

Rapid Development of Intelligent Agents in First/third-person Training Simulations via Behavior-based Control

George Alexander
Frederick W. P. Heckel
G. Michael Youngblood
D. Hunter Hale
Nikhil S. Ketkar

University of North Carolina at Charlotte
9201 University City Blvd.
Charlotte, NC 28223

gralexan@uncc.edu, fheckel@uncc.edu, youngbld@uncc.edu, dhale@uncc.edu, nketkar@uncc.edu

Keywords:

behavior-based control, intelligent agents, authoring tools

ABSTRACT: *First/third-person training simulations in virtual environments have become increasingly used; however, authoring intelligent virtual agents to populate these environments presents a large authorial burden. Our work focuses on building tools to enable rapid creation of intelligent agents for first/third-person game-like environments that enable users with no programming knowledge to develop interactive agents. This is made possible using an intuitive agent architecture known as behavior-based control combined with a user interface employing natural language-like agent specification and an interactive testing during agent development. We present the results of a study indicating that users with no programming experience can successfully design agents using our tool — defined as creating an agent that would carry out at least 80% of role-specific baseline behaviors — after only minimal training in the interface.*

1. Introduction

Lately, first/third-person training simulations have played an increasingly important role in facilitating mission rehearsal, environment familiarization, and cultural awareness (e.g., Hill, 2006). The two most important parts of any first/third-person training simulation are the environment (which consists of the terrain, static objects such as buildings, and interactive objects such as vehicles) and the intelligent agents in the environment (for example, bystanders at a car crash or victims in a building on fire). Typically, developing any new scenario requires building a new environment and new intelligent agents. This process is almost always time critical, as the earlier the scenario is developed, the more it can contribute to training. Hence, it is important to develop intuitive tools which facilitate the rapid development of such scenarios. While there has been a lot of progress in developing tools to model environments, the tools to model intelligent agents lag far behind. Often the design of intelligent agents requires programming, which considerably slows down the process of building scenarios. This paper demonstrates how behavior-based control can facilitate the rapid development of intelligent agents without traditional programming.

Behavior-based control has its roots in robotics and is an extension to the subsumption architecture (Brooks, 1986).

At an abstract level, agents created with behavior-based control consist of one or more prioritized layers of behavior, where each layer maps a combination of percepts to a combination of actions. At any instant in the simulation, the agent receives one or more percepts, activating one or more behavior layers, causing the action(s) associated with those layers to be carried out by the agent. We explain this paradigm more concretely with an illustrative example in Section 3, below.

Our recent work has focused on the use of behavior-based control in first/third-person training simulations, initially reported in (Heckel 2009). Our research framework DASSIES (Dynamic Adaptive Super-Scalable Intelligent Entities) incorporates tools to design agents via behavior-based control (BehaviorShop) and a behavior-based simulation engine (BEHAVEngine) which operates on the agent definitions (produced using BehaviorShop) and implements their behavior in any standard first/third-person game engine. In our own work we use the F13RST (First and 3rd Person Realtime Simulation Testbed) for experimentation, but are working with VBS2 and RealWorld as well.

We have evaluated the effectiveness of the behavior-based control paradigm in extensive human trials. Participants of the trial (mostly people not from a computer science background) were provided a text

description of an agent and were asked to design an agent using BehaviorShop. The study involved a total of 13 different baseline agent descriptions, divided into five scenarios, and over a hundred participants. Results indicate that at least 80% of the participants were able to build agents with at least 80% behavioral accuracy (based on compatibility with text descriptions of each scenario). This level of performance met our target benchmark, which was set based on initial promising results in a pilot study conducted with a simpler prototype of BehaviorShop.

These results strongly assert the potential of behavior-based control in designing agents for first/third-person training simulations. Behavior-based control and its implementation in BehaviorShop and the BEHAVEngine represent the state of the art in building agents for first/third-person training simulations, and their adoption will greatly enhance all first/third-person training simulations.

