BehaviorShop: An Intuitive Interface for Interactive Character Design

Frederick W. P. Heckel, G. Michael Youngblood, and D. Hunter Hale
The University of North Carolina at Charlotte
Game Intelligence Group, Department of Computer Science
9201 University City Blvd, Charlotte, NC 28223-0001
{fheckel, youngbld, dhhale}@uncc.edu

Abstract

Artificial Intelligence for interactive characters is held back by the difficulty of actually creating those characters. There is a need for intuitive tools which simplify the process of building complex AI characters. In this paper, we present BehaviorShop, a tool for building interactive characters, and the results of a study to test the effectiveness and ease of use of this tool. The majority (80%) of test subjects were able to successfully create a complex character using BehaviorShop. Our results show success with this interface, and the advantage of using the subsumption architecture as a paradigm for character creation.

Introduction

Progress in interactive characters for games lags behind that of other game technologies and also behind artificial intelligence in general (Nareyek 2004). As graphics and physics calculations have been offloaded to specialized GPUs in recent years, more resources have been freed for use by AI. Despite this, most character AI still uses only very basic techniques that are decades old.

Part of the reason for this is that building artificial intelligence for characters is a very difficult task. The developer not only needs to know how they want the character to behave, but also the AI principles required to make that behavior emerge. This can be especially difficult if the characters are part of a training or cultural simulation, where they must act in very specific ways to conform with the real-life domain. Even though more computational resources are available, creation of richer characters is blocked by the complexity of AI modeling. Just as image editing and desktop publishing have greatly improved with more intuitive tools, character AI can benefit from more effective interfaces. With an easy to learn graphical user interface, AI developers could focus on providing more complex techniques while game designers and creative talent take on the task of actually creating characters.

In this paper, we show our initial work creating BehaviorShop, an intuitive, subsumption-based graphical user interface for building interactive characters such as those in Figure 1. We also present the results of a user study with BehaviorShop, showing that it is easy to learn and usable by AI novices.

Figure 1: Two agents, created with BehaviorShop, interacting in the FI3RST game environment

Background

Tools such as Photoshop, for image editing, and PowerPoint, for presentations, have been around for many years. These programs provide simple interfaces for performing complex tasks, enabling more users to produce high quality images and presentations. The state of design tools for AI is not as strong; while there is some agreement that the basic unit of AI design is a decision or behavior, to our knowledge there is no work on the basic set of behaviors necessary for building agents from the ground up. Discussion during the recent AI Summit at the 2009 Game Developers Conference called for an editing paradigm which allows immediate feedback, similar to that of image and modeling tools, but there was no consensus on how this could be done.

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). Figure 2 shows the RobotFlow interface. The base-level units of RobotFlow are low-level,

allowing a great deal of flexibility when creating new systems. Unfortunately, this level of complexity is daunting for non-expert users.

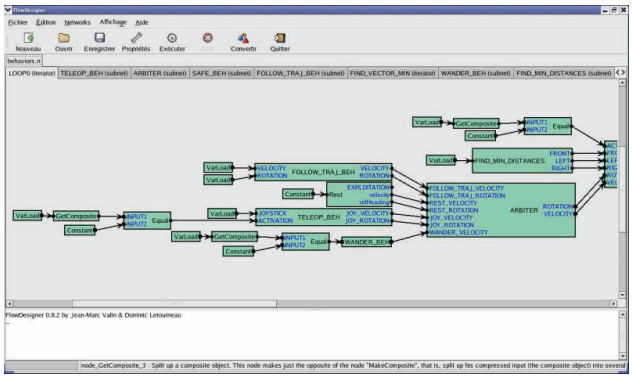

Figure 2: Screenshot of RobotFlow, developed by University of Sherbrooke.

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. While we cannot provide screenshots of these commercial products, the interfaces are similar to, or more complex than, the RobotFlow interface in Figure 2.

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 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.

Cognitive architectures provide testbeds for theories from cognitive psychology. It can be difficult to work with cognitive architectures, because they attempt to be cognitively plausible. Two examples of cognitive architectures are ACT-R and Soar (Anderson 1996; Laird, Rosenbloom, and Newell 1987). Typically cognitive architectures are not appropriate for games (although Soar has been applied to game AI, and ACT-R used to control mobile robots).

