Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A flexible model for real-time crowd simulation

Jürgen Rossmann
Institute of Man-Machine Interaction
RWTH Aachen University
Aachen, Germany
rossmann@mmi.rwth-aachen.de

Nico Hempe
Institute of Man-Machine Interaction
RWTH Aachen University
Aachen, Germany
hempe@mmi.rwth-aachen.de

Philipp Tietjen
Institute of Man-Machine Interaction
RWTH Aachen University
Aachen, Germany
philipp.tietjen@rwth-aachen.de

***Abstract*—This paper will introduce the generic concept of a multi-agent based crowd simulation prototype. The prototype consists of many distinct components that contribute to the system as a whole. Based on the criteria for a human agent model, the agent's module-based, layered architecture is introduced. Subsequently, a closer examination of each module reveals a detailed insight into the agent's architecture. The examined modules are: The agent's finite state machine, its steering behavior, its locomotion model, the path planner and the messaging capability.**

Finally, with respect to the simulation's general architecture, the optimization techniques to further improve the simulation's runtime performance are discussed. This is especially important, since the simulation should run at inter-active frame rates and allow the simulation of as many agents as possible for scalability reasons. The investigated optimization techniques are multi-threading and cell space partitioning. The simulation is implemented in our 3D simulation system VEROSIM, which also handles 3D graphics to visualize the results in real-time.

***Index Terms*—Multi-agent simulation, interactive simulation, multi-threading, software design**

I. Introduction

Crowd simulation deals with modeling the movement of a large number of human characters. Usually, it investigates the effects of group dynamics and crowd psychology, often in public safety planning. The focus of such studies is the behavior of crowds, to predict or explain certain crowd phenomena. Other fields of application of crowd simulations can be found in virtual training and computer graphics imagery.

Because of the highly complex nature of human behavior - such as emotion, stress effects, decision making - no existing representation has been known to be reliable until now. Existing crowd simulation applications do incorporate a few different human behavior models, which can be modified through certain parameters. Sometimes however, it is desired by a designer to apply different behavior models to test and validate the accuracy of a behavior model for a given scenario.

In this sense, the design of an innovative solution should be flexible to be able to incorporate as broad of a variety of different behavioral models as possible. Ideally, it should even be possible to model behavior other than human, such as animal behavior or autonomous vehicle behavior. Secondly, minimal effort should be expended to implement new behavioral models in order to increase productivity and minimize development time. This implies the necessity of designing an agent in a modular way to ease incorporation of different agent components.

At present, the occupants movement and decision-making process is independent of each other within many applications (i.e., pedestrians do not communicate amongst themselves), except for the consideration of crowd density and collision avoidance. This lack of accuracy can be overcome by an event-handling mechanism to allow the modeling of social behavior.

The majority of available crowd simulation applications does not allow simulation in real-time. Whereas this is feasible for simulating static structures, it certainly is not when trying to utilize crowd simulation in the area of virtual training, such as combat operations training or evacuation management drills where interactivity is needed. Hardly any evacuation applications to date seem to support simulating more than 1,000 pedestrians and visualizing them in real-time because most of these applications focus on analyzing simulation results through a post-processor.

The main emphasis of this paper is on the development and prototypical implementation of an efficient and flexible simulation model that utilizes modern hardware to obtain optimal runtime results. To model a human crowd, a multi-agent based approach is used since it is widely recognized that multi-agent simulation leads to plausible results in crowd simulation and provides intuitive modeling of a human crowd from a microscopic perspective.

A module-based agent design is proposed in section III. The agent's modules are a finite state machine administering the agent's different states and in turn its cognitive behavior, a path-planner module, a messaging module to allow for inter-agent communication and a layer defining the agent's physical features. It will be shown that in principle, a module-based agent design allows for the definition of behavior in a way that goes far beyond an exclusive investigation of evacuation scenarios. In this sense, the prototypical implementation exhibits a far greater flexibility then most of the existing evacuation simulation systems.