1.1 Related work

An AI building tool should take into account three major factors: the manipulation of simple atomic decision units into larger wholes (pixels in images, primitive shapes in 3D modeling), immediate feedback as the character is modified, and abstraction and reuse of existing character models.

The existing work in AI builders address, at most, the first of these factors. Tools have been developed for robotics, including the RobotFlow builder from the University of Sherbrooke (Cote et al. 2004). The base-level units of RobotFlow are low-level (higher-level behaviors are built from networks of *nodes*, input/output units roughly equivalent to programming language functions), allowing a great deal of flexibility when creating new systems. Unfortunately, this level of complexity is daunting for non-expert users.

Sony uses Brian Schwab's Situation editor for building characters in sports games (Schwab 2008). This editor has similar goals to our own, but even the author admits that it is difficult to learn, noting that experienced programmers require at least a week of training. The Eki One Configurator from Artificial Technology is a commercial product aimed at games (Artificial Technology 2009). It provides a more polished FSM editor, but does not solve the problem of transition complexity. Xaitment also produces a set of commercial packages for editing FSMs and knowledge bases, but these tools are not appropriate for AI novices (Xaitment 2009).

FSMs (Finite State Machines) are a common choice for the architectures underlying agent builders. The

commercial package SimBionic is designed for building game AI, and provides a HFSM (Hierarchical Finite State Machine) modeling interface, a debugger, and engine (<http://www.simbionic.com/>). SimBionic is an extension of Fu's BrainFrame software (Fu and Houlette 2002). AI.implant is another commercial package for building simulation AI, and is developed by Presagis (Presagis 2001). The AI.implant tool allows the user to model game agents using a variety of methods, most notably FSMs and HFSMs.

Agent Wizard is a specialized interface for building software agents (Tuchinda and Knoblock 2004). It uses a question-based system, which queries the user to specify various facets of the desired agent. This approach is accessible, but this tool is domain-specific for web software agents rather than game/simulation agents.

Each of these builders uses an artificial agent architecture to instantiate the created agents. Many possible architectures exist, but in game AI, FSMs are very commonly used to drive character AI. While they can be used to quickly build AI, and the basic idea is intuitive, the number of transitions between states can grow to an unmanageable level for complex agents. This can be partially overcome through the use of HFSMs or Behavior Trees, which are also commonly used in games (Fu and Houlette 2004). The hierarchical approach can reduce the complexity of the top level FSM, but are still time-consuming to build.

2. Designing a Scenario in a First/third-person Training Simulation

The key goal of designing a scenario in a first/third-person training simulation is to facilitate the training of operatives for a particular situation in a test bed that closely resembles the real world. This allows them to develop expertise with respect to the particular situation (for example, training rescue workers to systematically search a building for victims of a fire). Training scenarios can be extremely diverse in nature, but at an abstract level consist of three key elements, namely, the environment, the intelligent agents, and the human agents. A training simulation in a sense emerges from the interaction of these three elements, and designing a particular scenario involves modeling the environment and building the intelligent agents to mimic a real world situation. This process is best described by considering a specific scenario. Consider, for example, a scenario which focuses on training rescue workers to systematically search a building for victims of a fire. In this case, the terrain and the particular building (with static elements such as walls and interactive elements such as doors and elevators) form the environment. The victims of the fire in different

parts of the building are the intelligent agents. Designing this scenario would thus involve modeling the environment and building the agents, after which this scenario would be ready to be used for training rescue workers.

The current state of the art in building such scenarios includes a diverse array of tools for environment design. Intuitive 3D modeling tools, as well as existing libraries of static and interactive objects, can be leveraged to construct a realistic environment. An important fact to note about building environments is that this process is fundamentally a 3D modeling exercise requiring no programming expertise and, given intuitive tools, can be completed relatively easily although, some artistic talent/training is often required. Furthermore, existing libraries of environments and objects can be easily leveraged. For example, objects such as vehicles need to be designed only once and can be reused for multiple scenarios.

