Appears in Proceedings of the 2008 IEEE Aerospace Conference, Big Sky, Montana.

Rapid Simulation Construction

Jeremy Ludwig, Ryan Houlette, and Dan Fu
Stottler Henke Associates
951 Mariner's Island Blvd, Suite #360
San Mateo, CA 94404
650-931-2700
ludwig, houlette, fu @ stottlerhenke.com

Abstract—In the course of building four low-cost and fairly distinct desktop training simulations for the Air Force's Air University and a simulated control system for NASA's International Space System, we have worked to develop in-house standard practices and a toolset for rapid construction of training simulations. We present here this process and toolset and discuss its key strengths and shortcomings in the context of the simulations we have built using it. We focus in particular on two aspects of development that we found to be pivotal: scoping the level of fidelity for the simulation logic, and designing and constructing cost-effective simulation user interfaces that achieve instructional goals. These two aspects will be described in the context of the SimVentive and SimBionic toolsets, both of which are freely available for use by NASA or any other government agency.

TABLE OF CONTENTS

1. INTRODUCTION

Desktop PC simulations are increasingly finding a place in the classroom as an effective way to provide immersive experiential training. While significantly less expensive than dedicated-hardware simulators, these simulations still tend to be costly and time-consuming to develop, in large part because training simulations are often built as custom "one-offs" with little shared infrastructure or content. Improving simulation development practices and tools to enable true rapid development has obvious cost benefits and will permit more widespread adoption of training simulations. [1][2]

In the course of building four low-cost and fairly disparate desktop training simulations for the Air Force's Air University and a simulated control system for NASA's International Space System, we have worked to develop in-house standard practices and a toolset for rapid construction of training simulations. We present here this process and toolset and discuss its key strengths and shortcomings in the context of the simulations we have built using it. We focus in particular on two aspects of development that we found to be pivotal: scoping the level of fidelity for the simulation logic, and designing and constructing cost-effective simulation user interfaces that achieve instructional goals. These two aspects will be described in the context of the SimVentive and SimBionic toolsets, both of which are freely available for use by NASA or any other government agency.

2. SIMVENTIVE

The SimVentive software toolset enables instructional designers and subject matter experts to author computer-based simulations and serious games without programming. This software provides a visual authoring environment designed to simplify and streamline the development of simulations by moving away from the traditional text-based programming interface. Underneath this visual interface lies a sophisticated simulation engine capable of deploying complex, highly-interactive scenarios both on the desktop as well as via the web browser.

SimVentive is composed of two main components—a Scenario Editor to create simulations, and a Scenario Player for playing those simulations. Figure 1 shows the relationship between authoring and playing simulations.

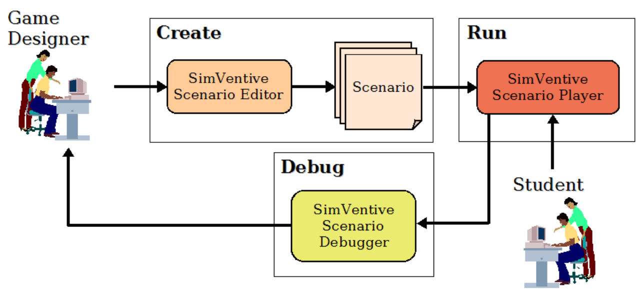

Game Designer
Create
SimVentive Scenario Editor
Scenario
Run
SimVentive Scenario Player
Debug
SimVentive Scenario Debugger
Student

Figure 1: SimVentive Usage Overview

On the left is an instructor who uses the Scenario Editor to build a simulation scenario. The resulting scenario is then loaded by the Scenario Player, which enables the student on the right to play the simulation.

The SimVentive Scenario Editor allows an instructor or subject matter expert to visually construct simulations out of

1
[1] 1-4244-1488-1/08/$25.00 ©2008 IEEE.
[2] IEEEAC paper #1206, Version 3, Updated 2007:11:28

a set of basic building blocks, as shown in Figure 2. An **entity** represents a single simulated object in the simulation. It may be concrete (like an airplane) or abstract (like a communication network). Each entity can have one or more **attributes** that describe its current state – for example, an airplane might have a "speed" attribute with the value "300."

A **rule** is a piece of game logic that defines how some aspect of the simulation responds to the passage of time or to player actions. Rules describe how the simulated world works and define the gameplay for the scenario. They provide the "AI" for entities and computer-controlled players. Rules can also provide feedback and hints to the player.

