SmartObjects and Agents for Autonomous High Population Simulated Environments

Bill Blank, Gedalia Pasternak,
Adam Crane, Alex Broadbent
DI-Guy
Waltham, MA
bblank@diguy.com, gedalia@diguy.com
arcane@diguy.com, alex@diguy.com

ABSTRACT

Populating simulated virtual environments with hundreds and thousands of lifelike interactive human characters requires specialized techniques to create scalable results while containing authoring labor costs. Characters need to perform kinetic, non-kinetic, and "pattern of life" behavior. Traditional animation techniques do not scale well as entity count increases, resulting in unsatisfactory agent performance and overly expensive authoring costs.

This paper details techniques for modeling high entity count human behavior to meet the challenge. Humans are modeled as autonomous and semi-autonomous agents with scriptable Hierarchical Finite State Machine (HFSM) AI Minds featuring coroutine-based cooperative multi-tasking. Interactive props use SmartObject technology, packaging objects and intelligence together for increased scalability. SmartObjects are extended to create SmartBuilding aggregates composed of intelligent agents, terrain tagging, buildings, and other SmartObjects into single reusable components enabling faster authoring.

We have successfully applied these technologies to create human behavior solutions for a number of projects including ECO (Enhanced Company Operations) Sim with the Marine Corps Tactics and Operations Group (MCTOG) at Twentynine Palms CA.

These concepts show promise and scalability for quickly authoring civilian "patterns of life" in simulations. We describe herein their utility for modeling complex opfor operations we describe as "human networks". The techniques have also been used to successfully model squad-level friendly tactical behavior. The application of these techniques shows promise for other areas of interest such as logistics-oriented simulation.

ABOUT THE AUTHORS

Bill Blank is the DI-Guy Product manager and an expert in visual human and physics-based simulation. Bill is a graduate of Phillips Academy Andover and received in BS in Computer Engineering from Brown University in 1984.

Gedalia Pasternak is a Senior Software Engineer with DI-Guy focusing on rendering tech and AI solutions for human behavior. Before joining DI-Guy in 2003 Gedalia worked on TV special effects and massively multi-player real-time games. Gedalia received his B.S. in Engineering from the Cooper Union in 1998.

Adam Crane is the Technical Director and co-inventor of DI-Guy, specializing in human motion simulation, AI, software engineering, and 3D graphics. Adam received both BS and MS degrees in Computer Science and Electrical Engineering from MIT in 1993.

Alex Broadbent is the ECO Sim Project Manager and Senior DI-Guy Content Engineer. An expert in 3D geometry, human motion and scenario content, Alex works closely with the US military to ensure DI-Guy is used to its full potential for training. Alex has formerly worked at CNN, ABC Sports, and NECN.

SmartObjects and Agents for Autonomous High Population Simulated Environments

Bill Blank, Gedalia Pasternak,
Adam Crane, Alex Broadbent

DI-Guy

Waltham, MA

bblank@diguy.com, gedalia@diguy.com
arcane@diguy.com alex@diguy.com

INTRODUCTION

There is widespread recognition that today's modern soldiers require sophisticated training if they are to be successful when charged with challenging missions in peacekeeping, counterinsurgency, antiterrorism, and IED defeat operations. Effective training often involves the use of immersive 3D virtual environments. By the nature of the missions and theatres of operation, robust human simulation is a critical component. Typically these missions occur in a mixed force environment including the soldier's unit, other friendly units, other allied forces units, and asymmetric units in the opposing forces. These forces engage against a greater backdrop of a large civilian populace, whose welfare may also be critical to the mission and whose presence affects both parties.

Realistic virtual environments therefore need to support high entity count often lacking in today's simulations. Simulations require realistic characters and animations using artificial intelligence (AI) to autonomously and semi-autonomously control their behavior. Kinetic and non-kinetic behavior, including tactical and strategic action of non-player friendly forces and oppositional forces is required. Large numbers of civilians need to populate the environment, displaying the "pattern of life" of the area, but also being reactive to the ongoing events triggered by the warring forces in their midst. The control and authoring of AI behavior needs to be affordable and scalable.