Another architecture which is not commonly used in games, but, we argue, is appropriate, is the subsumption architecture (Brooks 1986). Subsumption systems use multiple simple behavior layers which are triggered based on sensor input. Layers do not interact with one another, but the output of higher priority layers may override or *subsume* that of lower level layers. Pure subsumption architecture uses stateless layers and is most appropriate for low-level reactive control, but has been extended as behavior-based control (Matarić 1992). We have found subsumption using behavior-based control to be a very intuitive approach for AI novices.

Design

The DASSIEs (Dynamic Adaptable Super-Scalable Intelligent Entities) Project addresses the problem of building an intuitive user interface for interactive character design (see Figure 3). DASSIEs is composed of three major parts: CGUL (Common Games Understanding and Learning Toolkit), BEHAVE (Behavior Emulating Hierarchically-based Agent Vending Engine), and BehaviorShop (Heckel, Youngblood, and Hale 2009b). Our game environment is FI3RST (FIrst and 3rd-person Realtime Simulation Testbed).

CGUL provides information services to BehaviorShop and BEHAVE, such as navigation services (world geometry and planning) and information tagging through influence points (Heckel, Youngblood, and Hale 2009a). Game events flow to CGUL, which updates its services, and passes processed information on to BEHAVE and BehaviorShop. BEHAVE uses CGUL during runtime, while BehaviorShop polls CGUL during agent creation to provide accurate information about the simulation environment to the user. BehaviorShop also queries BEHAVE for information about available pre-built character behaviors.

BehaviorShop and BEHAVE are based on the subsumption architecture, as we have found that AI-naive users find the layered architecture more intuitive than FSMs when creating agents.

BEHAVEngine

Our primary goals for BEHAVEngine are for it to be scalable, modular, and subsumption-based. It is also a platform for research in discovering an ontology of behaviors—BEHAVE is an important component for finding a minimal set of low-level behaviors that can be used to build any complex behavior that is needed. This requires that it provide a

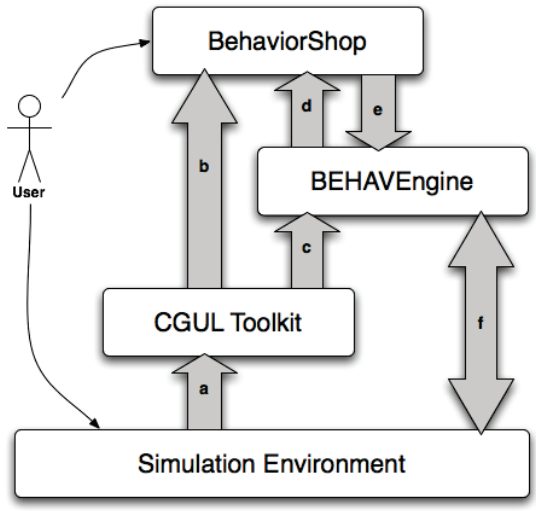

Figure 3: The DASSIEs system: Information flow is labeled with the large arrows showing a) the flow of the environment information into CGUL for processing into knowledge, b) the flow of processed information and created knowledge for incorporation into the behavior construction task, c) the flow of processed information and created knowledge to BEHAVEngine for use in reasoning and agent interaction, d) the flow of agent action information to BehaviorShop for incorporation into the behavior construction task, e) the flow of the behavioral model to BEHAVEngine for creation of the simulation agent(s), and f) the control and perception information interchange between the BEHAVEngine and the Simulation Environment.

hierarchical behavior-building system. Hierarchical behaviors reduce the total number of low level behaviors which must be created, as many important behaviors can be composed of multiple low-level behaviors. BEHAVE serves the purpose of finding an ontology of behaviors by providing a testbed to compare the performance of monolithic behaviors with compound behaviors composed of individual simple behaviors.

Scalability refers both to resource usage and the number of agents it is capable of running. BEHAVE must be resource-aware so that it can run on sub-notebooks and hardened laptops, yet scale up to multicore workstations. Modularity is an important requirement, as it should be easy to experiment with different perceptual systems and add new behaviors to the engine.