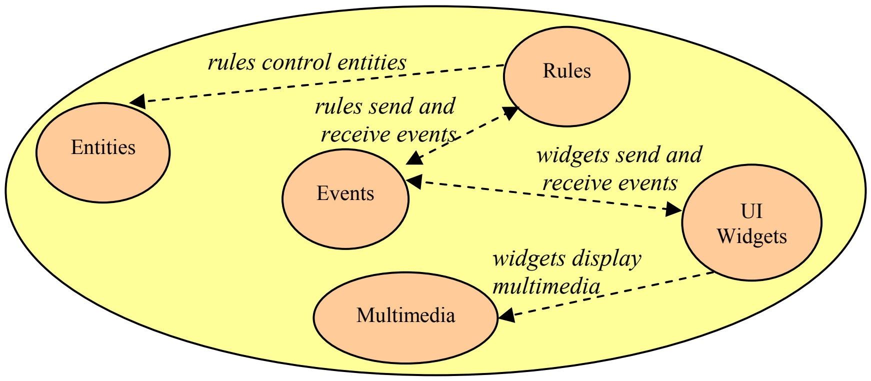

rules control entities
Rules
rules send and receive events
Entities
Events
widgets send and receive events
UI Widgets
widgets display multimedia
Multimedia

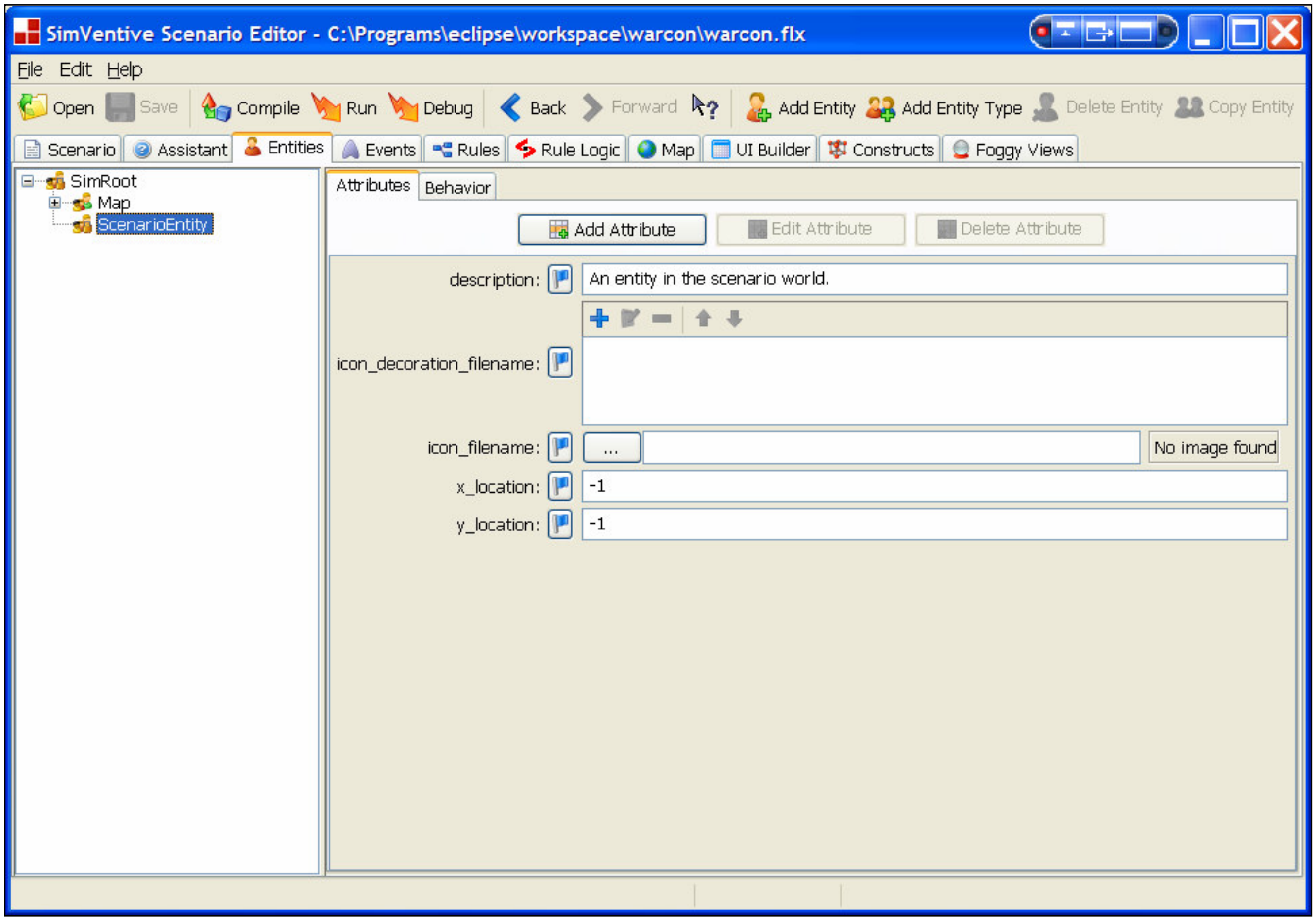
Figure 2 Scenario Specification

Widgets are components such as maps, HTML pages, tables, and buttons that make up the scenario's user interface. Widgets can make use of many kinds of multimedia files, ranging from bitmapped and vector images to audio files to 3D models to video. When the player plays a SimVentive scenario, he or she interacts with the scenario's widgets to perform actions in the simulation.

