Making User-Defined Interactive Game Characters BEHAVE

Frederick W. P. Heckel, G. Michael Youngblood, and D. Hunter Hale
University of North Carolina at Charlotte
9201 University City Blvd
Charlotte, NC 28223
{fheckel, youngbld, dhhale}@uncc.edu

Abstract

With the most resource intensive tasks in games offloaded to special purpose processors, game designers now have the opportunity to build richer characters using more complex AI techniques than have been used in the past. While additional CPU time makes improved AI feasible, better tools for building agents are needed to make good interactive characters a reality. In this paper we present the BEHAVEngine and BehaviorShop which enable the creation of rich interactive characters.

Introduction

Artificial Intelligence in video games lags behind AI in general (Nareyek 2004). In the past, graphics have been the major bottleneck in games, resulting in a lack of spare CPU cycles to devote to better AI. With the recent development of powerful GPUs, graphics and even physics calculations have been offloaded to special purpose CPUs. This leaves more system resources available to the AI developer, therefore providing a new opportunity to use state of the art AI techniques in games. There is a further problem beyond system resources though—building AI is difficult and games with tens or hundreds of characters may need a lot of custom AI to provide a rich experience. We propose that the solution to this second problem is to make it easier for designers to build intelligent agents (game characters).

Our current research project is an effort to create the *Photoshop of AI*—an application that allows *anyone* to quickly and easily build intelligent agents. The primary focus of this work is agents for games and training simulations, especially for use by law enforcement, warfighters, and educators. Individuals in these areas may not have a background in AI, but still have useful applications for agents in trainint simulations within their domain.

The DASSIEs (Dynamic Adaptable Super-Scalable Intelligent Entities) project is composed of three major parts: the agent builder user interface, the agent engine to implement the generated agents, and game information services. Our agent builder, BehaviorShop, is designed to provide an intuitive interface for building agents. The BEHAVEngine (Behavior Emulating Hierarchically-based Agent Vending Engine) accepts agent descriptions specified in our agent description language and instantiates the intelligent agents. The CGUL (Common Games Understanding and Learning Toolkit) is a collection of game services which provides information about the game environment, including world geometry (Youngblood, Hale, and Dixit 2008).

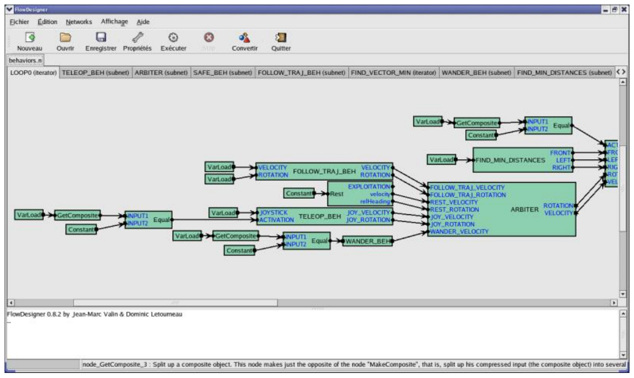
Background

Over the years, many different agent and cognitive architectures have been developed. Our goal is to build an architecture appropriate for gaming that takes advantage of results from cognitive psychology as well as the best practices of this prior work.

In game AI, FSMs (Finite State Machines) are the most commonly used method of driving character AI. They can be used very effectively to build AI, but the number of transitions between states can quickly grow to an unmanageable level. While it is the simplest method of building AI, it can also be very difficult to work with for this reason. Behavior Trees or Hierarchical Finite State Machines are a variation on FSMs, which are also commonly used in games (Fu and Houlette 2004). These can reduce the complexity of the top level FSM, but are still time-consuming to build.

Brooks' subsumption architecture is a layered approach to intelligence that is used in many robotics applications (Brooks 1986). Subsumption-based systems use multiple simple behavior layers which may be triggered based on sensor input. The layers do not interact with one another, but higher priority layers may *subsume* or suppress the output of lower priority layers. Pure subsumption architecture is designed as a reactive control method, but the concept was expanded with behavior-based artificial intelligence (Matarić 1992). Instead of being a top-down approach, like behavior trees, subsumption is a bottom-up approach to intelligence.