HSFM Minds and SmartObjects

Creating scenarios where all possible events need to be pre-scripted does not scale well once hundreds or thousands of entities are involved and the number of actionable events and agent-interactive objects increases. To solve this class of problems we've focused on robust implementations of Hierarchical Finite State Machine (HSFM) Agent Minds and SmartObjects. These solutions do not work in a vacuum and rely on other system components for success:

- A low level C++ agent movement and animation toolkit
- A navigation system that supports environment tagging and automated path planning
- A high level scripting language for authoring
- Infrastructure for associating physical objects and the high level logic together
- A terrain-aware global repository of actionable props

Together, we present end users with a collection of intelligent object-oriented agents and interactive props that can be placed into terrains arbitrarily in large numbers. The autonomous agents exhibit sophisticated non-linear behavior, achieve high performance, while providing an authoring framework that reduces upfront and revision cost and keeps complexity manageable.

Estimating the cost of "pattern of life" authoring

Creating high population simulations is challenging both in terms of realism of the human performances and the cost of authoring such performances. If we consider a scenario where there are N virtual characters, P actionable props, and D "destinations of interest", that is, locations such as schools, marketplaces, homes, fields, etc that will be the places individuals navigate between as they go about their daily business, the amount of authoring time, T_A required to individually script each character, script interactions between characters and props, identify each

destination, and create paths for characters to follow between the destinations is roughly on the order:

$$T_A = NxP+D+Nx(D!) \quad (1)$$

(Note that this assumes all entities can use the same navigation information)

If we assume that entities can share pathing information between destinations, the number reduces to:

$$T_A = NxP+D+D! \quad (2)$$

If navigation path planning can be automated, the amount of time is reduced by removing the D! factorial:

$$T_A = NxP+D \quad (3)$$

If the individual civilians are aggregated into groups or "crowds" and strong object-oriented classification of the humans is imposed, then the operation is reduced further to:

$$T_A = CxP+D \quad (4)$$

where C is the number of distinct crowds/subclasses whose members behave similarly, and where C is a number significantly less than N.

If the logic required for interacting with objects was encapsulated in the objects themselves, the time is reduced to:

$$T_A = C+P+D \quad (5)$$

If one could place the crowds while simultaneously placing destinations of interest, than the amount of time required is reduced even further to be on the order of :

$$T_A = C+P+D-(2*A) \quad (6)$$

where A is the number of crowds and destinations of interest placed simultaneously.

Example: Here the costs are shown for 2 sets of input, one a population of 50 individuals, the second a population of 1000 individuals. Note how in equation five and six the cost of authoring of the large populace is within the same magnitude of the cost of the smaller.

Estimating the Costs of Authoring	Set #1	Set #2
Number Of Humans N	50	1000
Number of Interactive Props P	10	50
Number of Destinations D	5	20
Number of Crowds/Subclasses C	5	30
Number of aggregated destinations and crowds	5	20
TA from equation 1	6505	2.43E+21
TA from equation 2	625	2.43E+18
TA from equation 3	505	50020
TA from equation 4	55	1520
TA from equation 5	20	100
TA from equation 6	10	60

Figure 1 – This chart measures complexity of authoring in "authoring units (TA)". Set #1 vs. Set 2 compares authoring a population of 50 vs. a population of 1000. The chart demonstrates that without the techniques described herein, authoring does not scale well as entity count increases. The chart also shows that when using all the techniques, the magnitude of authoring cost is preserved despite a 20-fold increase in population.

We achieve the cost savings in equations 4 and 5 via our implementation of HSFM Minds, strong object-oriented design featuring polymorphism, and SmartObjects. The cost savings of equation 6 are the result of an advanced implementation of SmartObjects we call SmartBuildings.

Note that the cost savings in equation 3 are the result of automated path planning, only discussed tangentially in this paper.

SmartObjects

Our SmartObjects implementation extends the base SmartObject concept, perhaps most famously associated with the Sims© PC-based game, into an open authoring toolkit integrated with an agent-based AI framework and built on top of a complete and portable multi platform human visualization SDK.