Events are the "glue" that binds the pieces of the scenario together. Each time the player interacts with a widget, it generates an event. Rules and other widgets can listen for and respond to those events. Rules can also generate their own events, to which other rules and widgets can in turn respond.

Scenario editing is performed by using a number of different editor panes, such as the pane shown in Figure 3. We focus on only the core editors below.

The **scenario pane** allows the author to specify descriptive text that will be displayed at the start of the scenario. It also enables the author to create teams, roles, and player positions within the game for multi-player games. In addition, the Time Settings tab allows the author to specify the game type, either turn-based or real-time, and the various time settings. The **assistant pane** informs the author of the overall status of the scenario and the current open tasks. It also provides a number of wizards that step the author through complicated procedures. The **entity pane** is used to create and edit the *entities* that make up a scenario. This involves the construction of entity types, the attributes associated with a type, and entity instances. Each entity can also have a control rule associated with it, which uses SimBionic to drive the behavior of the entity in the game or simulation. The **events pane** is used to create, edit, and manage the *events* that form the "glue" of the scenario. The **rule pane** is used to create and edit the *rules* that define the gameplay of a scenario, where each rule has a type (immediate, persistent, control) and a list of triggering events. The **rule logic pane** is used to graphically specify the logic that defines the behavior of each of these scenario rules. This pane is based on Stottler Henke's SimBionic intelligent simulation behavior technology, discussed in more detail below. The **UI builder pane** is used to author graphical user interfaces for the Scenario Player. *Widgets,* such as a text input field, are placed in a *Canvas* and their *properties*, *methods*, and *interactions* determine much of the game play. The **map pane** allows the author to create and edit maps that will be displayed in the Scenario Player as a map widget. For example, this pane allows the author to set the type of map (pixel, grid, lat/long), paint textures, draw annotations, and place entities in the map.


SimVentive Scenario Editor - C:\Programs\eclipse\workspace\warcon\warcon.flx
File Edit Help
Open Save Compile Run Debug Back Forward Add Entity Add Entity Type Delete Entity Copy Entity
Scenario Assistant Entities Events Rules Rule Logic Map UI Builder Constructs Foggy Views
SimRoot
Map
ScenarioEntity
Attributes Behavior
Add Attribute Edit Attribute Delete Attribute
No image found

description: An entity in the scenario world.
icon_decoration_filename:
icon_filename: ...
x_location: -1
y_location: -1

Figure 3 SimVentive Editor Pane

3. SIMBIONIC

The rule logic pane is used to create the logic for the underlying models in the simulation. This pane makes use of the SimBionic modeling architecture [Fu, Houlette, & Ludwig, 2007]. The core of SimBionic is a visual authoring tool that allows users to draw flow chart-like diagrams that specify sequences of conditions and actions. Actions are something the modeled entity can do and are represented by rectangular nodes. Ovals indicate conditional nodes that are true or false based on the agents perception or its internal variables. Control flows from one action to another by directed connectors.

Figure 4 shows an example SimBionic (SB) behavior. Execution starts in the start node (green) at the top. The first transition (labeled 1) is then tested. If the agent is within range of the solider, the action *MoveFrom* is taken. Otherwise, the second transition from the start node is taken which performs the action *MoveTowards* action. The behavior ends in the final node (red). A hierarchy of behaviors can be built as an action may be a primitive action (as shown in the example) or another behavior.

The SimBionic language supports a number of features in addition to the basic hierarchal task network model such as local and global variables, direct use of Java classes (i.e. using methods as actions and conditions), ordered transitions, polymorphic behavior, blackboard communication, behavior interrupts, and exception handling.

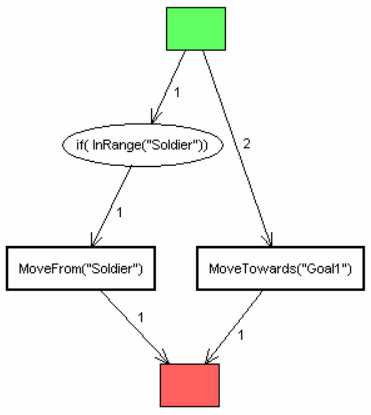


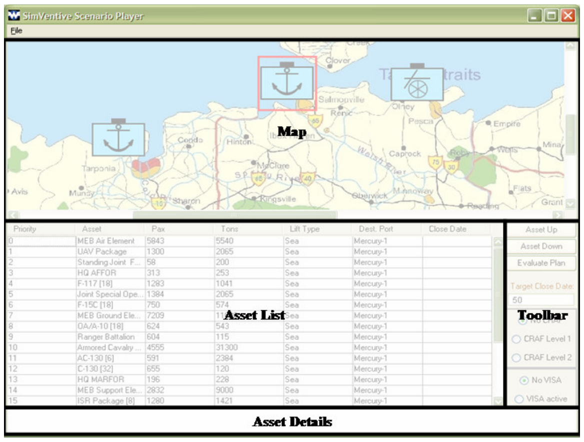
Figure 4 Sample SimBionic Behavior

4. Training Scenarios

In this section we summarize five training simulations built with the SimVentive tool. The first four were built for the Air Force's Air University and the last was built for NASA. Of the five simulations, only an overview is provided of the first three. The last two simulations, in aerospace domains, are discussed in greater detail.