Cognitive architectures, as compared to basic agent architectures, aim for psychological plausibility by building systems that take into account results from cognitive psychology. Examples of cognitive architectures include ACT-R and Soar (Anderson 1996; Laird, Rosenbloom, and Newell 1987). One example of the constraints placed upon these systems is the production rule timing in ACT-R; by default, rules take at least 50 ms to fire, and different modules have other timing constraints that have been determined empirically to match with results from psychology. Cog-



FlowDesigner 0.8.2 by Jean-Marc Valin & Dominic Letourneau

node_GetComposite_3 : Split up a composite object. This node makes just the opposite of the node "MakeComposite", that is, split up his compressed input (the composite object) into several

Figure 1: Screenshot of RobotFlow, developed by University of Sherbrooke

nitive psychology provides theories about perception such as the existence of mechanism like visual iconic memory and the phonological loop (Sperling 1960; Baddeley 1997). Iconic memory provides a brief period of retention after visual stimuli have dissipated, while the phonological loop is a mechanism which maintains a short period of aural memory. These concepts may be integrated into cognitive architectures as a way to verify the psychological theory and provide more human-like behavior. Cognitive architectures are primarily research systems (though Soar has been applied to game AI, and ACT-R used to control mobile robots).

Building agents in these architectures can be difficult, and researchers have developed graphical editors for building agents. RobotFlow was produced at the University of Sherbrooke with the goal of creating a graphical builder for robotic systems (Cote et al. 2004). Figure 1 shows the RobotFlow interface with the model of a tele-operated agent application. Behaviors can be dragged into place and connected to set input and output streams. Underneath RobotFlow is the MARIE robotics middleware, which is an extremely flexible and extensible package for building robotic systems.

Several simulation agent builders use FSMs for creating agent AI. SimBionic is a commercial package designed for building game AI, which provides a HFSM agent modeling interface, debugger, and engine (http://www.simbionic.com/). Simbionic is an extension of Fu's BrainFrame software (Fu and Houlette 2002). Another commercial package for simulation AI is AI.implant, a product of Presagis (Presagis 2001). AI.implant allows creation of game agents using a variety of methods, notably FSMs and HFSMs. Because these packages are commercial software, we cannot include screenshots of the agent interfaces in this paper. The commercial interfaces are similar to, the RobotFlow interface in Figure 1.

Agent Wizard builds software agents through a question-based system; the user answers questions on the different facets of the desired agent, and it produces the corresponding AI (Tuchinda and Knoblock 2004). Agent Wizard's approach is very accessible, but is geared toward web software agents instead of game agents. Also, as a result of this approach, the complexity of the agents is limited and Agent Wizard is a more domain specific approach which does not apply to game agents.

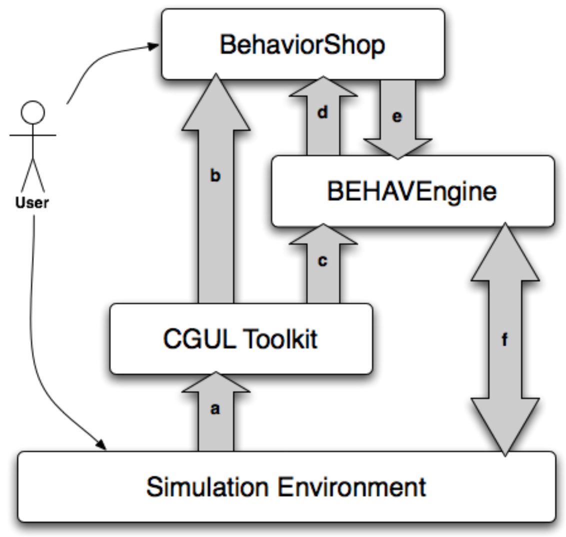


Figure 2: The DASSIEs system: Information flow is labeled with the large arrows showing a) the flow of the environment information into CGUL for processing into knowledge, b) the flow of processed information and created knowledge for incorporation into the behavior construction task, c) the flow of 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.

Design

The previous work in architectures, cognitive psychology, and agent building interfaces has informed the development of the DASSIEs project. While each of the existing architectures and interfaces has its strengths, we have found it necessary to develop new tools. Notably, we have found that subsumption agents are more intuitive to the general public than FSM or HFSM-based agents, and we use our interface to explore this finding more fully. The existing agent architectures are proprietary, defined too strongly by cognitive plausibility, or do not take cognition strongly enough into account. In addition, our goal is to provide a framework for building cultural and social factors into agents to build truly rich characters.

