The Use of Functional L-Systems for Scenario Generation in Serious Games

Glenn A. Martin
University of Central Florida
3280 Progress Drive
Orlando, FL 32826
+1 (407) 882-1300
martin@ist.ucf.edu

Charles E. Hughes
University of Central Florida
4000 Central Florida Blvd.
Orlando, FL 32816-2362
+1 (407) 823-2341
ceh@cs.ucf.edu

Sae Schatz
University of Central Florida
3100 Technology Parkway
Orlando, FL 32826
+1 (407) 882-1300
sschatz@ist.ucf.edu

Denise Nicholson
University of Central Florida
3100 Technology Parkway
Orlando, FL 32826
+1 (407) 882-1300
dnichols@ist.ucf.edu

ABSTRACT

So called "serious games" have used games (in a sense, virtual environments) for reasons other than entertainment. Particularly within the training community, they have garnered increasing attention over recent years. However, means of generating new scenarios that have increased training effectiveness has continued to be lacking. Because creating new scenarios is a time-intensive and costly exercise. existing scenarios are commonly reused with only minor changes, a practice that can hamper training effectiveness over time.

We have been pursuing a thrust of research in automated scenario generation. In this paper, we present our work in the use of Functional L-systems for generating scenarios. We first review some of our previous work in defining scenarios; then show how Functional L-systems are used to build up the scenarios.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems – *environments.*

General Terms

Algorithms, Management, Design, Human Factors, Languages.

Keywords

Scenario Generation, Simulation, Training, FL-Systems.

1. INTRODUCTION

Serious games (and virtual environments, in general) have great promise for use in training. However, for the most part, scenarios are created manually, which is a costly and time-intensive process. The consequence is that a small set of scenarios is commonly re-used over and over, with few or no changes, which can cause reduced training effectiveness.

We are pursuing a line of research investigating procedural generation for automating the scenario building process. The initial focus of our work is in supporting the creation of scenarios for Fire Support Teams in the U.S. Marines. However, the system we are building addresses the general case and is, thus, adaptable to many other domains.

Fire Support Teams coordinate artillery strikes and ground attacks by aircraft on targets through a complex set of actions. They typically observe the targets and direct such strikes by providing direction to the firing units. Scenarios based on their training must provide the team position, artillery and airstrike assets, and target(s). Figure 1 shows an image from one common Forward Observer application used for training [1].

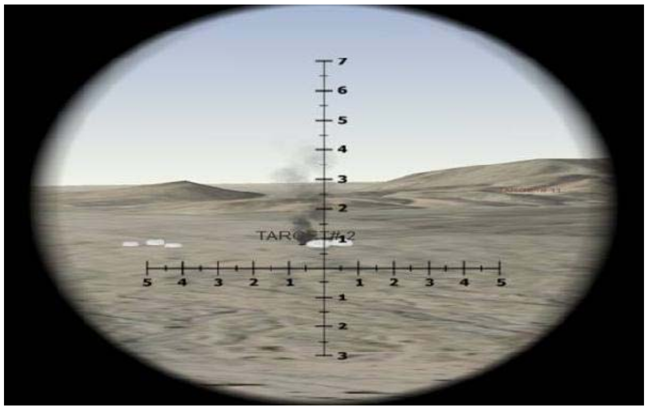

Figure 1. A Forward Observer application

2. SCENARIO-BASED TRAINING

The use for scenario generation for training arises from the desire to increase the effectiveness of the training. In fact, scenario-based training is distinguished from simulation-based training. Here, simulation-based training refers to the simply use of a virtual environment to provide practice of some skills or tasks. In contrast, scenario-based training is based on the targeted creation of specific simulator events to create desired psychological states.

Scenario-based training is now widely accepted. Through its use, trainees can learn to integrate multiple supporting skills, cope with realistic distracters, practice their higher-order cognitive skills, and exercise naturalistic decision making [2]. However, it has been found that development of these advanced cognitive skills requires extensive varied experience [3][4]. This notion drives the need for scenario generation to further improve training effectiveness.

3. EXISTING SCENARIO GENERATION

Before we consider our own approach, we review just a few contributions from others. A complete review is beyond the scope of this paper, but here we focus on works that provide some key concepts. Many fields have elements to offer scenario generation (such as interactive and narrative storytelling); however, our focus here is on actual systems built with training in mind.