ACSC: Logistics Scenario

This scenario is a training simulation for logistics mobility planning. The simulation provides an example of how a Joint Force Commander's (JFC) staff needs to understand the tradeoffs between requirements and prioritization of scarce lift and logistics assets. Students grapple with the challenges of limited lift capability while providing force flow (military capabilities) to meet JFC requirements in a notional scenario.



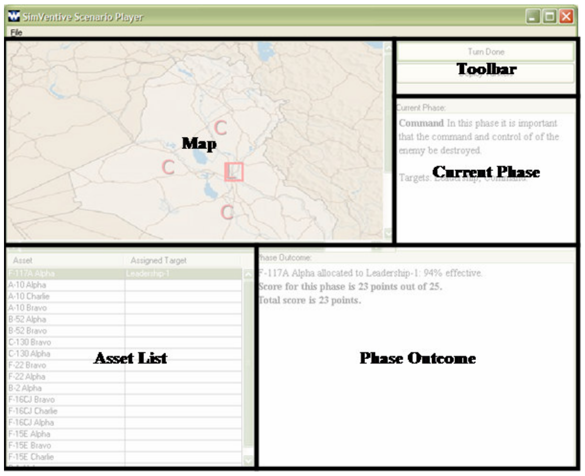
Priority	Asset	Pax	Tons	Lift Type	Dest. Port	Close Date
0	MEB Air Element	5843	5540	Sea	Mercury-1	
1	UAV Package	1300	2065	Sea	Mercury-1	
2	Standing Joint F...	58	200	Sea	Mercury-1	
3	HQ AFFOR	313	253	Sea	Mercury-1	
4	F-117 [18]	1283	1041	Sea	Mercury-1	
5	Joint Special Ope...	1384	2065	Sea	Mercury-1	
6	F-15C [18]	750	574	Sea	Mercury-1	
7	MEB Ground Ele...	7209	[illegible]	Sea	Mercury-1	
8	OA/A-10 [18]	624	543	Sea	Mercury-1	
9	Ranger Battalion	604	115	Sea	Mercury-1	
10	Armored Cavalry ...	4555	31300	Sea	Mercury-1	
11	AC-130 [6]	591	2384	Sea	Mercury-1	
12	C-130 [32]	655	120	Sea	Mercury-1	
13	HQ MARFOR	196	228	Sea	Mercury-1	
14	MEB Support Ele...	2832	9000	Sea	Mercury-1	
15	ISR Package [8]	1280	1421	Sea	Mercury-1	

Figure 5 ACSC UI

Overall, this application demonstrates the importance of integrating logistics requirements with operational planning objectives. The scenario is designed to be a one and a half-hour student-participation exercise in which the instructor will introduce and close the exercise. Students are provided with a list of forces that must all be prioritized, and they prioritize and match these forces with available lift assets (air and sea). They then use their list to determine if JFC objectives and phasing can be met within the given timelines, and should determine that the JFC requirements are not supportable.

AFOATS: Targeting Scenario

This training simulation puts the player in the role of a military commander. All US efforts have failed to bring a peaceful resolution. The US will use military force to achieve national security objectives. To achieve supremacy, the player will task air and space assets appropriately and see their effects throughout the war. Roughly, the game will go through three phases: an initial strategic attack, air dominance, and ground support. This training simulation puts the player in the role of a military commander.



Current Phase:

Command In this phase it is important that the command and control of of the enemy be destroyed.

Targets:

Asset	Assigned Target
F-117A Alpha	Leadership-1
A-10 Alpha	
A-10 Charlie	
A-10 Bravo	
B-52 Alpha	
B-52 Bravo	
C-130 Bravo	
C-130 Alpha	
F-22 Bravo	
F-22 Alpha	
B-2 Alpha	
F-16CJ Bravo	
F-16CJ Charlie	
F-16CJ Alpha	
F-15E Alpha	
F-15E Bravo	
F-15E Charlie	

Phase Outcome:

F-117A Alpha allocated to Leadership-1: 94% effective.
Score for this phase is 23 points out of 25.
Total score is 23 points.