When it comes to building intelligent agents, the situation is far more complicated. There are hardly any tools that match the intuitiveness or maturity of 3D modeling tools, and often, agents need to be built by programming or scripting. Achieving the most trivial behaviors takes a significant amount of time, and agent building remains the most time consuming step of scenario design. Furthermore, it is relatively hard to reuse intelligent agents. For example, in the rescue worker scenario described earlier, we cannot use a single agent to model all victims in the building. Typically, we would want a range of behaviors randomly assigned to different agents. The important fact to note about intelligent agents in training simulations is that, in contrast to building the environment, this has fundamentally been a programming exercise requiring a certain amount of expertise in logic/algorithm construction.

Another important point to note is that, unlike environments that can be designed by 3D modelers based on descriptions, building intelligent agents requires a subtle understanding of the scenario and needs to involve domain experts who often lack the programming skills to achieve this task. For instance, building intelligent agents mimicking bystanders in a foreign country would require a nuanced understanding of the culture, which is hard to describe, and should be built by a domain expert, while the buildings and vehicles can be easily described via standard technical specifications. The lack of intuitive tools for building intelligent agents often requires the domain expert to collaborate with a software developer, complicating and delaying the process of scenario development.

It is thus critical to develop a theoretically sound framework for building intelligent agents to serve as a foundation for designing intuitive tools to address the problem of creating interesting agents in first/third-person training simulations. While this overarching objective is clear, achieving it requires incorporating ideas from two seemingly diverse fields, artificial intelligence (AI) and human-computer interfaces (HCI). The field of AI, to a large extent, has focused on building intelligent agents achieving concrete goals in an optimal manner with little regard to the complexity of defining such agents. HCI, on the other hand, studies the design and implementation of intuitive interfaces that allow the human user to achieve the task at hand with relative ease. The problem at hand requires formulating a framework that balances what the agent can achieve with how complex the agent specification is. Behavior-based control achieves this balance, as discussed in the next section.

3. Behavior-based Control

Behavior-based control is an extension to the subsumption architecture (Brooks, 1986) and has its roots in robotics. Agents created with behavior-based control consist of one or more prioritized layers of behavior, with each layer mapping a combination of percepts to a combination of actions. At any instant in the simulation, the agent receives one or more percepts, activating one or more layers and causing the action(s) associated with those layers to be carried out by the agent. Behavior-based control is inherently parallel in the sense that multiple percepts can be received at a single instant of time, which can lead to multiple actions also being performed at a single instant of time. There are two key aspects to behavior-based control, the first being the mapping of percepts to actions (or combinations of percepts to combinations of actions) represented using one or more behavior layers, and the second being the prioritization of the layers, which specifies which layers override other layers in the case where multiple percepts are received. We illustrate these key ideas in behavior-based control using a toy example.

Consider an intelligent agent which mimics a simple organism in an environment with the following two predefined percepts: a) perceive food and b) perceive predator. Furthermore, the agent has the following three predefined actions: a) explore new regions, b) consume food, and c) flee from predator. Given these basic percepts, an agent design using behavior-based control is illustrated in Figure 1. The key points to note are as follows. Firstly, note that each of the layers maps percepts to actions. For example, layer L2 maps the percept *food* onto the action *eat*. Furthermore, note that the layers are prioritized. Layer L1 is overridden by layer L2 which in turn is overridden by layer L3. Also note that layer L1

does not have a percept and corresponds to a default action which is performed when no percepts are received by the agent. This simple agent designed using behavior-based control has the following overall behavior. When there are no percepts available, the L1 layer is triggered, and the agent explores new regions. In the case where the agent perceives food, the L1 layer is overridden by the L2 layer, and the agent consumes the food. In the case where the agent perceives a predator, the L3 layer is triggered, all the layers below it (L1 and L2) are overridden, and the agent flees from the predator. Note that the prioritization of the layers is the most important part of the agent definition.