One event-based approach to scenario generation was the Rapidly Reconfigurable Event-Set Based Line-Oriented Evaluations (RRLOE) Generator, developed for Federal Aviation Administration (FAA) flight simulators [5][6][7]. RRLOE builds scenarios from small, FAA-approved scenarios. A set of 128 heuristics determine the adequacy of each constructed scenario. RRLOE is highly successful and is still used by the FAA for pilot qualification testing and training.

Similar to RRLOE, the Interactive Specification Acquisition Tools, ISAT, uses heuristics to build a scenario using smaller scenario pieces [8]. However, in the case of ISAT, each scenario piece satisfies some subgoal. In addition, it performs analysis to determine error states in the heuristic model; for example, identifying states that are never executed and those with conflicting "next" states. ISAT also allows users to intercede in the generation process, permitting scenarios to be more finely tailored.

Pffefferman developed a system for automatic scenario generation associated with combat simulations [9]. His focus was on adapting a structured "mission file" in order to create a scenario. The mission file included information on the situation, mission, execution, service support, and command and signal elements. An important aspect of his work is in the use of domain-specific information (military doctrine in his case) for filling in gaps missing from the mission file.

The Framework for Enabling Adaptive Scenario Generation for Training (FEAST) uses fine-grained training context analysis and knowledge modeling methods to support generation of scenarios [10]. Its focus is on dynamic, adaptive training and is unique in its use of captured domain knowledge. A "domain ontology" is formed and drives the scenario generation process [11][12].

Tbese previous scenario generation efforts provide many lessons learned. RRLOE and ISAT show the possibility of building scenarios from smaller, premade mini-scenarios. The major advantage of this approach is that it allows smaller elements to be independently developed and then certified as "valid" and stored for later use. Pfefferman's approach shows us that the structure that a specific domain may offer can be key to addressing the automated generation desire. In addition, it also allows such a system to have some knowledge to guide it in filling in gaps in information. FEAST takes it one step further showing how an operationalized, comprehensive domain ontology can help drive scenario generation as well.

Even with these significant contributions, many challenges remain. In particular, despite the possible advantages of working within a single structured domain, there are strong incentives (e.g., cost, consistency, verifiability and tool integration) to build a generation system that can adapt to multiple domains. Most importantly, we want to avoid a "stovepipe" approach where we build a system for only one set of training applications.

Still, before we can consider our own system, the concept of a "scenario" must be defined and made concrete enough to support an automated approach. In the remainder of this paper, we will briefly review our work in defining a scenario in concrete terms before going into details of our automated approach.

4. SCENARIO DEFINITION

Before a system can be built to automate (or even semi-automate) the scenario generation process, a scenario must be defined in terms that such an automated process could be built around. In our previous work, we developed a detailed model of a scenario for this purpose [13]. Scenarios are defined in terms of training objectives, baselines, augmentations and vignettes (themselves, defined in terms of triggers and adaptations). Each is referred to as a facet of the scenario and may also have a set of "requirements" that must be defined and met.

Training objectives are a list of specific tasks, appropriate for the domain, that potentially require training. Training scientists will further split these objectives into Knowledge, Skills and Attitudes (KSAs) that focus the task on a particular learning objective. The selection of training objectives drives the rest of the scenario generation process since it determines exactly *what* is to be trained.

Baselines are the simplest form of scenario. They are minimal (they may contain only the trainee within a particular map or database) and take place in perfect conditions (perfect lighting and no detrimental weather such as rain). Baselines provide the foundation for the scenario generation process. When generating a scenario, a single baseline is chosen that is appropriate for fulfilling the training objective(s) selected.

Tomizawa and Gonzalez very nicely define the difference between scenarios and situations [14]. Situations define a snapshot in time while scenarios include events that occur to define the overall exercise. Within our scenario model, augmentations help define the initial situation of our training scenario. Specifically, augmentations are used to define initial elements of the scenario. This can include the type and position of any friends or opposition, or it may include overall effects throughout the scenario (such as nighttime or rain). Together with a baseline, the augmentations define the initial condition (situation) of the scenario.

Vignettes are "mini-scenarios" and provide for events that may occur during the scenario (or exercise) itself. Vignettes add the time element to the scenario and help distinguish the scenario from a situation. We build vignettes in terms of a set of triggers and a set of adaptations. Triggers are events that can be detected and can be chained together to form a more complex notion. It may include elements such as a specific exercise time being reached or an explosion going off near a specific location. An adaptation can be added to a trigger (or a trigger chain) to cause some resulting behavior. Example behaviors include creating a new entity (addressing the problem of a key entity being killed too early in a training exercise) or killing an entity (if a munitions round falls near an entity but does not kill it, it may be desirable to kill the entity anyways).

