional environments should be easier to connect with by using the various XML protocols to allow for *generality* and relevant *environment representation*. Other representation abilities such as dialogue generation, which depend more on the architecture used with Scribe, are included in the design but how they will affect the tool is still being discussed. *Usability* of the tool has undergone internal user testing, with future external usability tests to come further into development. Events have the ability to have their *pace and timing* set so the story can be controlled to a higher degree and Scribe is scheduled to *enable debugging*. Scribe does a good job of fulfilling the above requirements and providing the *scope* needed for an authoring tool that will be used to produce interactive dramas.

The director and Scribe interactions are still being finalized. How much power the author will have over what the director can do with the story content must be determined. Also, how the director uses the story content is still under development. One example of this is whether a planner will be the best option to create the story graph that will then be used by the director at run-time.

Debugging is currently low priority on the project schedule since it is not part of the final project deliverables but it is a feature we want to add to Scribe. It is our goal to at least get test cases and a prototype interaction with the director by the end of next year. In the future, expanding on any work that is completed on Scribe's debugger, or testing other debugging methods, will be the best way to further the abilities of this authoring tool.

The ISAT project is looking at whether interactive drama could be more effective for training than by using other interactive or physical means. Scribe will help non-programmers, the real users of these tools, create superior story content. Through the definition of key areas of storyline creation, placing elements, creating the plot, and debugging interactions, Scribe will ease the pressures of creating interactive drama.

References

1. Neverwinter Nights, Atari, 2002.
2. Starcraft, Blizzard Entertainment, 1998.
3. Unreal Engine, Epic Games Inc., 1998-2006.
4. Warcraft 3, Blizzard Entertainment, 2002.
5. Carbonaro, M., Cutumisu, M., McNaughton, M., Onuczko, C., Roy, T., Schaeffer, J., Szafron, D., Gillis, S. and Kratchmer, S.: Interactive Story Writing in the Classroom: Using Computer Games. *International Digital Games Research Conference*, Vancouver, CA, 2005.
6. Donikian, S. and Portugal, J.-N.: Writing Interactive Fiction Scenarii with DraMachina. *In the Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment: Second International Conference*, pages 101-112, Darmstadt, Germany, 2004. Springer Berlin / Heidelberg.
7. Fowler, S., Smith, B. and Litteral, C.D.J.: A TC3 Game-based Simulation for Combat Medic Training. *The Interservice/Industry Training, Simulation & Education Conference*, 2005.
8. Iurgel, I.: From Another Point of View: Art-E-Fact. *In the Proceeding of the Technologies for Interactive Digital Storytelling and Entertainment: Second International Conference*, pages 26-35, Darmstadt, Germany, 2004. Springer Berlin / Heidelberg.
9. Magerko, B.: Building an Interactive Drama Architecture. *In the Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pages 226-237, Darmstadt, Germany, 2003. Springer Berlin / Heidelberg.
10. Magerko, B. and Laird, J.E.: Mediating the Tension Between Plot and Interaction. *In AAAI Workshop Series: Challenges in Game Artificial Intelligence*, pages 108-112, San Jose, CA, 2004.
11. Magerko, B., Wray, R.E., Holt, L. and Stensrud, B.: Improving Interactive Training through Individualized Content and Increased Engagement. *Interservice / Industry Training, Simulation, and Education Conference*, Orlando, FL, 2005.
12. Mateas, M. and Stern, A.: A Behavior Language for Story-Based Believable Agents. *AAAI Spring Symposium Series: Artificial Intelligence and Interactive Entertainment*, Palo Alto, CA, 2002.
13. Mateas, M. and Stern, A.: Facade: An Experiment in Building a Fully-Realized Interactive Drama. *Game Developer's Conference*, San Francisco, CA, 2003.
14. Nielsen, J.: Usability Engineering. Morgan Kaufmann, 1994.
15. Pearson, D.J. and Laird, J.E.: Redux: Example-Driven Diagrammatic Tools for Rapid Knowledge Acquisition. *Behavior Representation in Modeling and Simulation Conference*, Arlington, VA, 2004.
16. Riedl, M.O. and Stern, A.: Believable Agents and Intelligent Scenario Direction for Social and Cultural Leadership Training. *Behavior Representation in Modeling and Simulation Conference*, Baltimore, Maryland, 2006.
17. Rimmon-Kenan, S.: Narrative Fiction. Methuen & Co. , 1983.
18. Weiss, S., Muller, W., Spierling, U. and Steimle, F.: Scenejo - An Interactive Storytelling Platform. *In the Proceeding of the Virtual Storytelling: Using Virtual Reality Technologies for Storytelling, Third International Conference*, pages 77-80, Strasbourg, France, 2005. Springer Berlin / Heidelberg.
19. Young, R.M., Riedl, M., Branly, M., Jhala, A., Martin, R.J. and Saretto, C.J.: An Architecture for Integrating Plan-based Behavior Generation with Interactive Game Environments. *Journal of Game Development*, pages 51-70, 2004.