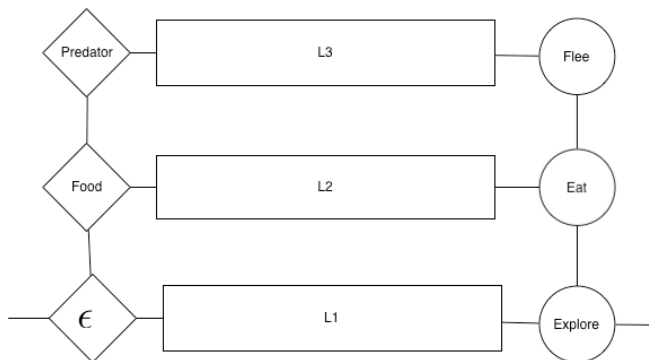


Figure 1. Behavior-based control for a toy agent. The agent has three behavior layers, with associated trigger conditions on the left and resulting actions on the right.

While the toy example discussed here is only for the purposes of illustration, it does demonstrate the key aspects of behavior based control, namely, the mapping of percepts to actions via layers and the prioritization of layers. Given any large set of percepts and actions, any reactive agent of arbitrary complexity can be constructed using behavior-based control. It is now essential to demonstrate how behavior-based control fits in with first/third-person training simulations from a systems perspective, which we discuss in the next section.

Behavior-based control has a number of advantages over the use of other architectures for game agents, such as finite state machines (FSMs), hierarchical finite state machines (HFSMs), and behavior trees. Behavior-based control is inherently parallel, as multiple active behaviors can be run at once by varying the override policy of each layer. The representational complexity of a behavior-based control agent, which is an important consideration for the agent authoring process, is far lower. In the example from Figure 1, the corresponding finite state machine requires more transitions (see Figure 2). If multiple behaviors are allowed to be active at once, the finite state machine becomes increasingly more complex, as each allowable combination of behaviors requires an additional state. HFSMs and behavior trees reduce this

complexity over simple FSMs, but still require more complex models to represent the same agent. The reduced complexity of behavior-based control makes it simpler and faster to create intelligent agents that can embody more complex and expressive intelligence.

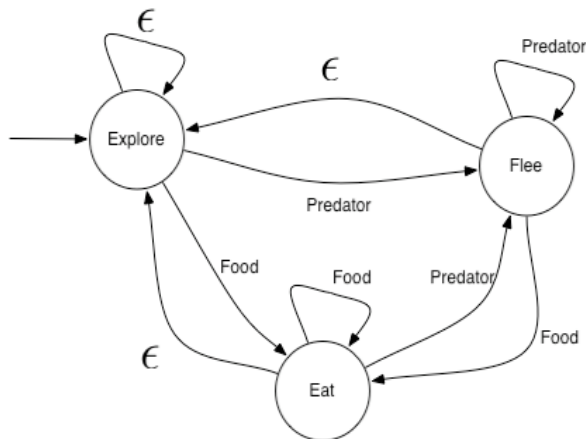


Figure 2. The finite state machine equivalent of our example agent requires only three states but nine transitions, one for each (trigger condition, state) pair.

4. Implementation of Behavior-based Control Using BehaviorShop and BEHAVEngine

DASSIES (Dynamic Adaptive Super-Scalable Intelligent Entities) is our primary research framework, and it includes an industry strength implementation of behavior-based control targeting first/third-person training simulations. The architecture of DASSIES is illustrated in Figure 3.

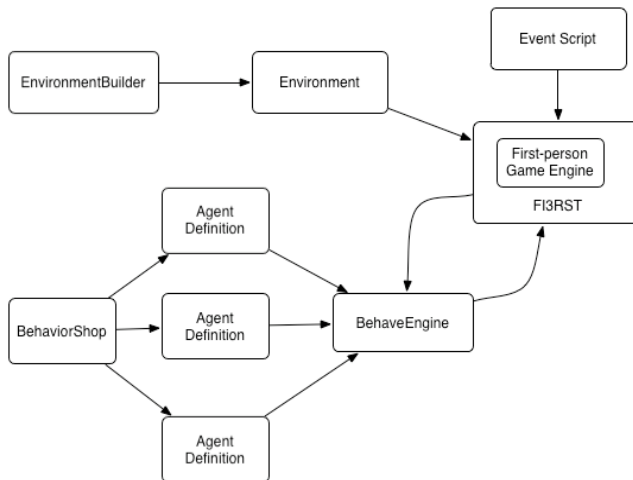


Figure 3. The DASSIES system is composed of four principal components: the agent builder interface (BehaviorShop), the agent engine (BEHAVEngine), a first/third-person simulation environment, and the interface between our agents and the environment (FI3RST).