5. SCENARIO BUILDING

Obviously, there is a wide range of scenarios that can be built. When building a scenario, we use a notion of "scenario complexity" in deciding what should be in a scenario. We define scenario complexity as a quantity between 0 and 100, although we often split the range into three portions representing novice, intermediate and advanced complexities. Once a desired complexity is chosen (usually based upon the trainee's past performance profile), the scenario is built up to that level. When considering complexity, the goal of the scenario building process is to create a scenario within a specific complexity range (since achieving a specific single value would be difficult).

Each of the baselines, augmentations and vignettes has an assigned complexity level. Typically, this is a number between 0 and 100 and is chosen by a subject matter expert. As mentioned, before choosing training objectives, an overall desired complexity level of the scenario is entered. As the baselines, augmentations and vignettes are chosen, the current scenario complexity level is increased and tracked. Scenarios must be within the defined complexity range to be considered valid for the given trainee. Only once a scenario is deemed valid may it be exported to the specific training applications.

Once the facets forming the basis for a scenario are chosen, the scenario is conceptually built. However, there is one final step necessary. Scenario facets will specify a particular component of a scenario (such as a Target), but often they will leave some parameters of the facet unspecified. For example, the type and position of an entity representing a Target may need to be specified.

Given this approach, we can support a manual scenario generation process. A user can select the complexity and training objectives desired; then a baseline, zero or more augmentations and zero or more vignettes are added. The user then satisfies the requirements of each facet that are unspecified. Each facet adds a complexity cost to the total scenario (a simple addition of costs is currently performed). The system enforces the desired complexity level and will not allow the user to export the scenario unless it is within the correct complexity level range. However, this alone does not support automatic or semi-automatic scenario generation.

6. PROCEDURAL GENERATION

While we wish to support a manual scenario building process (to encourage acceptance of the overall process), our main focus is on supporting automatic or semi-automatic scenario generation. The cost of building scenarios is a major problem in the simulation and training domain. This, coupled with recent impressive demonstrations of procedural generation, motivated us to consider the potential use of procedural modeling as a mean to address this problem.

6.1 Previous Scenario Generation Work

A complete examination of previous work in this area is beyond the scope of this paper. However, in this section we review just a few of the most relevant papers related to our research.

Shape grammars were first used for representing architecture by Stiny [15]. In general, they define the replacement of lower detail items as well as rules to add, scale, translate and rotate shapes. Applied to scenario generation, components within the scenario can be altered by performing operations defined within the grammar. Within shape grammars, rules are applied sequentially (as is typical in a grammar-based system such as this).

Lindenmayer systems (L-systems) use formal grammars to define how components are altered [16]. Similar to shape grammars, L-systems are defined by a set of variables, a set of constants, a start state of the system and a set of production rules. However, one major difference is that L-systems apply the production rules repetitively in parallel rather than sequentially (serially) as in shape grammars. L-systems use all production rules that match at each derivation step and trigger them simultaneously.

7. FUNCTIONAL L-SYSTEMS

Functional L-systems (known as FL-systems) are an extension of L-systems [17]. The primary difference is that FL-systems use terminal *functions* whereas L-systems use the traditional terminal symbol. The terminal functions can be executed during the rewriting process and can provide side-effects during the process. For example, this allows creation of objects or evaluation of decisions at each step in the rewriting process. Marvie et al. use this approach to create a scene graph of a building scene [17].

Both L-Systems and FL-Systems can be thought of as "growth" approaches where the components are refined over time. Müller states that "parallel grammars like L-systems are suited to capture growth over time" whereas systems such as shape grammars with "a sequential application of rules allows for the characterization of structure" [18]. Structure approaches create a generic representation and then refine it.

A question in our research is whether generic scenarios can be generated and then refined, and how the refinement process would be procedurally governed. Therefore, we are pursuing a FL-System approach for scenario generation as growth of scenarios seems to better fit our conceptual model.

FL-systems provide a technique that allows the creation of scenario elements based on the terminal symbols in the grammar. In addition, the terminal functions allow hierarchical elements to be created to fulfill the scenario requirements. For example, a series of vignettes can be assembled together into a larger scenario using this approach.

8. SCENARIO GENERATION

Recall that some of the facets of the scenario described earlier may contain *requirements*. These are components that the facet requires but is not defined inherently within the facet itself. For example, one augmentation may add an additional target without necessarily defining the type or position of that target.