In section IV, the implementation is optimized using several optimization techniques such as *bin space partitioning* and *multi-threading*, which has only been taken advantage of by very few applications so far. The obtained results, using a representative simulation setup will show, that it is possible to simulate and visualize up to 2,000 agents in real-time using this prototype.

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

II. Related Work

Crowd system performance is distinguished by many factors. There are costs for large simulations and separate costs for producing images of large numbers of characters. Systems differ by the complexity of agent behavior, the graphical level of detail of each agent and the complexity of the environment. All of these factors make it difficult to compare the performance of the different systems.

A categorization by features of each model - such as the modeling method, purpose, space representation, model structure and perspective, methods for simulating movement and behavior, output, use of visualization and CAD drawings - is possible [1]. Usually, the models are using a mix of different approaches in the different categories. For a detailed review of some of these models, please refer to [2], [3], [4]. Also, a slightly different categorization of egress models can be found in [5], [6].

The variety of scenarios which can be used for some of the reviewed applications is often limited to several specific cases. Most applications are only capable of simulating specific kinds of structures they are designed for, i.e. evacuation scenarios. This is because they are specialized to incorporate features to simulate egress situations (occupants responding to smoke conditions, toxicity or heat), but misses the capability for being used for different applications, i.e., military training, CGI or city planning simulation of behavior other than escape-panic. Consequently, these models can give quite good results at the desired fields, in contrast, they often need to be completely redesigned to simulate other cases. For instance, the software Legion 3D from Legion Studios gives good results for pedestrian simulation, however, it cannot simulate more than a pedestrian behaviour.

Many software applications do not support visualization of the simulation in 3D, although those that do, produce output capable of being read by a post-processor virtual reality software or a simple 2D visualization environment. However, visualization in real-time at interactive framerates is only supported by a few applications allowing the simulation of less than 1,000 agents. A short description of closely related models will follow in this section.

High-Density Autonomous Crowds (HiDAC) can be used to simulate different types of crowds, ranging from extreme panic situations (escaping fire) to high-density crowds under normal conditions (leaving a movie theatre) [7]. The developers of HiDAC indicate a number of 500 simulated agents as the maximum number of simulated agents. This relatively small number can be explained by the relatively complex behavior engine of the agents. Multi-Agent Communication for Evacuation Simulation (MACES) is a simulation system that uses the Human Performance Moderator Function Server (PMFserv) as a mature, psycholgical model for human behavior [8]. The simulation system was first released in 2006 and it is said to be able to simulate up to 1,000 agents in real-time. Both of the described crowd simulation systems are still under development and not commercially available to the public yet.

Finally, the Massive Prime software focuses on simulating large crowds for generating CGI in movies. It has been successfully utilized in movies simulating up to 100,000 virtual characters in one scene. The Massive Prime software gives quite realistic results but due to the fact that the needed calculations for such large crowds are highly complex, the simulation is an offline process. Rendering a large scene can take up to several days. Thus, the Massive Prime software is not used for egress simulation.

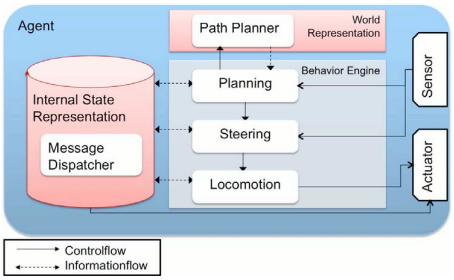

Fig. 1. Agent architecture incorporating a simple HBR

Researchers in the field of rational agent research have developed a wide range of methods, often formal and grounded logics, to support agent reasoning [10], inter-agent communication [11] and autonomous planning and learning [12]. These methods enable unembodied agents to sense and respond to their virtual environments. However, extensive computing resources are needed to support these abilities and therefore the sophstication of an agent's model is highly limited by the number of agents that need to be simulated in real-time.

In the following sections the agent model used in our simulation system will be introduced. Furthermore, optimization techniques that enable the simulation to run at interactive framerates on standard PCs will be presented.

III. The Agent Model

An agent's behavior is described by the *agent function* that maps any given perception sequence to an action. Implementing the agent function results in the *agent's behavior*. The term "behavior" can refer to many meanings.