The key components implementing behavior-based control are BehaviorShop, which is a tool to design intelligent agents, BEHAVEngine, which operates on one or more agent specifications, and any standard game engine to produce a specific simulation. An additional support component is FI3RST (First- and Third-person Realtime Simulation Testbed), which is a wrapper around game engines to provide a standard interface for BEHAVEngine (Currently FI3RST supports the Panda3d (www.panda3d.org) and Irrlicht (irrlicht.sourceforge.net) game engines, but could be extended to support any standard game engine).

4.1 BehaviorShop

BehaviorShop, the component which allows users to build intelligent agents using behavior-based control, has an intuitive user interface based around using sentence-like constructions to define agents. Screenshots of the startup screen, the layers window (where the user defines the layers), and the trigger-action editor are presented in Figures 4, 7, and 8, respectively.

The layers window, illustrated in Figure 7, can be used to add, delete, and move layers. The layer editing window depicted in Figure 8 can be used to select triggers and behaviors for the layers and define levels of priority. Often, actions performed by the agents require positional parameters. These can be specified by selecting locations on a preloaded map through the map window, illustrated in Figure 5. At any point while designing the agent, the user can test and debug the agent by watching the simulation in the output window, depicted in Figure 6.

Each behavior layer in BehaviorShop is defined by selecting choices to fill out an if-then sentence, possibly with multiple triggers and/or actions. For this reason, the vocabulary presented to the user is very important. The language in our early prototype was based on developer opinion, a practice commonly referred to as armchair design. Of course, the HCI community has long been aware of the difficulties with this approach (Furnas et al. 1987). To bring our interface vocabulary more into line with the vernacular, we conducted a study in which participants were asked to read a brief scenario description and provide free-form text instructions for a selected actor and to watch a short video clip and describe the actions of one of the actors in the scene. From this vocabulary study, we were able to present a more natural syntax in our interface as well as to ensure we included the most commonly used words for describing a scenario.

4.2 BEHAVEngine

Agents defined by BehaviorShop are executed by BEHAVEngine in conjunction with FI3RST and the game

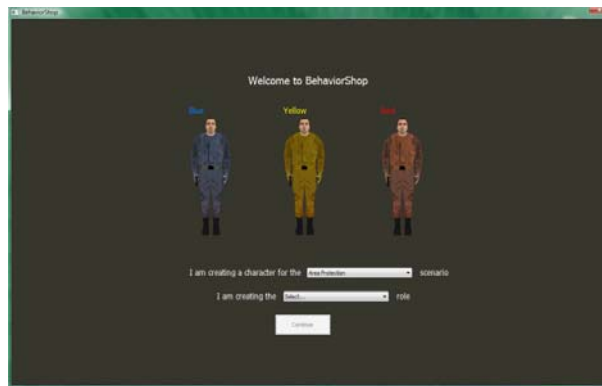


Figure 4. Upon starting BehaviorShop, users are asked to choose a scenario and role to create an agent for.



Figure 5. In order to define triggers and actions that involve location(s), such as guarding or patrolling, users are presented with a 2D top-down view of the simulated world.



Figure 6. Users can interactively test agents in the simulated environment during the agent building process.

engine. BEHAVEngine constantly receives percepts for the agents in the simulation, interprets the agent design, computes the appropriate actions based on the agent design, and passes the action messages on to FI3RST.

These action messages are interpreted by FI3RST and appropriate action animations (for example walking, jumping, and shooting a target) are chosen from a library of basic actions and played in the game engine.