We use hierarchical subsumption, so each layer may consist of a full subsumption instance itself. Subsumption potentially suffers from unpredictability as the complexity of the system increases, due to emergent behavior (Simmons 1994). So far this has not been a major issue (possibly due to the simplicity of our scenarios, but we are creating useful characters), but the incorporation of deliberative components as well as parallel subsumptions will help to mitigate this problem in the future. Deliberative components can be used to modify the active subsumption, possibly replacing it entirely in certain situations (such as when the agent is under duress). This eliminates the need for a single model to handle all situations the character may experience.

BehaviorShop

BehaviorShop is a subsumption-based interactive character builder designed to be utilized by users with little to no AI experience. It is capable of querying game services and the underlying AI engine to receive information appropriate to the given scenario. This helps the user ground the character being created in the game environment for a given scenario.

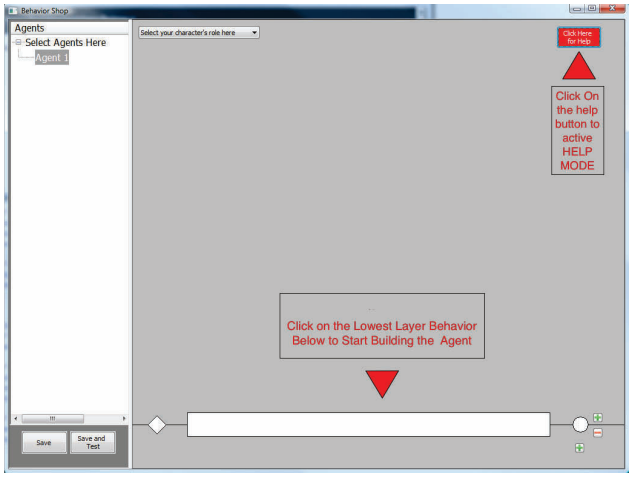

Figure 4: The main subsumption view in BehaviorShop.

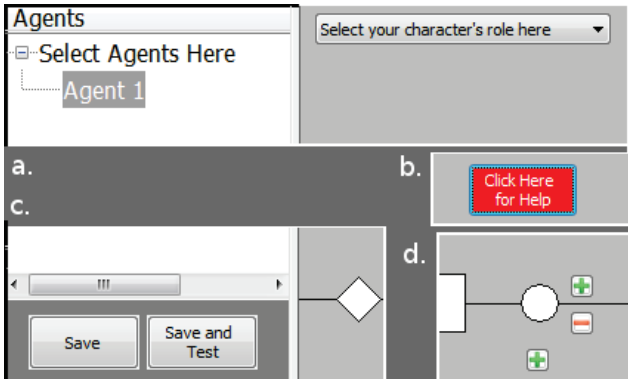

Figure 5: Details of main view: a) The character role selector, b) Help activation button, c) Save buttons, d) Layer creation controls.

When a user initially starts BehaviorShop, they are taken to the screen shown in Figure 4 and detailed in Figure 5. The left pane provides an area where the user can view the currently open characters. The user first selects the role of the agent to be created(5a). The roles provide predefined information for the agent, such as the model to be used, starting location, and default parameter settings. The help button (5b) provides a tooltip-like facility for obtaining information about each area of the builder, and at the bottom of the pane

the first layer is available. When the user clicks on this layer, he is taken to the behavior screen. The + and – buttons (5d) allow the user to add or remove layers from the subsumption. Finally, the user can save the current agent to a file, or save it and view a preview of the agent's behavior in the game engine (5c).

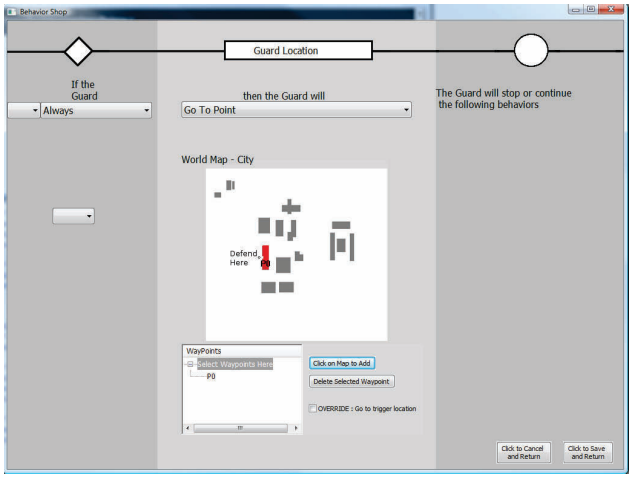

Figure 6: The BehaviorShop specific behavior builder screen. This screen allows the user to define a specific layer in the subsumption agent. Note the map of the environment which is received from CGUL, and the menus which are propagated by querying BEHAVEngine.