The Sims is a groundbreaking video game that in 2002 displaced the game Myst as the top-selling PC game in history[2]. The Sims[1] lets players guide the development of a semi-autonomous simulated character (or agent) in a simulated community, developing from infancy through childhood to adulthood. Much of the gameplay involves having the simulated characters interact with other character and objects in their world. These objects have their own AI software code intended to inform the virtual characters about how their interaction with the objects will proceed. By using this paradigm, the amount of intelligence the agents needs to have resident in their "brain" at any given time is dramatically reduced. The agents now need only to carry around a limited amount of information because

much of the agents' intelligence for interacting with its world has been off-loaded into the AI algorithms that now reside within the SmartObjects.

Attempting to build agents that know about every possible object or environment they'll come in contact with rapidly creates an impractical amount of centralized code. By distributing usage logic into the objects themselves the complexity of individual agents is kept at a minimum. While the concept of inanimate objects having intelligence is at first counter-intuitive, further reflection reveals that in a very real sense this is the way living human beings operate in the world. If one considers commuting to the office in the morning for example, the roadway informs the driver of the shape of the road, where the off-ramp is, etc. When arriving at work, the layout of the building informs the commuter where the elevator is, the elevator control panel informs him/her where and which buttons to push, etc.

An excellent source of information about the SmartObjects concept as applied in the Sims is the AI Game Development website article "*Living with the Sims' AI: 21 Tricks to adopt for your game*"[3].

The logic that resides within a SmartObject is referred to as a Plan, we will see in coming sections how the plan is implemented to inform virtual character interactions.

SmartBuildings

SmartBuildings bundle path-planning regions, agents, and buildings into a single reusable package that can placed as a single aggregate into simulations during authoring or in real-time.

A typical "SmartBuilding" example in our implementation is a Farm. The pre-packaged Farm SmartBuilding contains:

- One or more farm buildings
- The designated arable fields surrounding the farm
- The members (agents) of the farming family

When the farm is placed in the world, its instantiation logic tags the farming area on the greater global navigation mesh. The members of the farming family are also created and will be found inside the farm building. The members of the family automatically have different roles and assignments, and come with a loose schedule for how they spend their time during the day.

Figure 2 – A farm SmartBuilding includes the regions marked as arable land as well as the agents that make up the farm family.

SmartBuildings are a structured set of simpler primitives, and are really just an advanced implementation of a SmartObject.

SmartObject Components

The most basic use of SmartObjects, whether in a video game or in a military training simulation, are as interactive props in the virtual world such as IEDs, swords, button panels, etc. that autonomous agents and avatars interact with. SmartObjects are composed of a common set of software elements:

- Initialization
- Usage Criteria
- Plan
- Navigation Information

SmartBuildings add additional information to the SmartObjects:

- Routing Information
- Environment tagging
- Associated agents

Typically agents decide that they want to do something, either from direct input from the user, or via a needs model, where a desire has reached a certain threshold. At that point, the agent will seek out a SmartObject, SmartBuilding or other agent that will help satisfy that need. If it is a SmartObject or SmartBuilding, the agent's AI will query the Global Terrain-Aware Repository system to locate an appropriate building.

The Usage Criteria of each object in the Repository is employed to return the best fit for the request.

For example, if the agent is hungry, the agent might query the repository for the nearest SmartObject (it could be a SmartBuilding) that's registered as a food provider. The agent is then offered the SmartObject associated state machine Plan, defining a sequence of tasks that will satisfy the need. The agent then passes its control thread to the SmartObject's plan, which takes direct control of the character. Plans typically apply the navigation information that specifies how a character should move to the object, followed by instructions to perform a single action, or a series of actions. Plans can be simple or complex; they can have sub-plans that differ according to the characteristics of the agent using the plan. For example, our implementation of a Mosque SmartBuilding had a different sequence for men vs. women, as the behavior of the sexes at the mosque differs.