BEHAVEngine is a multi-threaded behavior-based control engine for game agents. In addition to the core intelligence architecture, it integrates navigation using

navigation meshes (McAnils 2008) and a modular perception and action system. Navigation meshes are decompositions of navigable space in the world into convex regions. These enable efficient path planning and information compartmentalization. The perception and action systems make it simple to adapt the engine to different simulation environments.

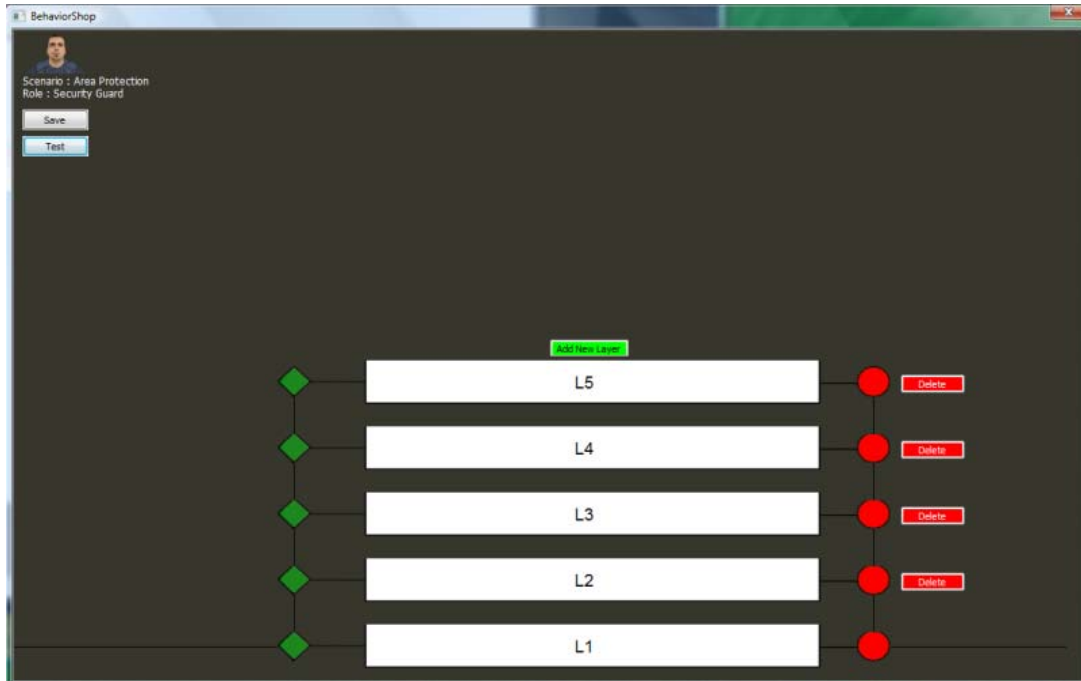


Figure 7. Behavior layers are constructed individually with the lowest priority layers on the bottom. Layers can be reordered by dragging them to a new location with respect to the other layers.