The first layer of the behavior is preset to use an "Always" trigger—this is the base layer behavior, and one must always be present. Figure 6 shows the behavior screen for this layer, and demonstrates the flow of information from the information services into BehaviorShop. For the behavior "Go To Point", BehaviorShop provides a map of the game environment so that the user can select the target location in the world.

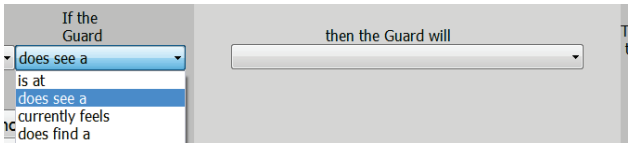

Figure 7: behavior layers are specified as English language sentences

Note that, as seen in Figure 7, behaviors are created as simple sentences: *if the [role] [triggering condition], then the [role] will [behavior response]*. We have found that providing this style of interface makes the interface far more intuitive than simple lists of trigger and behavior names.

In addition to basic *if/then* sentences, trigger conditions can be modified through negation as well as combined as conjunctions or disjunctions. Figure 8 shows the trigger options as well as the list of behaviors. These behaviors are obtained by querying the BEHAVEngine. This allows BehaviorShop to provide access only to the behaviors available

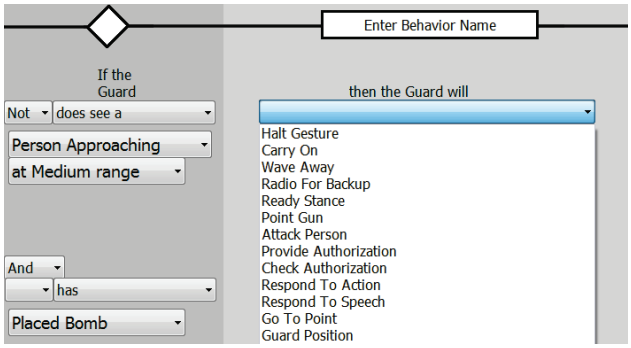

Figure 8: Triggers can be negated using the "not" dropdown and combined with the "and" (conjunction/disjunction) dropdown. Behaviors available to the user are displayed using a drop down list.

to the user in the particular scenario.

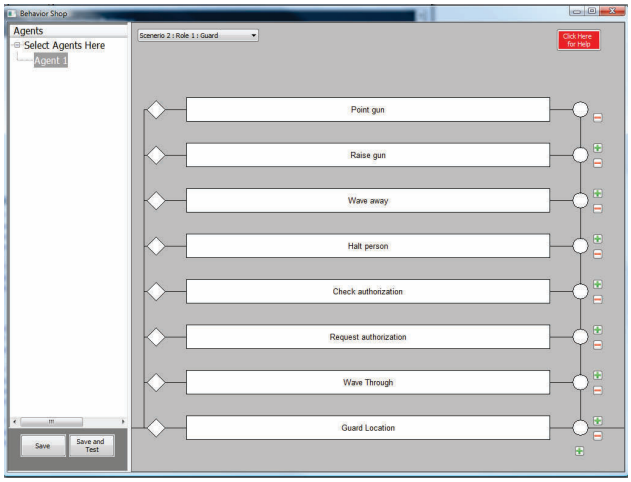

Figure 9: The BehaviorShop subsumption screen. This shows the full layered behaviors for a guard character.

After creating several subsumption layers for a character, such as that seen in Figure 9, the user can test the character using the *Save and Test* button (which can be used at any time during creation), or if satisfied, can save the complete character with the *Save* button to finish the task.

The specified subsumption is translated to a simple behavior description language which is understood by BEHAVE. This language lists all of the behaviors for the character, coupled with one or more triggers per behavior.