Our technique for semi-automatic scenario generation uses FL-Systems as the procedural system. The rules are domain-dependent and are built to satisfy the requirements of the set of training objectives contained within that domain. We use the rules to create elements within the scenario and to more intelligently resolve the requirements given by the facets.

For example, adding the "additional target" augmentation would cause a "Target" requirement needing to be resolved. The FL-system addresses these unresolved elements. Specifically, the unresolved elements map into symbols of our grammar. The grammar takes the form of:

<predecessor> : <condition> → <successor> : <probability>

where the <predecessor> is replaced by the <successor> under the probability <probability> if the <condition> is true. This basic structure is the same used by Müller [18].

As a simple example, consider the following set of rules:

Rule 1:	{SCENARIO}	:	→ {TO1}	: 1.0
Rule 2:	{TO1}	:	→{TARGET}{A}{OBS}	: 1.0
Rule 3:	{A}	:	→ {artillery}{POSITION}	: 1.0
Rule 4:	{OBS}	:	→ {observer}{POSITION}	: 1.0
Rule 5:	{TARGET}	:	→{tank}{POSITION}	: 0.5
Rule 6:	{TARGET}	:	→ {apc}{POSITION}	: 0.5
Rule 7:	{POSITION}	:	→{position}	: 1.0

Symbols are surrounded by braces. Those in all capital letters represent non-terminals and those in all lowercase represent the terminal functions. Here we have a simple set of rules for the creation of a target, an observer of that target and some artillery unit to shoot at that target. Rule 2 lists the basic components needed by Training Objective 1. Rules 3 and 4 have terminal functions that will cause the creation of each respective entity (Rule 7 consolidates the position selection by calling that respective terminal function). Rules 5 and 6 show how probability can add variety by selecting different target types. Obviously, the set of rules can be made much more complex to provide even greater capabilities and variety than this simple example can show.

The chief advantages of FL-systems in this application are two-fold. First, by using terminal functions as opposed to terminal symbols, the application has access to higher-level reasoning in that the functions can have some advanced computation built within them. Second, the terminal functions allow postponing resolving requirements, which allows the basic rule system to be built with a fewer number of rules.

The limitations of FL-systems include the additional work necessary to author the rule systems and the need to write the terminal functions themselves. However, both limitations are minimal in that each is only performed once per training domain. When another training domain is desired (a rehabilitation scenario, for example), then a new set of rules and terminal functions must be written.

We have built our system into a tool known as the Procedural Yielding Techniques and Heuristics for Automated Generation of Objects within Related and Analogous Scenarios, or PYTHAGORAS. It is built as a "scenario generation engine" in that it provides the core capabilities needed to build scenario generation applications (much like a game engine allows the creation of individual games). This is important in order to provide the ability for building scenarios for different domains.

We have built our first application, a scenario generation system for Fire Support Teams known as COGS, on top of this engine. Figure 2 shows a snapshot of COGS in manual use mode showing the facet library on the left, the scenario editor window in the middle and the list of requirements (some satisfied so far; others not) on the right. A status window in the lower-left shows the current state of building up the scenario.

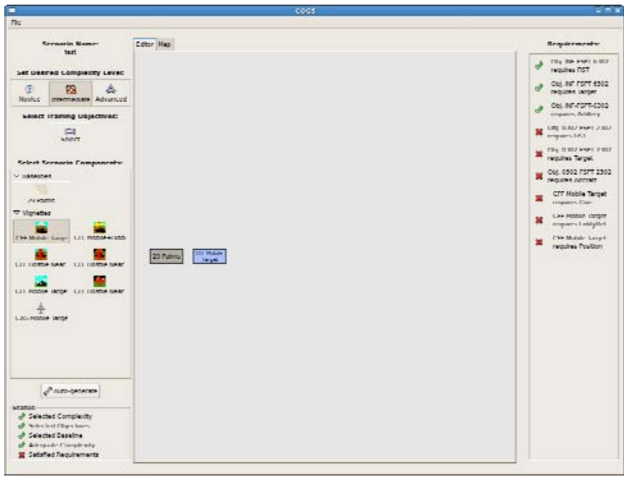

Figure 2. Snapshot of COGS application

The automated mode consists of a single button "Auto-generate" that processes these steps automatically. The FL-System is used to choose scenario facets taking into account the scenario complexity desired and variety in facet selection. The terminal functions of the FL-System become particularly important when satisfying the "requirements" of each scenario facet. The extra capability provided from a function (as compared to a symbol) allows the application to more intelligently choose parameters (entity types and position, for example) as necessary.