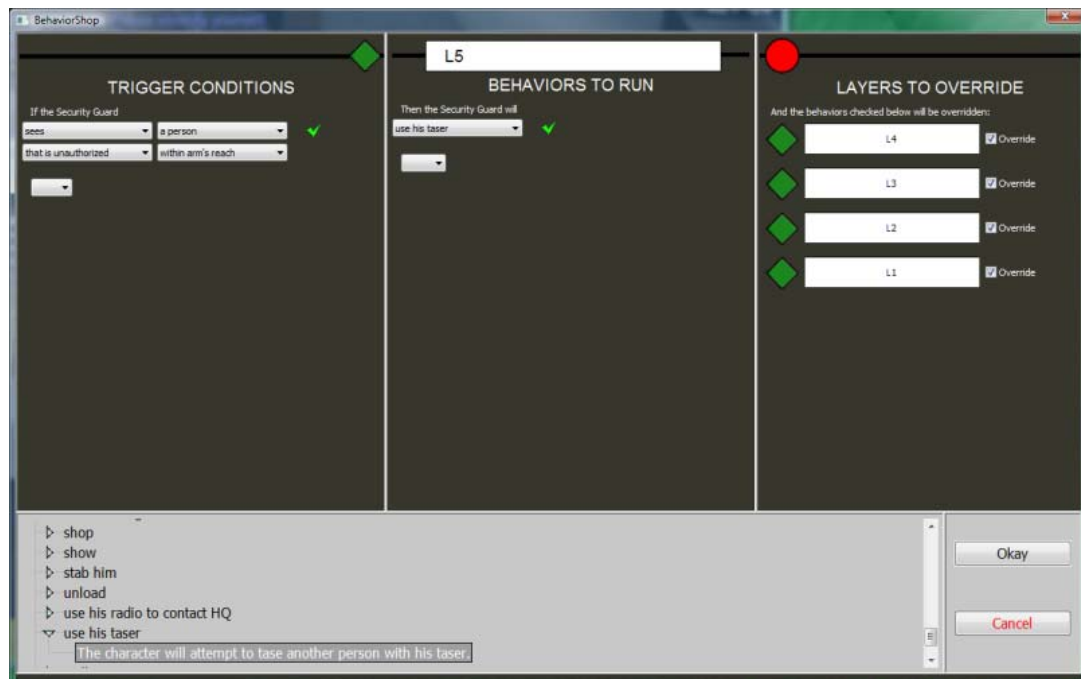


Figure 8. Each behavior layer is defined by selecting options to fill out an if-then sentence structure, possibly containing multiple trigger conditions (disjunctions or conjunctions) and multiple resulting actions (to occur sequentially or in parallel). By default, each layer overrides every layer below it, but these overrides can be disabled. 192

5. Human Trials with Behavior-based Control

The effectiveness of behavior-based control and its implementation in BehaviorShop and BEHAVEngine have been extensively validated using a large-scale human trial. Participants were asked to construct an agent for one of thirteen possible characters based on a written description. Characters ranged in complexity from a simple shopper in a market to a bomb squad technician requiring several complex behaviors. After watching a short instructional video 10 minutes in length, participants built an agent using BehaviorShop. A total of 102 participants submitted an agent they had constructed to be evaluated. These participants were drawn from a random sampling of people, the vast majority of whom had no previous experience creating agents or programming. Agents were evaluated on a ten-point scale by a panel of experts based on how closely they adhered to baseline agents developed for each character. Any borderline agents were loaded into the simulation environment, and their performance was evaluated in the simulation. A score of eight or higher indicated that the agent could successfully perform the assigned task. Based on this scoring metric, 82 successful agents were created (scoring 8 or higher) for a success rate of 80.39% of all agents created. Among the successful agents, the average score was 9.4 out of 10, which indicates a high degree of convergence with one of the baseline agents for a given task.

Feedback from the participants about their experiences with BehaviorShop was recorded using five-point Likert scales, where a rating of one indicates the user strongly disagreed with the statement and a rating of five indicates they strongly agreed. Participants were asked to rate the statement "Creating simulation characters is easy with the DASSIEs Creation Tool"; overall, the users averaged a 3.8, indicating they agreed with the statement and found agents easy to create. Additionally, participants were asked to rate the statement "I understood how to use the tools"; this statement averaged a 3.9 on our Likert scale, which indicates that most of the users did in fact understand BehaviorShop.

6. Conclusions

First/third-person simulations are an important part of modern training regimens for complex situations, facilitating mission rehearsal, environment familiarization, and cultural awareness. However, until now, creating complex intelligent agents for these simulations has required similarly complex authoring tools or computer programming knowledge. Behavior-

based control is a new paradigm for modeling these agents in an intuitive manner, without sacrificing the expressive power of more cumbersome formalisms such as finite state machines. Employing BehaviorShop and BEHAVEngine to leverage the power of behavior-based control, users can easily create interesting intelligent agents with complex behaviors, overcoming a major hurdle in the development of first/third-person training simulations.

Our future work includes extending BehaviorShop to incorporate teams of agents to discover whether non-expert users can successfully create teams of cooperative agents in more advanced variations of the scenarios employed in the study discussed here.