Evaluation

During the initial development of BehaviorShop, we created several different versions to explore different types of agent architectures. The initial interface was a FSM builder, but most users had difficulty with this due to the complexity of the transitions. A HFSM version also proved unintuitive, and so FSM-based approaches were quickly abandoned. Surprisingly, the layered builder was immediately

found more intuitive, and most users were able to use it more easily.

During a pilot study at a public library, an earlier subsumption based version of BehaviorShop proved successful. Fifteen subjects participated in this pilot. Each subject was given the task of creating a game character for one of five possible described roles, which involved some complex behaviors (e.g., security for a location, gathering information from other characters, and other military-type scenarios). A single subject quit in frustration, citing basic unfamiliarity with computers. Eleven subjects created characters that behaved correctly and accomplished all key behaviors for the assigned role (as compared with characters created by experts from our lab), and three others created partially correct characters. The pool of subjects was 53% female and covered an age range from just over 18 to mid-50s.

Online Study

Before running full user studies with BehaviorShop, we first created an online study. In the online study, subjects were asked to write instructions for a person filling a specific role in a scenario. One of thirteen possible roles in five different scenarios was chosen, and a high-level description of the role was given. The subject was then asked to write a more precise description of the tasks for a person executing that role. After completing this portion, the subject was shown a video of actors demonstrating another scenario. Following the video, the subject was asked to write a precise description of how one of the actors performed their role.

Descriptions from this study were analyzed for word frequency to find the most common language used to describe behaviors for each role. The results of frequency analysis based on 147 people are being used to adjust the presentation of behaviors and triggers for the interface, but the current results are preliminary. Once we have completed the analysis, it will be used to develop a more complete vocabulary of behaviors and language specific to user profiles for use with BehaviorShop and BEHAVEngine.

User Study

Our first full user study with the complete system (BehaviorShop, BEHAVEngine, and FI3RST) included fourteen subjects, but data from four subjects was unusable. Each subject was asked to build a character filling a guard role. Users were given the following description of the scenario and guard role:

> This is a normal day outside a secure office building. A security guard (Role 1) armed with a gun and a communication radio to the main guard dispatch office stands guard at the entrance to the facility. Authorized personnel must show their badges to the guard to gain entry. An authorized person (Role 2) will enter the building through the doors after showing their badge to the security guard. An unauthorized person (Role 3) is trying to make several attempts to enter the building. They try to run past the guard, trail an authorized person, and other such activity. The unauthorized person does carry a knife. The security guard is authorized to use deadly force if threatened with any type of lethal force.

After being given the description and a 1-minute tool and subsumption technique tutorial, the user was asked to create the guard agent in BehaviorShop.

Before the study, subjects were given a demographic survey. The majority of subjects had substantial experience playing video games, and 7 were male. All of the subjects were in the 18-24 age range. Of the subjects, 6 had no AI experience, while 3 more described themselves as having had a low level of experience; 6 of 10 subjects had either a bachelor's degree or were currently in a bachelor's program, 3 had advanced degrees, and 1 high school education.

Results Of the 10 users with valid data, 8 successfully created agents that filled the guard role. To determine success, the characters generated by the users were compared with a set of reference agents created by members of the research team. 3 users created agents which exactly matched one of the reference agents, while 5 more created agents which were acceptable—they did not match exactly, but used all of the critical behaviors as defined by the AI experts. These measures were also verified by calculating a graph edit distance between the reference agents and created agents. Subsumptions were transformed into a graph where each layer of the subsumption (both trigger and behavior together) was treated as a vertex in the graph, and an edge was created between adjacent layers (ie, the third layer had edges to the second and fourth layers). If the behavior and trigger of a layer matched the reference character, no change was made (distance of zero). If only one of the behavior or trigger was correct, it was considered a distance of 1. If the entire vertex had to be removed and replaced, that counted as a distance of 2. The average graph edit distance for successful agents was 1.25, while the average distance for unsuccessful agents was 7.25. The difference between the two groups was found to be statistically significant at $p < .05$.

We learned several important lessons from this user study. First, contrary to our expectations, the most confusing aspect of the user interface was the negation modifier for the trigger conditions, rather than the conjunction/disjunction feature. No users used the help button, suggesting that a new paradigm for providing support information is needed; this is especially important given that users were sometimes confused by what different triggers and behaviors actually do. Four users attempted to click and drag layers of the subsumption around to rearrange their priorities. Two users actually requested the ability to make more complex behaviors, by linking multiple behaviors to a single trigger, either in series or parallel.