Figure 6 AFOATS UI

SAASS: Instruments of National Power Scenario

This training simulation puts the player in the role of the Air Theater Commander in the Middle East region. The United States is pursuing the cessation of a country's nuclear weapon program development. The player must make and implement strategic decisions to produce the most favorable outcome possible. This outcome, which can be produced by ceasing operations at any time, is measured in terms of the diplomatic concessions that are available under the prevailing conditions at that time. These conditions include military dominance, viability of the country's nuclear weapons program, coalition status, and political will as reflected in US public opinion. Operations may also cease spontaneously at any time as an act of US congress. The game is turn-based, meaning that most actions only take effect "overnight," when the day is advanced by the player.

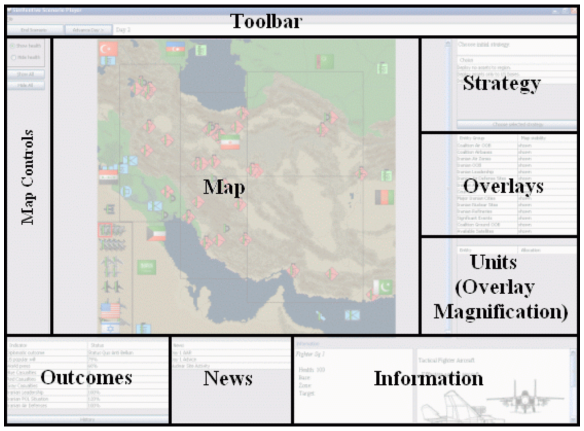
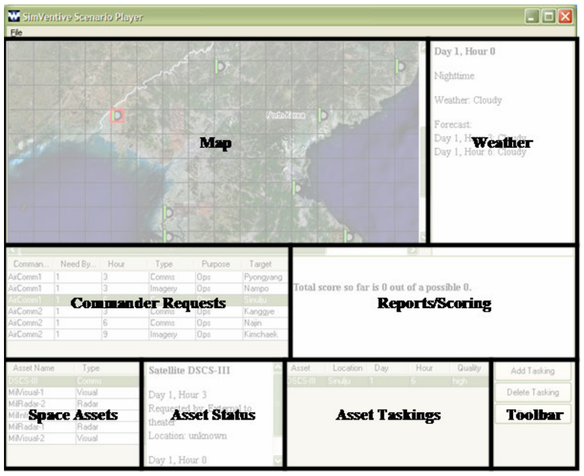


Figure 7 SAASS UI

SOC: Satellite Tasking Scenario

In this training simulation, the student is in charge of planning the use of and making requests for information and services provided by space-based platforms, in the context of a theater-level conflict. The student's decisions are driven by the needs of simulated friendly commanders and constrained by limited availability of resources, so the student must determine the optimal use of space capabilities given the current situation. The outcome of the simulated conflict, which will unfold over time, will provide feedback on the merit of the student's choices. Note that due to the highly classified nature of satellite operations, this scenario is largely notional and as such is intended to simply give the general flavor of satellite operations. It should not be assumed to be correct in any specific details.



Day 1, Hour 0

Nighttime

Weather: Cloudy

Forecast:
Day 1, Hour 3: Cloudy
Day 1, Hour 6: Cloudy

Comman...	Need By...	Hour	Type	Purpose	Target
AirComm1	1	3	Comms	Ops	Pyongyang
AirComm1	1	3	Imagery	Ops	Nampo
AirComm1	1	[illegible]	[illegible]	[illegible]	Sinuiju
AirComm2	1	3	Comms	Ops	Kanggye
AirComm2	1	6	Comms	Ops	Najin
AirComm2	1	9	Imagery	Ops	Kimchaek

Total score so far is 0 out of a possible 0.

Asset Name	Type
DSCS-III	Comms
MilVisual-1	Visual
MilRadar-2	Radar
MilInf[illegible]	[illegible]
MilRadar-1	Radar
MilVisual-2	Visual

Satellite DSCS-III

Day 1, Hour 3
Requested by: External to theater
Location: unknown

Day 1, Hour 0

Asset	Location	Day	Hour	Quality
DSCS-III	Sinuiju	1	6	high

Figure 8 SOC UI

There are eight main areas in the SOC scenario's user interface.