For SmartBuildings, routing is an important component. The mosque had different regions of the mosque marked and the Plan would send the women to one section of the Mosque while sending men to another. By providing routing information, the SmartObject greatly enhances the behavior of agents inside or maneuvering around the outside of a SmartBuilding, supplementing the cruder methods agents use when navigating the general terrain. But even some of the simplest SmartObjects have important routing information, for example, a door knob will position the character properly to open the door.

SmartObjects often have initialization code that is run when the object is placed in the world. Authoring productivity is directly enhanced by this step. When the farm is placed in the world, the farm not only autonomously creates and places other SmartObjects into the world simultaneously like the surrounding fields, but also generates the agents that occupy the house, a default schedule for the occupants, and specialized user interface to assist the author with customizing the farm and its occupants' behavior.

A Global Terrain-Aware Repository

The GTA Repository is an important concept for proper use of SmartObjects. By use of the Repository, agents are encouraged to request things from their world in a general manner vs. directly accessing a SmartObject. Agents can express general requests ("I am hungry", "I want to socialize", "I want to plant an IED") and the repository will respond with the nearest eating establishment, nearest park or tavern, or suitable road locations frequently trafficked by friendly forces.

Figure 3 – Agents hunger is fulfilled by the fruit stand, found in the Repository. The agent's microthread is passed to the fruit stand to manage the interaction. Note how thought and speech balloons can help understand character motivation during debugging.

HSFM Minds, high level scripting and coroutines

The majority of the virtual humans in the simulation must act as autonomous agents in high human entity count simulations. Because the goal is to decrease authoring time and be reactive to temporally unpredictable simulations events, our implementation uses HSFM Minds to model the artificial intelligence of an agent. These Minds are implemented using the Lua scripting language.

AI developers for games such as the popular World of Warcraft choose Lua[7] for a number of reasons in common with ours. By using Lua, authors are able to construct and extend AI Minds without the time-consuming recompile and relink required by C++ or other high performance low level coding implementations used in the majority of the simulation code, increasing author productivity. But perhaps the most important feature in Lua that attracts AI developers is its native support of coroutines[8].

Coroutines are a method of creating lightweight threads (or microthreads, as they are sometimes called) which allow for cooperative multi-tasking, instead of the more typical pre-emptive multi-tasking prevalent in most languages. While C++ functions have a single entry point and need to complete the function in order to not lock up the application, coroutines allow pausing and resuming inside of a single function via their *yield* and *resume* functionality. Coroutines therefore enable a series of time-consecutive actions to be grouped together in a single function, enabling programmers to write code that maps well to a behavioral procedure that can be described in plain English (e.g. "Travel to

location and talk to shopkeeper") A function can be as simple as:

```
Play_animation()
Sleep_until_animation_done()
Location = where_is_a("shopkeeper")
Move_to_location(location)
Sleep_until_arrived()
Talk_to_shopkeeper()
```

Such a function can play out over any length of time, without the CPU having to spend time in the high level language (Lua in our case). This naturally allows for hundreds or thousands of agents to yield to each other at runtime, resume their calculations in the next simulation timestep, and sleep until a new decision point is reached, all in a simple and straightforward manner.

Coroutines also make AI user-scalable with respect to time. Agents that play an important role in a particular simulation scenario can be set to access decision-making code more frequently. Agents that are meant to be mindless "ambient" characters wandering around in the background can be set to call computationally expensive code less often.

Figure 4 – Using coroutines, high entity count in this train station simulation becomes manageable.

Lua's support for coroutines as first class primitives means that a plan can be a simple function that the agent invokes. Other scripting languages popular due to their first class coroutine support include Python[9], a Python-variant called BOO[10], Unity 3D scripting[11], and UnrealScript[12].

While other languages can support cooperative multi-tasking, because Lua supports coroutines as one of its basic primitives, it makes it exceedingly easy to encapsulate the state of the character and pass coroutine control outside of the character – we will see that this is a critical component in a successful implementation of SmartObjects.

Why non-linear? Why hierarchical?

In general, our implementations have favored a fairly flat (or non-linear) state machine conceptualization, where most states can be directly transitioned to from the other states, these transitions occurring based on incoming messages, scenario-level events, and changes in agent internal state. We believe that this non-linear state machine approximation of human behavior is reflective of the way human beings actually behave: we're highly interruptible and spontaneous.