The overall design of our system can be decomposed into three major parts: BehaviorShop, BEHAVEngine, and CGUL. BehaviorShop is our GUI front-end for creating agents, while BEHAVEngine interprets the agent description files. CGUL is a collection of game services which pro-

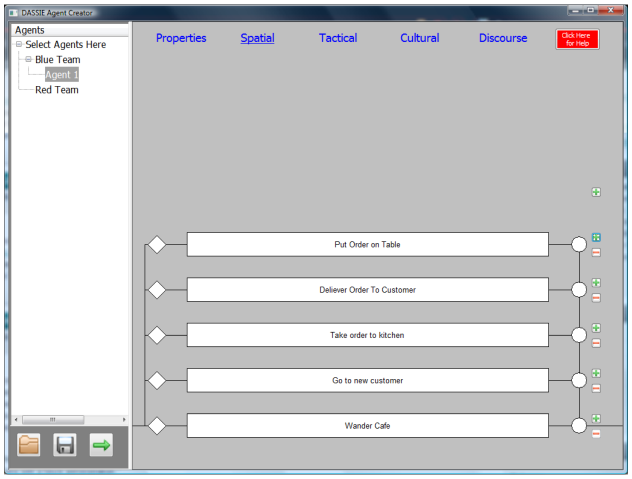


Figure 3: The BehaviorShop Interface main screen. This screen shows the architecture for a barista agent from our Paris Café demonstration.

vide additional information and create knowledge for agent consumption from game world information (eg, CGUL provides world geometry, object information, and fixed world affordances to the agents). Figure 2 shows the full DASSIEs system, and demonstrates the information flow between different components.

BehaviorShop

BehaviorShop allows users to build a subsumption-based agent specification, and is integrated closely with our game services and the BEHAVEngine. Figure 3 shows the main screen of our interface: the user can add additional layers through the + buttons and remove them using the - buttons. Once a blank layer has been added, clicking on the layer brings up the behavior editor (shown in Figure 4), which allows the user to assign a pre-built behavior, any options it needs (such as world points to visit), and under what conditions the behavior will become active (trigger conditions). The behavior editor also allows the user to specify how the current behavior should subsume the output of lower layers; higher layers can completely suppress the output of lower layers, suppress only certain effectors, or *throttle* the behaviors to provide partial subsumption.

BehaviorShop provides the user with information about the environment, behaviors, and trigger conditions which are available to the agent. This information is directly queried from CGUL and BEHAVEngine. Providing this information dynamically guarantees that the user will always have accurate information about the simulation environment and capabilities available for agent creation.

While full agents can be built using only the simple behaviors available through the BEHAVEngine, users can also combine behaviors to build additional complex behaviors. Behaviors can also be modified using *adverbs* which describe both how the behavior executes a task and potentially how the environment should animate the character. Additional depth can be applied to the character by using the

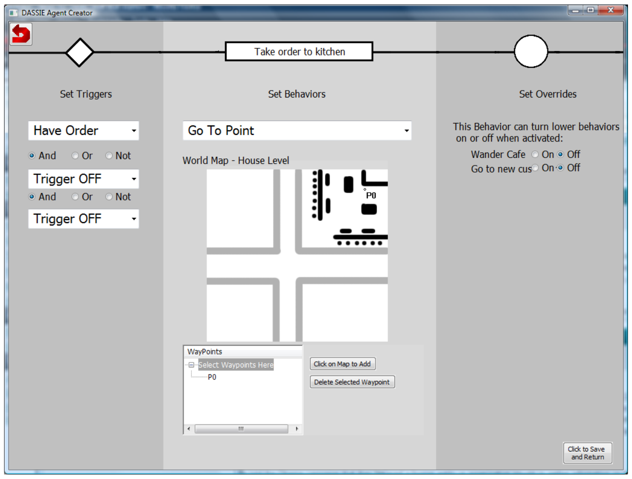


Figure 4: The BehaviorShop Interface behavior screen. This screen allows the user to define a specific layer in the subsumption agent. Note the map of the environment which is received from CGUL, and the menus which are propagated by querying BEHAVEngine.

other agent description categories.

The main category is *Spatial*. Basic agent behavior is described here in this section, as described above. The *Tactical* category provides an alternate mode of action for the agent—tactical behavior is activated when the agent is under duress, as occurs when the agent is involved in a combat situation. The *Properties* tab allows the user to adjust the agent's behavior parameters, which are described in the BEHAVEngine section. Character interaction through speech is created under the *Discourse* tab. Finally, specific cultural values, norms, and beliefs can be added to the agent using the tools in the *Cultural* tab. Cultural factors affect the agent's decisions in a variety of ways, notably in planning.