- The **map** shows the theater of operations with targets of interest to the commanders. Each target has a health bar indicating its observed status. A green bar indicates full health; yellow indicates some damage; red indicates severe damage.
- The **weather** area displays the current time, whether it is day or night, and the weather. It also shows a weather forecast for the next two turns. This information is useful in deciding which satellite asset will be most effective at a given time.
- The **commander requests** area shows all of the current requests for space capabilities from theater commanders. Each request has an originating commander, a "need by" time, a type (either Comms or Imagery), a purpose (either "Ops," indicating pre- or during-strike needs, or "BDA", indicating post-strike BDA), and a target area. Clicking on a request will highlight the target area on the map.
- The **space assets** area shows the space assets that can be tasked by the player. Each asset has a name and a type. Selecting an asset will display its current status and tasking in the Asset Status window.
- The **asset status** area shows detailed information about a single asset. This information includes the overflight time of the asset where relevant as well as all of the currently-scheduled taskings for that asset. Note that taskings may arrive from sources external to the theater as well as from the player.
- The **asset tasking** area shows all of the satellite taskings that the player currently has scheduled. Each tasking lists the asset being tasked, the target location, the scheduled time, and the desired quality of service. Selecting a tasking will highlight the associated request and asset. The scheduled time and quality of service for a tasking can be modified by clicking on the appropriate cell and choosing the desired value from the combo box. Invalid values will generate a warning message from the scenario.
- The **reports/scoring** area shows the result of taskings and strikes from the previous turn. It displays the extent to which the player has satisfied each commander request as it comes due. It also gives the player's score.
- The **toolbar** area contains the "Add Tasking", "Delete Tasking", and "End Turn" buttons. To add a tasking, select the asset to task and the request to satisfy and click the "Add Tasking" button. To delete an existing tasking, select it and click "Delete Tasking." Finally, pressing the "End Turn" button will cause the scenario to advance the simulation time, executing any scheduled taskings and strikes.

There are six main principles that the students are expected to learn in this simulation:

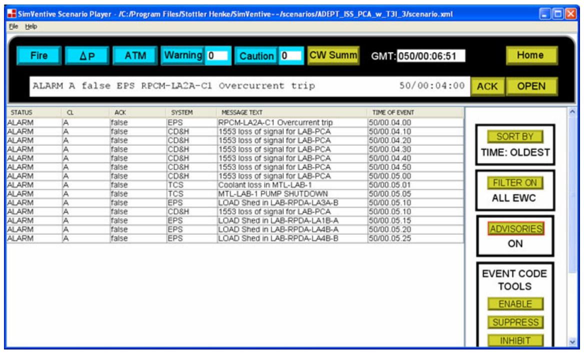
1. Space capabilities have the power to enhance operational effectiveness. In this scenario, strike effectiveness is directly tied to the satisfaction of commanders' requests for comms and imagery. An unsatisfied or poorly-satisfied request will lead to an ineffective strike.
2. Weather has a drastic impact on some types of satellites. Communications satellites are somewhat affected in this scenario, infrared satellites are strongly affected, and visual satellites are severely impacted.
3. Available daylight has a major impact on some kinds of imagery satellites, notably visual and, to a lesser extend, infrared.
4. Imagery satellites must be over the target area in order to collect imagery. This typically happens only once or twice a day, which limits the times at which collection can be performed.
5. Space assets are high-demand resources shared among many users, some of whom are higher-priority than the theater commanders. The player must contend in the scenario with requests from external sources.
6. After a strike, it is important that bomb damage assessment (BDA) be performed in order for the commander to know how effective the strike was. Without this information, the commander does not know whether to devote more effort to destroying the target. In this scenario, the observed status of a target does not change until BDA is performed.

ADEPT: Space Systems Diagnosis Intelligent Tutoring System

We have nearly completed development of a simulation-based intelligent tutoring system that trains astronauts and flight controllers to apply their understanding of systems operations to diagnose and recover from problems. The system simulates mission operations software, so students can apply their understanding of space systems and their interactions to investigate, diagnose, and recover from hypothetical problems. For example, one scenario challenges flight controller students to diagnose a problem by reasoning about the interactions among the electrical, thermal control, and environmental control systems on board the International Space Station.

SimVentive was used to construct the models and user interfaces required to simulate mission operations software. All of the screens for this simulation are created by using the Scalable Vector Graphics (SVG) widget included in the UI builder pane. Each SVG Widget displays interactive, 2-dimensional, graphics.

Parts of the ADEPT simulation resemble actual mission operations systems used by flight controllers. This screen shows a listing of cautions and warnings at the beginning of the scenario.


SimVentive Scenario Player - /C:/Program Files/Stottler Henke/SimVentive--/scenarios/ADEPT_ISS_PCA_w_T3I_3/scenario.xml

ALARM A false EPS RPCM-LA2A-C1 Overcurrent trip 50/00:04:00

STATUS	CL	ACK	SYSTEM	MESSAGE TEXT	TIME OF EVENT
ALARM	A	false	EPS	RPCM-LA2A-C1 Overcurrent trip	50/00.04.00
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.04.10
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.04.20
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.04.30
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.04.40
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.04.50
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.05.00
ALARM	A	false	TCS	Coolant loss in MTL-LAB-1	50/00.05.01
ALARM	A	false	TCS	MTL-LAB-1 PUMP SHUTDOWN	50/00.05.05
ALARM	A	false	EPS	LOAD Shed in LAB-RPDA-LA3A-B	50/00.05.10
ALARM	A	false	CD&H	1553 loss of signal for LAB-PCA	50/00.05.10
ALARM	A	false	EPS	LOAD Shed in LAB-RPDA-LA1B-A	50/00.05.15
ALARM	A	false	EPS	LOAD Shed in LAB-RPDA-LA4B-A	50/00.05.20
ALARM	A	false	EPS	LOAD Shed in LAB-RPDA-LA4B-B	50/00.05.25

Figure 9 Adept Caution/Warning Screen

Other parts of the ADEPT simulation do not resemble any current mission operations software. Instead, they use graphical display techniques to help the student visualize the physical and functional components of the Space Station and how they relate to one another. The interactive screen in Figure 10 is inspired by drawings in NASA technical documents that show an overview of the major components of the Space Station.