The flat or non-linear approach is also good in that it forces the majority of state transitioning into message handling routines, where the commonality of how and when state changes should occur can be shared amongst multiple states.

By their nature, non-linear state machines are difficult to visualize effectively. Thus visualization is not as helpful for understanding or extending the state machine as it is for its linear counterpart, where a state machine diagram helps understand how a soda machine operates. Nonetheless, we feel that linear approaches shortchange the realism of how people and human networks truly operate. Linear approaches often reveal their flaws to trainees who understand quickly how to get around the too-rigid implementations common in visually-generated state machines, while implementers can be frustrated when trying to address those shortcomings as they short circuit the linear approach they started with, resulting in the worst of both worlds.

We are exploring differences in how implementers write and design their HSFM state architecture, particularly the use of hierarchy, and are looking for ways to standardize. The use of hierarchy within the non-linear state machine allows some of the "linear" thinking of authors to not get lost in implementation, and increases the ease of revision. By better understanding how authors think about and wish to implement non-linear state machines, we also hope to develop better user interface tools to assist with their creation.

Modeling "Human Networks"

A powerful result of the combination of assets described in this paper has been the ability to model complex "Human Networks". We define Human Networks as an organized activity requiring multiple agents performing different roles over a length of time that result in measurable output. For IED defeat training, we have modeled Human Networks for the financing, creation, and deployment of IEDs. We have also used Human Networks to model insurgent recruitment.

As discussed in the Global Repository section, various SmartBuildings are registered. The agents that compose the human network tend to follow the pattern of:

- Ask repository for the appropriate building to satisfy his current "need" as part of the Human Network. Navigate to the building.
- Perform work at the building. In simple cases, the character just sleeps. In more complex cases, the character may request a transfer of currency or goods to/from the building. The building may even manufacture an object such as an IED.
- Loiter at the building using sleep(). Some agents must wait for a certain condition in the building to be fulfilled before they can satisfy the next step of their need, for example, a cash smuggler will wait until the building receives cash. Alternatively, a bomb smuggler may not exit his building until notified by other agents that certain parts are available for pickup.

Our implementation of the insurgent networks gives them an implicit "economy". A farmer grows poppies. When ready, he delivers poppies to a drug lab. The drug lab gives the farmer cash, which he may bring to the financier building where a cash smuggler may get them to a bomb parts supplier. The intent is not only to model a complex network, but also to understand how counter measures might slow down or even stop a network.

Figure 5 – Virtual soldiers deployed to a crowded urban environment. Soldiers today are increasingly being trained on "soft skills" such as negotiation and hostile-intent detection, requiring high entity count scenarios.

Notes on Technology Prerequisites

Effective high entity count simulation using the HSFM Mind and SmartObject solutions described herein requires a number of content and software module components be available to either directly support the implementation or indirectly to support the intelligent navigation and behavior of the Agents using it:

- realistic 3D buildings
- realistic 3D human models
- realistic low level human animation
- graduated methods for authoring human behavior (refer to our document "Defeating the Authoring Bottleneck…"[4])
- realistic terrain that is well-formed to satisfy automated path-planning requirements and environmental tagging.

In addition, we highlight in detail below some other key technology requirements.

Environmental Tagging

Our solution supports direct graphical mark-up of the terrain. Areas of preferred navigation such as crosswalks and sidewalks need to be identified for agents to use and as supplemental information for advanced navigation planning. The more the terrain is "pre-marked up", the better, but in general we believe that manual visual and interactive methods for terrain markup will be required as AI authors match how the terrain needs to inform the agents to the specific requirements of the end training. However, in some cases, pre-marked up terrain along with purely analytic methods will be sufficient.