User-defined agents are then transformed into an agent description language, which is currently a simple translation into a subsumption description. This output is loaded by BEHAVEngine to create the agents at runtime.

BEHAVEngine

We found that existing architectures did not meet our requirements for DASSIEs. Most game AI middleware is proprietary, without the ability to modify the engine itself. Several middleware packages for mobile robotics are available, but mobile robots typically have very noisy and information-poor perceptions of the environment. Interactive characters in virtual worlds can have very clean and information-rich world models. Robotics middleware focuses on providing a common interface for integrating complex software packages; our focus is on combining simple behaviors.

We have a few important design goals for our architecture. The most important goal is to provide a platform for research in discovering an ontology of behaviors. We hope to develop a minimal set of low-level behaviors which can be used to build any complex behavior that is needed. This requires that our architecture provide a hierarchical behavior-

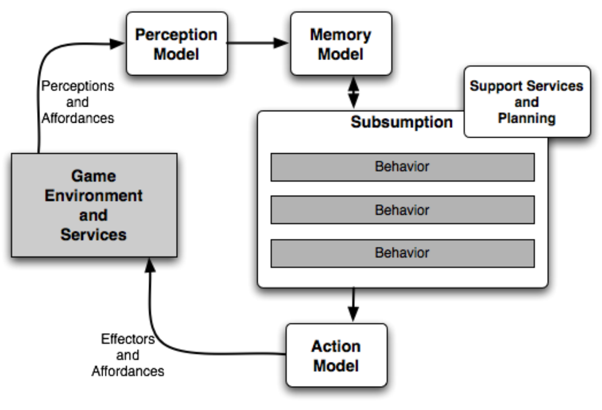

Perception Model
Memory Model
Support Services and Planning
Subsumption
Behavior
Behavior
Behavior
Perceptions and Affordances
Game Environment and Services
Effectors and Affordances
Action Model

Figure 5: The BEHAVEngine architecture diagram

building system.

The architecture must be scalable, in terms of both resources and the number of agents that it can manage at one time. It must run on anything from a resource-poor sub-notebook or hardened laptop to a high-power multi-core workstation. The architecture implementation must be aware of cpu and memory resources, and use any-time techniques to control its resource utilization.

Finally, the architecture must be modular. The first two goals suggest a modularity of behaviors and a modularity of computational components, but modularity goes beyond these two requirements. The architecture should support research in applying cognitive psychology to artificial intelligence, therefore it should be possible to swap in different models for perception, memory, and action.

The basis for our architecture is subsumption. We chose subsumption for its power, as well as its inherent modularity and parallelism. It also provides a simple direct translation of the description language used by BehaviorShop. We use hierarchical subsumption, so each layer may consist of a full subsumption instance itself. As a reactive method, subsumption potentially suffers from unpredictability as the complexity of the system increases(Simmons 1994). Our current agent scenarios are simple enough that it has not yet become a major issue, but we are planning to incorporate deliberative components to mitigate the problem in the future. We find that the explosion in size of FSM-based approaches causes more difficulty than unanticipated emergent behavior in subsumption.

Figure 5 shows the BEHAVEngine architecture. Information is received from the game environment and CGUL services, and passes to the *Perception Model*. The Perception Model filters the available percepts and affordances[1], and passes them on to the *Memory Model*. The Memory Model may store transient percepts in the appropriate short term memory (such as iconic memory for visual percepts) before passing on the currently available affordances and percepts. Behavior parameters and adverbs are also stored in the Memory Model. The *Subsumption Module* is responsible for executing the actual agent behaviors, and includes additional services and planning modules which assist with discourse, tactical, and cultural factors. Information flows bidirectionally between the Subsumption Module and the Memory Model, allowing the behaviors and cultural factors to modify the current adverbs and behavior parameters. Finally, effectors are passed to the *Action Model*, which resolves the actions of the agents. The output of the Subsumption Module includes both effectors and world information in the form of affordances: agents may add dynamic affordances to the world, or change existing affordances.

[1]We use affordances as per Gibson's definition as being actionable properties between the world and an actor (Gibson 1977). We also include the relationships between actors.

Percepts Percepts are received from the game world and CGUL. CGUL provides information about the environment including object information and world geometry based on a world decomposition created using DEACCON (Hale, Youngblood, and Dixit 2008). The most basic percepts provide ego information, such as the agent's current location and action. In addition, affordances are modeled as percepts which are received by the agent when it is within range. Percepts are filtered by the Perception Model and then enter the Memory Model. Some percepts are stored in the Memory Model to provide short term memory allowing the Subsumption Module time to respond to short temporal events.