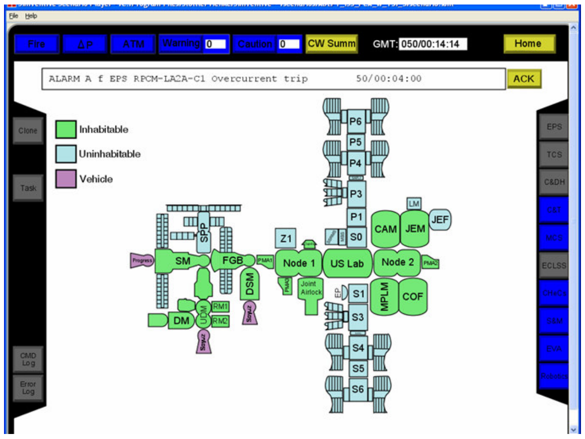

ALARM A f EPS RPCM-LA2A-C1 Overcurrent trip 50/00:04:00

Figure 10 ISS Systems

The next screen, Figure 11, shows an interactive schematic of the pipes, pumps, valves, and heat loads that are part of the thermal control system for the US Lab within the Space Station.

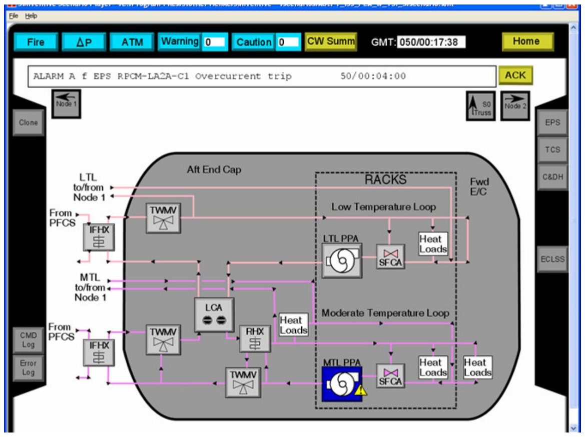

ALARM A f EPS RPCM-LA2A-C1 Overcurrent trip 50/00:04:00

Figure 11 ISS Schematics

5. Discussion

The process used in creating these fives simulations was quite similar. The first step is to determine what the objectives of the simulation are. For example, in the satellite tasking scenario, the student should learn that weather has a drastic impact on some types of satellites. Based on these objectives, the author can make a number of decisions. These include defining the nature of time in a game – real time, turn-based, or something in between. The author also determines what information needs to be presented to the user and what types of actions the user needs to be able to take. With this set of objectives, the desired game play, the display information, and the user actions, the author then needs to implement the simulation. This involves creating the user interface the student will interact with and defining the underlying models in the simulation. In building these five simulations, we found these final two aspects of development to be pivotal: designing and constructing cost-effective simulation user interfaces and scoping the level of fidelity for the simulation logic that achieves instructional goals.

With respect to the user interface, the author outlines a) what the user needs to know and b) what actions the user needs to take. However, there are many mappings from a) and b) to a set of inter-connected user interface widgets. To quickly create a simulation user interface, we found the UI needed to use the simplest, most flexible, widgets available. For example, in the first four simulations we made significant use of HTML display widgets to show information to the user. HTML is very flexible in what it displays, easy to create, and easy to change. In the fifth simulation, that of the international space systems, HTML image maps were not able to support the type of UI required. Instead, SVG images were used to create interactive interfaces that both display information and allow the user to perform the necessary actions. This UI leveraged the ability of existing tools that construct high-quality interactive images to create the simplest interface that met the simulation objectives.

Scoping the level of fidelity for the simulation logic that achieves instructional goals is also of paramount importance. Generally we found that the finer the granularity the more complex the model. To rapidly create a simulation, it is imperative to model only that which is absolutely necessary to support the training goals. One example is the granularity of time in the simulation. Do you model hours, days, weeks, months? Granularity of entities is another dimension. Do you model soldiers, fire teams, platoons, brigades, etc? The fewer the types of entities you're modeling, the less time will be required to author corresponding behavior logic. Domain is another way to scope. In military terms, this would be restricting to only a single service or mission type within a service. For example, the SAASS scenario doesn't model the naval aspect of the conflict at all, and has only a very limited model of ground combat and space operations. The focus of this simulation is on the air domain.