One of the most important discoveries based on user feedback was that no users had difficulty with the basic subsumption concept of triggers and actions. Only brief instruction was given about the subsumption architecture itself, yet it was still a very intuitive paradigm. The *Save and Test* button was extremely important, as users would frequently test out behaviors to see what they would do. This reinforces the notion that some mechanism for providing immediate feedback is needed.

Certain triggers and behaviors were confusing to users. BEHAVE includes a simple mental state model for characters, allowing them to be *scared*, *alert*, or *nervous*. These were chosen for specific scenarios, but, unsurprisingly, they caused confusion, as it is difficult to precisely define what these moods mean. Users had different expectations than the developers about what events would trigger a change in these states. For our guard scenario, initially the character authorization required two steps: *request authorization* and *confirm authorization*. Consolidating these two behaviors into a single more complex behavior greatly reduced the problem.

In the post-survey, subjects were also asked to rate two statements on a 5 point Likert scale: "Creating simulation characters is easy with the DASSIEs Creation Tool" and "Making simulation characters is fun". The average for ease of use was 3.38, with a standard deviation of 0.916, while fun was rated higher—average of 4.5, with a standard deviation of 0.756. While users found creating the characters moderately difficult, no user rated fun lower than a neutral "neither" value, with most users rating fun as "agree" or "strongly agree".

Conclusions

The user study with BehaviorShop shows that our approach is feasible, and that with very little training, individuals with little to no AI experience can build interactive characters if provided the right interface. Most of our subjects for this first study were college students with some computer science experience, but we believe that these results will hold for the general population. We are currently planning to run full scale user studies with 5 different scenarios and 13 different character roles to test this in a diverse population of subjects.

We have learned several useful lessons from this initial work which are informing the further refinement of our interface and AI engine. Important factors to take into account when designing a character designer include the method for displaying information about behaviors, dealing with negation of trigger conditions, methods for providing immediate feedback, and simplifying lists of behaviors to reduce perceived complexity.

Acknowledgments

Special thanks to George Alexander, Nick Crook, Keith Dublin, and Shawn Kirsch for their assistance with the FI3RST simulation environment and data analysis.

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.

References

Anderson, J. R. 1996. Act: A simple theory of complex cognition. *American Psychologist* 51:355–365.

Artificial Technology. 2009. Eki One. http://www.eikone.com.

Brooks, R. 1986. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of* 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. *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* 2:1820–1825 vol.2.

Fu, D., and Houlette, R. 2002. Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games. *IEEE Intelligent Systems* 81–84.

Fu, D., and Houlette, R. 2004. *AI Game Programming Wisdom 2*. Charles River Media. chapter 5.1: The Ultimate Guid to FSMs in Games, 283–302.

Heckel, F. W. P.; Youngblood, G. M.; and Hale, D. H. 2009a. Influence Points for Tactical Information in Navigation Meshes. In *Proceedings, International Conference on Foundations of Digital Games*.

Heckel, F. W. P.; Youngblood, G. M.; and Hale, D. H. 2009b. Making Interactive Characters BEHAVE. In *Proceedings, Florida Artificial Intelligence Research Symposium*.

Laird, J.; Rosenbloom, P.; and Newell, A. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence* 33:1–64.

Matarić, M. J. 1992. Behavior-based control: Main properties and implications. In *Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, 46–54.

Nareyek, A. 2004. AI in Computer Games. *ACM Queue* 59–65.

Presagis. 2001. Ai.implant. http://www.presagis.com/products/simulation/details/aiimplant/. November 23, 2008.

Schwab, B. 2008. Implementation Walkthrough of a Homegrown "Abtract State Machine" Style System in a Commercial Sports Game. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, 145–148.

Simmons, R. 1994. Structured control for autonomous robots. *Robotics and Automation, IEEE Transactions on* 10(1):34–43.

Tuchinda, R., and Knoblock, C. A. 2004. Agent wizard: building information agents by answering questions. In *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*, 340–342. New York, NY, USA: ACM.

Xaitment. 2009. xaitMove and xaitKnow. http://www.xaitment.com.