Path Planning

Considerable effort was spent during our AI and SmartBuilding implementations to support high performance automated navigation path planning. Along with intersection testing, navigation planning is a critical performance roadblock for real-time simulation of hundreds or thousands of lifeforms. Our implementation uses the popular A* search algorithms[6] for calculating path plans. Note that path plans algorithms will return the path with the lowest cost, which is not necessarily the shortest path. In addition to the terrain markup concepts discussed earlier for reducing the cost of using sidewalks and crosswalks for pedestrians, we have also implemented mission-specific costing algorithms. For example, path planning for friendly forces performing tactical movements near the enemy will decrease the cost of moving along the "edges" of the terrain to simulate soldiers taking routes that provide cover by maneuvering close to buildings and walls.

Navigation Mesh Generation

Navigation planning is directly supported by creating a high performance "navigation mesh" of the terrain being simulated. This step is typically automated and performed before simulation runtime – although it is not uncommon to have some amount of human refinement when finalizing the navigation mesh results (Note that in these cases having robust visualization and editing tools increases the efficacy of such editing). Simulations that support dynamic terrain deformation (e.g. – a wall is destroyed at some point during execution of the scenario) will want to have fast algorithms for incrementally modifying pre-generated navigation meshes at runtime.

Intersection Testing

Fast algorithms for looking up terrain polygons for static object collision avoidance are required. Typical solutions involve dividing the polygonal space into quad or octtrees[5]. Absence of such algorithms will cause excessive simulation update times as agents request line-of-sight tests, ballistic trajectory calculations through the environment, and collision information.

Communication

A common communication protocol shared by the agents is required for successfully implementing SmartObjects. Mechanisms must exist to broadcast messages to agents and SmartObjects, broadcast a message to other agents and SmartObjects within a certain range, as well as support point-to-point communication between agents and/or point-to-multiple-entity communication.

Scheduling

In direct support of the "pattern of life" goals of the AI implementation, scheduling techniques for loosely specifying behavior is required for modeling some of the most basic pattern of life concepts (everyone on the farm wakes up at dawn, but not at exactly the same time. A subset of people on the farm, the children, go to school at 7 AM, but not all exactly at the same time, etc). Graphical User Interface (GUI) support for scheduling should be provided to the author for easy creation, review, and editing of the schedules of the agents. The method typically designates a crowd or a subset of a crowd, the time for the goal / state change, and a +/- time duration during which the system will command the transition.

Perspectives

Performance Considerations

While one might conclude that the particular Mind or SmartObject Plan implementations of when agents should sleep, wake up, etc, might be the most critical factors in achieving high performance, our results suggest that the critical factor to monitor is the use of high cost C++ functions for performing calculation-intensive activities like path planning, analysis, line of sight calculations, etc.

For example, if a bomb goes off, the general reaction is for civilians to flee the area. If there are 100 civilians in the area and they use path planning as a means of calculating a route to a safer location, the simultaneous request of 100 path plans can cause an undesirable pause effect if not limited properly. Note that while C++ functions are a higher performance coding solution than Lua scripts, making them essential for computationally intensive tasks, due to their non-yielding functional nature they will not release the CPU until their calculations are complete. Techniques for mitigating this problem in the flee case include staggering the path planning requests, or designating some members of crowds as leaders who request a path plan that the other agents simply follow.

When agents need to seek cover, finding ways to pre-identify cover by marking up the terrain versus analytically determining cover via the execution of a large number of ray traces which determine line-of-

sight and collision information is another work around typical to video games and simulated battlefield training. It is not difficult to imagine 50 or more ray traces required to find cover for a single character seeking cover from a single source of fire. Note however that the pre-determined cover solution is not without its own problems. Sometimes cover is directional, which is harder for terrain markup to indicate, and the technique by its nature excludes the use of dynamic entities such as tanks or other vehicles to be used as cover. Nonetheless, the use of pre-determined cover regions in video games is highly prevalent, particularly when agents are in the proximity of buildings. Supplying SmartBuildings with pre-defined cover regions in their navigation information is an effective means of providing performance-friendly cover information in an efficient authoring manner: since the building has already had its cover points pre-processed when it was upgraded to a SmartBuilding, the work is essentially already completed when the scenario author simply places the SmartBuilding and agents into the virtual world.