In the satellite tasking simulation we could have created a model of satellite orbiting and used this to control the location of the satellites in the scenario. Instead, we simplified the scenario considerably by just specifying the time at which a satellite would be over a target. In this same simulation, we did construct a day/night modeling system. In this case, the logic was easy to implement and there was not a simpler alternative. Again, the lesson here is to simplify as much as possible while still meeting the training objectives.

As a more in-depth example, the SAASS scenario works, spatially, much like the board game RISK. Entity symbols inhabit sectors of action or influence, or rest on top of particular targets, merely to demonstrate a correspondence. There are also some other entity symbols, outside the user's control, to visualize army units fighting their way across the countryside. Originally we had planned much more freedom of action, similar to that found in a hex-map game, but those degrees of freedom collapsed when we found that the training objectives were more abstract. That is, we wanted to evaluate the user's decisions with respect to whether to deploy assets to certain kinds of bases, and what to target with what assets. In the end this is just a many-to-many iterative allocation game that could have been implemented in an Excel spreadsheet. However, the visual style of the training game suggests clues to the mechanics of interest: nuclear reactors near cities, bases in foreign countries, air presence across many regions, etc. Real spatial mechanics, like transit costs, don't exist in SAASS.

Evaluation

In construction of these five training scenarios, we found that it was certainly easier to build the simulations in SimVentive than it would have been to custom-code each one in Java. Excepting ADEPT, none of the simulations took more than a couple of weeks to build, and most took only a couple of days. In our opinion, this is far less than the corresponding coding effort would have been by at least a factor of two. We found that the more complex the training simulation, the bigger the win, since at some point a lot of the infrastructure provided by SimVentive would have been required by the simulation.

Limitations

The main limitation that we found in developing these scenarios was the over-reliance on the event model. The vast majority of events in existing SV scenarios are devoted to managing low-level interactions between rules and widgets. These events generally have little domain significance and are forced on the author by the requirement that all interaction between rules and widgets be mediated by events. A widget interaction cannot directly invoke a rule, nor can a rule directly change a widget's state by invoking a widget method. This amounts to a "tax" on the scenario author, who must go through a number of extra steps for each rule-widget interaction he wishes to specify. It also muddies the scenario with many events that are purely implementation details.

Related Tools

While a review of the diverse array of simulation / serious game construction toolsets is beyond the scope of this paper, it is important to note that SimVentive sits in the middle between the heavyweight simulation tools for modeling complex systems or that rely heavily on 3-D graphics and the lighter weight simulation tools that focus on being very author-friendly. For small scale training games, such as those developed in SimVentive, our limited survey found that the primary competitors in this field are Flash and Captivate, both by Macromedia.

6. Conclusion

While constructing the SimVentive tool and creating five distinct simulations, we found that two aspects of development that are pivotal: scoping the level of fidelity for the simulation logic, and designing and constructing cost-effective simulation user interfaces that achieve instructional goals. In this paper we provide an overview of the SimVentive tool and briefly describe five training simulations that have been built with SimVentive. Following this, we discuss the lessons learned while creating the scenarios.

Building these simulations has also provided some direction for the future work on the SimVentive tool. Our main focus right now is on making the tool more accessible to non-programmers. To achieve these, one aspect of SimVentive that we are trying to simplify is the event model. As many events are only used for a single purpose, the simulation quickly gets cluttered with a large number of events. The second aspect that we are working on is identifying complex tasks shared by a number of scenarios, in order to create automated wizards that will complete these tasks for the author.

References

[1] Fu, D., Houlette, R., & Ludwig, J. (2007). *An AI Modeling Tool for Designers and Developers*. Paper presented at the IEEE Aerospace Conference, Big Sky, MT.

Biographies

Jeremy Ludwig is the technical lead on the SimVentive simulation construction toolkit. He is also involved in the continuing development of the SimBionic behavior modeling tool and currently directing the addition of a machine learning component. His research areas include machine learning, behavior modeling, intelligent training systems, and expert systems. Mr. Ludwig joined Stottler Henke in the fall of 2000 after completing his M.S. in Computer Science at the University of Pittsburgh with a concentration in Intelligent Systems.

Ryan Houlette is a project manager and lead software engineer at Stottler Henke Associates. He holds an M.S. in Computer Science (Artificial Intelligence) from Stanford University. He has participated in the development of a wide range of AI systems, with a particular focus on autonomous agents and intelligent interfaces. Mr. Houlette is lead architect of the SimBionic behavior modeling tool and product manager for the SimVentive simulation construction toolkit. He is also an editor for the AI Game Programming Wisdom book series.

Dan Fu is a group manager at Stottler Henke Associates. He joined nine years ago and has worked on several artificial intelligence (AI) systems including AI authoring tools, wargaming toolsets, immersive training systems, and AI for simulations. Dr. Fu is the principal investigator for SimBionic, which enables users to graphically author entity behavior for a computer simulation or game. Dr. Fu holds a B.S. from Cornell University and a Ph.D. from the University of Chicago, both in computer science.