7. Acknowledgements

This material is based on research sponsored by the US Defense Advanced Research Projects Agency (DARPA). The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the US Government.

8. References

- Artificial Technology (2009). Eki One. <http://www.eikone.com>.
- Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*..2(1), 14-23.
- Cote, C., Letourneau, D., Michaud, F., Valin, J. M., Brosseau, Y., Raievsky, C., Lemay, M., and Tran, V. (2004). Code reusability tools for programming mobile robots. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, vol. 2, 1820-1825.
- Fu, D., and Houlette, R. (2004). The ultimate guide to FSMs in games. In (Ed.), *AI Game Programming Wisdom 2*.pp.283-302. Boston: Charles River Media.
- Fu, D., and Houlette, R. (2002). Putting AI in entertainment: an AI authoring tool for simulation and games. *IEEE Intelligent Systems*, 17(4), 81-84.
- Furnas, G. W., Landauer, T. K., Gomez, L.M., and Dumais, S.T. (1987). The vocabulary problem in human-system communication. *Commun. ACM*, 30(11), 964-971.

- Heckel, F. W. P., Youngblood, G. M., and Hale, D. H. (2009). BehaviorShop: an intuitive interface for interactive character design. In *Proceedings of the Fifth International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2009)*.
- Hill, R. W., Belanich, J., Lane, H. C., Core, M., Dixon, M., Forbell, E., Kim, J., and Hart, J. (2006). Pedagogically structured game-based training: development of the elect bilat simulation. In *Proceedings of the 25th Army Science Conference (ASC 2006)*
- McAnils, C. and Stewart, J. (2008). Intrinsic detail in navigation mesh generation. In Rabin, S. (Ed.), *AI Game Programming Wisdom 4*, (pp. 95-112). Boston: Charles River Media.
- Presagis. (2001). Ai.implant.
<http://www.presagis.com/products/simulation/details/aiimplant/>. November 23, 2008.
- Schwab, B. (2008). Implementation walkthrough of a homegrown “abstract state machine” style system in a commercial sports game. In *Proceedings of the Fourth Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2008)*, 145–148.
- Tuchinda, R., and Knoblock, C. A. (2004). Agent wizard: building information agents by answering questions. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI 2004)*, 340-342. New York, NY, USA: ACM.
- Xaitment. 2009. xaitMove and xaitKnow.
<http://www.xaitment.com>.

Author Biographies

GEORGE ALEXANDER is a doctoral student in the Game Intelligence Group at the University of North Carolina at Charlotte. He received his B.S. in Mathematics and Computer Science, summa cum laude, from UNC Charlotte in 2006. His current work involves improving the BehaviorShop interface to improve usability and reduce user confusion.

FREDERICK W. P. HECKEL is a doctoral student in Computer Science at the University of North Carolina at Charlotte. He received his BA in Computer Science & Political Science from Swarthmore College in 2005, and his MS in Computer Science from Washington University in St. Louis in 2008. His current research focus is in reactive control for agents in games, including evaluation of different reactive architectures and methods to reduce agent complexity.

G. MICHAEL YOUNGBLOOD, Ph.D. is an Assistant Professor of Computer Science at the University of North Carolina at Charlotte and head of the Game Intelligence Group. He received his Ph.D. in Computer Science and Engineering from the University of Texas at Arlington in 2005. His research interests include interactive AI, game knowledge and information structures, and machine and human learning in games.

D. HUNTER HALE is a doctoral student at the University of North Carolina at Charlotte, working in the Game Intelligence Group. He earned his MS in Computer Science from UNC Charlotte in 2008. His research interests include spatial representations for artificial intelligence, computational geometry, graphics and rendering, and artificial intelligence for games and simulation.

NIKHIL S. KETKAR, Ph.D. is a Postdoctoral Researcher in the Game Intelligence Group at the University of North Carolina at Charlotte. He received his Ph.D. in Computer Science from Washington State University in 2009. His research interests include machine learning, data mining, and graph theory.