Goal Evaluation

We have also observed a particular spectrum for the HSFM minds worth discussing here. At one end are entirely autonomous characters. These agents are placed in the world and for the duration of the scenario will not have any direct human operator intervention. At the other end are "directive characters" like 1st person player avatars and 3rd person avatars controlled in a god-like manner by the simulation operator (who in our system might right-click on characters to reveal a pull-down menu with commands for formations to form, goals to achieve, interactive dialogs for allowing the operator to designate a location on the terrain for the agent to navigate to, etc). Depending on where an agent is located on the spectrum, how often and when the agent needs to re-evaluate his goals based on changes in the environment varies greatly, causing differences in how the states and message processing are implemented. Agents in the middle of the spectrum can be the most difficult to author (see the next section). If we examine our flee situation earlier from when a bomb exploded, one can imagine how different the state machine implementation would be between the two ends of the spectrum. The autonomous character in particular is challenged to determine what he should do after he has successfully fled. Does he resume the activity he was doing before the bomb explosion occurred? It might not make sense to continue shopping if the market has just been blown up! This situation is much less acute for the directive characters, where the operator will most likely decide the next step to take.

Conclusion and Future Work

Overall, we believe we've had tremendous success with the AI Minds, SmartObject, and SmartBuilding framework described herein. Future work involves continuing to try to make the authoring more accessible – much of what humans do is complicated so providing authoring techniques that reduce that complexity is an ongoing challenge. Identifying standards for AI Mind and SmartObject implementation authoring is also a goal. We are planning to make the user interface portable into 3rdparty applications to stimulate more use of the authoring tools.

The applications for pattern-of-life and human-oriented simulation are as wide the activities of the human race itself. While application of these techniques have been focused in gaming and military simulation, fields such as emergency response, law enforcement, warehouse management training, interrogation, and interpersonal skills training are all application areas which might benefit from this approach.

ACKNOWLEDGEMENTS

The authors want to thank Marc Raibert, co-inventor of DI-Guy, as well as Ryan McMahon, Marco da Silva, Gregory Kaufman, Robert Playter, Bob Bryan, Dave Wooden, Patrick Mayeux, Marc Schlackman and the other members of the DI-Guy team for their support of this paper and providing the content and code foundation upon which the SmartObject implementations were built. The authors also wish to thank Lt. Col. Timothy E. Barrick, Lt. Col Stephen Van Riper, Modeling and Simulation Officer Austin "Pops" Bryant, Gunnery Sergeant (retired) Jeremy M. Sisco, Federation Simulation Analyst Kyle Teasdale and the US Marine Corps' MCTOG organization for their vision and insights that have helped spur the development of DI-Guy AI.

REFERENCES

[1] The Sims is a copyright of Electronic Arts Inc. Latest news about the Sims can be found at *http://www.thesims3.com*

[2] Crecente, Brian D. Sims best-selling game of all time! http://www.geek.com/articles/games/sims-best-selling-game-of-all-time-20020325/

[3] Champandard, Alex J. Living with the Sims' AI: 21 Tricks to Adopt for your game http://aigamedev.com/open/highlights/the-sims-ai/

[4] Blank, Bill. Defeating the Authoring Bottleneck: Techniques for Quickly and Efficiently Populating Simulated Environments. 2009 IMAGE Conference Proceedings.

[5] Glassner, A.S. Space subdivision for fast ray tracing. IEEE Computer Graphics and Applications vol. 4, no. 10, pp. 15-22, Oct. 1984

[6] Hart, P. E.; Nilsson, N. J.; Raphael, B. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *SIGART Newsletter* 37: 28–29.

[7] Lua WoWWiki Your guide to the World of Warcraft, http://www.wowwiki.com/Lua

[8] Conway, Melvin E. Design of a Separable Transition-Diagram Compiler". *Communications of the ACM* (New York, NY, USA: Association for Computing Machinery) 6 (7): 396–408.

[9] Python Programming Language Official Website, http://www.python.org

[10] BOO – Home, http://boo.codehaus.org

[11] Unity Script Reference – Scripting Overview, http://unity3d.com/support/documentation/ScriptReference/index.html

[12] UnrealScript Language Reference – Latent Functions, http://udn.epicgames.com/Three/UnrealScriptReference.html