Layered Behaviors The behaviors within the Subsumption Module are either simple built-in behaviors, or complex behaviors built from combinations of simple behaviors. Each behavior layer includes information about triggering conditions and a policy for subsuming lower priority behaviors. Behaviors make use of the behavior parameters and adverbs that are stored in the memory model, and may change the stored values. In addition, behaviors may query the cultural and discourse information. Tactical behaviors may take over when the agent is under duress.

Behavior Parameters Behavior parameters are values which affect the agent in consistent ways across behaviors. In cases of uncertainty, these parameters provide weight to the options available to the agent. The parameters are also intended to provide information for character animation. Behavior parameters modify actions within the layered behaviors. A subset of behavior parameters provide representations of Hofstede's cultural dimensions (Hofstede 1996). The Hofstede parameters are fixed once the agent is initialized and cannot be modified by the behaviors.

Effectors An agent's effectors either act directly in the game world or can add (or remove) affordances to the world. In a simulation of a café, for example, an agent may first enter the café and sit (acting directly upon the world), and then add an affordance that indicates it wishes to be served. A waiter agent can monitor for customer affordances and respond appropriately. This enables rich interactions between agents without requiring world knowledge to be written into the agents—while it is a goal to build agents which can understand the subtle social cues of every day interactions, in most cases we need an agent which just acts correctly.

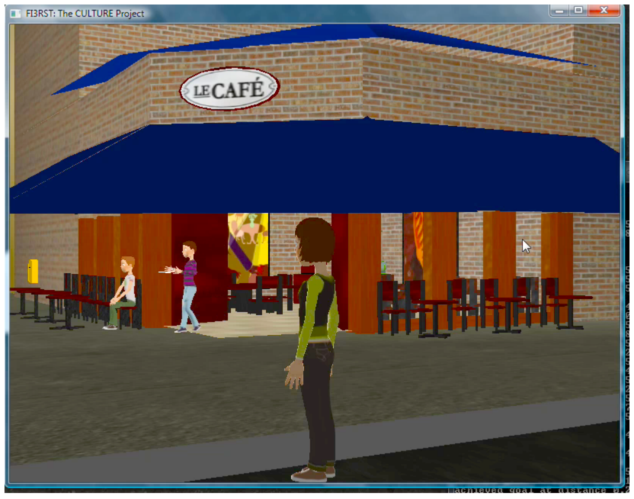


Figure 6: The CULTURE Project: Paris Café Simulation

Action Models are used to filter the agent effectors. This mechanism also combines the effector output of different layers of the subsumption implementation, so that some behaviors may be throttled rather than fully subsumed. The action model also translates effector requests into the form required by the simulation, so changing the action model allows the agent to be attached to different systems. Action effectors may also be modified by adverbs which control how atomic actions are performed.

Affordances and Influence Points Affordances are represented as points attached to regions of the world navigation mesh. They have a detection radius as well as an action radius, which defines the area in which the agent can act upon them. Agents may add affordances to the world directly.

Influence points are added by the BEHAVEngine to provide additional information. These are an adaptation of influence maps (Tozour 2001). Influence maps are used in games to provide tactical information about the world, such as how much power a particular group has in a region, or areas which have recently been dangerous. Influence maps rely on a discrete map of the world, but our navigation meshes allow us to improve on influence maps using influence points. These are treated similarly to affordances (Heckel, Youngblood, and Hale 2009).

Evaluation

Case Study

new

For a public demonstration, we have created a *Paris Café* simulation using BEHAVEngine (see Figure 6). The Paris Café simulation is an effort to create a cultural experience using our engine and is part of a larger project to build cultural simulations that individuals can use to learn about societal norms and expectations in other cultures.

The Paris Café is our first cultural simulation. It is a representation of a street café in Paris, France, complete with crowds, a wait staff, and seating. Users control an agent which can sit at a table inside or outside of the café, watch the crowds pass, and have some coffee. For our prototype of the simulation, we created agents walking through the streets, barista and waiter agents, and a customer agent. The customer walks to the café and sits at a table outside (the chair has an affordance added by the model creator indicating the agent may sit). The customer waits for the waiter (adds a *waiting for service* affordance). A waiter notices the customer (perceives the *waiting for service* affordance at one of his tables) and takes the customer's order (removes the *waiting for service* affordance). The waiter then enters the café, and places the order with the barista (setting a *placed order* affordance). The simulation continues with each agent adding and removing affordances according to the specified rules. The interaction between the agents occurs dynamically as a result of the behaviors, instead of being a pre-written static script.