9. CONCLUSION

In this short paper, we have reviewed our previous work in defining a scenario and then presented our work in procedural generation of scenarios built around this definition. We use Functional L-systems for our system as the use of terminal functions provides the additional power we need to create elements within the scenario and satisfy requirements of those elements. Our work is built into an engine known as PYTHAGORAS and our first application, COGS, which focuses on building scenarios for training of Fire Support Teams. FL-Systems provide a powerful mechanism for resolving the required parameters of scenario facets.

10. ACKNOWLEDGMENTS

This work is supported in part by the Office of Naval Research Grant N0001408C0186, the Next-generation Expeditionary Warfare Intelligent Training (NEW-IT) program, and in part by the National Science Foundation Grant IIP0750551. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ONR or the US Government. The US Government is authorized to reproduce and

distribute reprints for Government purposes notwithstanding any copyright notation hereon.

11. REFERENCES

[1] Forward Observer PC Simulator (FOPCSIM). Retrieved on April 23, 2010 from http://www.delta3d.org/article.php?story=20041115153320456

[2] Cannon-Bowers, J. A. and Salas, E. 1998. Team performance and training in complex environments: recent findings from applied research. Current Directions in Psychological Science, 7(3): 83-87.

[3] Ross, K. G., Phillips, J. K., Klein, G., and Cohn, J. (2005. Creating Expertise: A Framework to Guide Simulation-Based Training. In Proceedings of I/ITSEC, Orlando, FL: NTSA.

[4] Salas, E., Rhodenizer, L., & Bowers, C. A. 2000. The design and delivery of crew resource management training: Exploiting available resources. Human Factors, 42(3): 490–511.

[5] Bowers, C., Jentsch, F., Baker, D., Prince, C., and Salas, E. 1997. Rapidly reconfigurable event-set based line operational evaluation scenarios. In Proceedings of the Human Factors and Ergonomics Society, 4, 912–915.

[6] Jentsch, F., Abbott, D., and Bowers, C. 1999. Do three easy tasks make one difficult one: studying the perceived difficulty of simulation scenarios. Proceedings of the 10th International Symposium on Aviation Psychology (Columbus, Ohio).

[7] Jentsch, F., Irvin, J., and Bowers, C. 1997. Differences in situation assessment between experts and prospective first officers. In Proceedings of the 9th International Symposium on Aviation Psychology (Columbus, Ohio). 1228–1232.

[8] Hall, R. J. 1998. Explanation-based scenario generation for reactive system models. In Proceedings of the 13th Conference on Automated Software Engineering (Honolulu, HI).

[9] Pfefferman, M. W. 1993. A Prototype Architecture for an Automated Scenario Generation System for Combat Simulations. Thesis, Air Force Institute of Technology (Air University).

[10] Erraguntla, M., Benjamin, P. C., and Mayer, R. J. 1994. An architecture of a knowledge-based simulation engine. In the Proceedings of the 1994 Winter Simulation Conference (San Diego, CA). Society for Computer Simulation International, 673-680.

[11] KBSI . 2008. FEAST: Framework for Enabling Adaptive Scenario Generation for Training. Retrieved from http://www.kbsi.com/.

[12] Benjamin, P. 2008. Knowledge based simulation: Methods and enabling technology. Tutorial presented at I/ITSEC 2008 (Orlando, FL).

[13] Martin, G. A., Schatz, S., Hughes, C. E. and Nicholson, D. 2010. What is a Scenario? Operationalizing training scenarios for automatic generation. In Proceedings of Applied Human Factors and Ergonomics (Miami, FL).

[14] Tomizawa, H. and Gonzalez, A. 2007. Automated scenario generation system in a simulation. In Proceedings of Interservice/Industry Training, Simulation and Education Conference (Orlando, FL).

[15] Stiny, G. 1975. Pictorial and Formal Aspects of Shape and Shape Grammars. Birkhauser Verlag.

[16] Prusinkiewicz, P. and Lindenmayer, A. 1990. The algorithmic beauty of plants. Springer-Verlag.

[17] Marvie, J., Perret, J. and Bouatouch, K. 2005. The FL-system: A Functional L-system for procedural geometric modeling. The Visual Computer. 21, 5, 329–339.

[18] Müller, P., Wonka, P., Haegler, S., Ulmer, A. and Van Gool, L. 2006. Procedural modeling of buildings. ACM Transaction on Graphics (SIGGRAPH Proceedings). 25, 3, 614-623.