BEHAVEngine ran the Paris Café simulation with over 20 agents controlled on a single processor core while simultaneously running a game environment. The engine has been tested running 80+ agents simultaneously on a notebook PC, demonstrating the scalability of our implementation.

Pilot Studies

In the process of developing out first version of BehaviorShop we created several versions to try out different forms of agent architectures. We started with a FSM builder that just about everyone had difficulty creating agents with for the reasons previously mentioned, but mostly because people did not know where to start and then had problems wiring the state transitions when the models grew beyond 5-6 nodes. Exploring a HFSM (a.k.a., Behavior Tree) builder also caused most participants difficulty. The initial results from several lab visitors was so poor that we immediately abandoned these approaches. To our surprise, the first tests of a layered behavior builder were met with immediate satisfaction and visitors were able to quickly create layered agent behaviors.

A preliminary pilot study evaluation of an earlier version of BehaviorShop (now subsumption based) conducted in a public library showed very promising results. Fifteen users were given the task of creating a game character for one of five possible described roles using our tool, which involved some complex behaviors (e.g., security of a target location, information gathering from other characters, and other military-type scenarios). Only one subject (male, over 50) quit in frustration citing a basic unfamiliarity with computers. Eleven others created agents that behaved correctly accomplishing all of the key behaviors desired for the assigned role (we compared the subsumptions between these users and ones created from experts in our lab), while three people created agents which performed an incomplete set of behaviors. These incomplete agents did exhibit a number of designed behaviors of similar complexity to those of the desired agents, but appeared more whimsical—reflecting their designer's inclinations that were reported to be their "interpretation" of desired behavior. The sample population ranged in age from just over 18 to mid-50s and was 53% female. No user took longer than 15 minutes to create an agent—members of our research group were able to create similar agents in 3-4 minutes—suggesting that our tool was approachable and intuitive. These results were from a structured, but informal, testing session, but suggest that the current work is on the right track.

Additional early testing with the most recent version of BehaviorShop has also shown positive results. Four subjects, who were visitors to our lab, with little to no AI background have successfully created complete agents with the current release in similar scenarios to our other informal studies. Our IRB-approved testing process covers these pilot studies and in-lab constant testing as well as a full formal study that we are currently about to begin to test BehaviorShop on a large sample population.

Conclusions and Future Work

Continuing work with the DASSIEs project will focus on further refining the interface and creating a much larger set of primitive behaviors for use with the agent builder. There is further work to be done in the translation of user input into the agent description language—this translation process is currently direct, and does not attempt to discover user intent. User intent is important, as it is likely users will make certain common errors which can be corrected through analysis.

Our current agent description language is an ad-hoc XML specification, but we are pursuing an adaption of situational calculus to build a more general language.

We are currently running further studies to evaluate both BehaviorShop and BEHAVEngine in implementing specified agents. For these studies, agents built by human subjects in Behaviorshop are evaluated against agents built by AI experts.

BehaviorShop and BEHAVEngine demonstrate that it is possible to build an agent creation toolkit which is accessible to a general audience with no AI background. Our initial studies show that subsumption-based builders are more intuitive for AI-naive users while still being powerful enough to create complex behaviors.

Acknowledgments

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.

Baddeley, A. 1997. *Human Memory: Theory and practice*. Psychology Press. chapter 4 : The Role of Memory in Cognition: Working Memory.

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.

Gibson, J. J. 1977. *Perceiving, Acting, and Knowing*. Lawrence Erlbaum Associates. chapter The theory of affordances.

Hale, D. H.; Youngblood, G. M.; and Dixit, P. 2008. Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

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

Hofstede, G. H. 1996. *Cultures and Organizations: Software of the Mind.* McGraw-Hill.

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.

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

Sperling, G. 1960. The information available in brief visual presentations. *Psychological Monographs* 74:1–29.

Tozour, P. 2001. *Game Programming Gems 2*. Charles River Media. chapter 3.6: Influence Mapping, 281–297.

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.

Youngblood, G. M.; Hale, H.; and Dixit, P. 2008. CGUL Toolkit. http://playground.uncc.edu/GIG/Projects/CGUL/.