Human behavior representations (HBRs) are used to model people's behavior to various degrees. They either represent parts of individuals (e.g., walking legs), individuals, aggregates of individuals, or aggregates of organizations. An HBR is used to map cognitive functions (e.g. ,planning, reasoning), performance restrictions (e.g., sensing bandwidth, decision latencies) and the effects behavior moderators have (e.g., stress, fatigue, discomfort) [13].

The minimal configuration of a behavior engine has to perform three functions: It needs to accept input from the environment and update its internal state accordingly, choose additional information from its knowledge base to expand its internal representation of the world state, and generate responses that can lead to achieving its goals based on gathered information and its internal state. The behavior engine determines actions by applying information from the knowledge base to the internal state representation (see fig. 1).

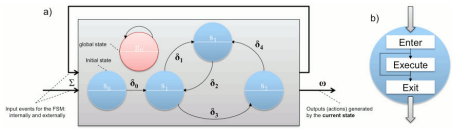

Fig. 2. Event-based finite state machine of the planning layer

The agent architecture defines a structural model of the components that constitute an agent as well as the interconnections of these components together with a computational model that implements the basic capabilities of the agent. Agents in crowd simulation generally should map human features into the simulation. Based on the introduced HBR, a simple model comprising of a few states, sensors and actuators are able to design and implement a prototypical human agent, sufficiently realistic enough to yield characteristical crowd phenomena.

A. Multi Layered Architecture

An agent's behavior can be easier to model if the behavior is separated into different layers. Although it is possible to apply a different layer model, Reynolds' article "Steering behaviors of autonomous characters" [14] introduced a method where movement of an autonomous agent can be broken down into a hierarchy of three levels[1]. This approach is adopted and incorporated within the introduced HBR as its three layers are planning, steering and locomotion.

1) Planning: The aim of a scalable crowd simulation is to have a behavior model that is simple enough to allow real-time simulation of a large number of agents, yet sufficiently complex to provide realistic, believable results. Considering these requirements, a *finite state machine* (FSM) was designed and implemented for controlling the agent's behavior. FSMs decompose a model of behavior into a finite number of states and transitions between them (see fig. 2).

From the software engineering and behavior modeling perspective, there are a couple of good reasons to use FSMs to model agent behavior as they encode context-dependent actions in a set of states. FSMs are simple and quick to code, they enable easy debugging and have little computational overhead. In addition, FSMs are a very intuitive way to model behavior and are highly flexible.

2) Steering Behavior: This layer is responsible for calculating the desired trajectories required to satisfy the goals and plans set by the planning layer. Steering behaviors produce a steering force that describes where an agent should move and how fast it should travel to get there. Steering behaviors define the *microscale movement* for autonomous embodied agents. In contrast to the planning (cognitive) behavior, which can be modeled using FSMs, steering behavior is located on a reflex-based level.

A few of Reynolds' proposed steering behaviors that are fundamental for crowd simulation have been incorporated into the steering behavior module on the basis of the design, that is described by Buckland [16]. Certainly, new behaviors can be developed and integrated.

[1]A similar three layer hierarchy is described by Blumberg and Galyean [15], using different names for the layers: motivation, task, and motor.

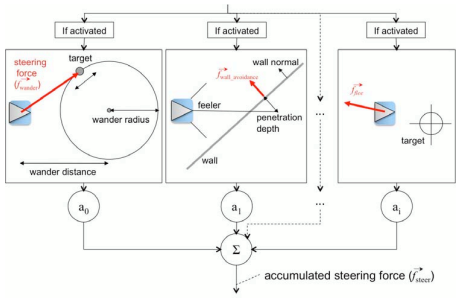

Fig. 3. Simple strategy for negotiating concurrent steering behaviors

Often, a combination of steering behaviors to get the intended behavior is desired. This occurs for example, if an agent needs to follow a path but has to avoid other agents at the same time. A simple approach is to multiply each steering behavior with a weight, sum all of the weights and clip the resulting steering vector to the maximum steering force the agent is allowed to apply (see fig. 3).

3) Locomotion: The foundation layer, locomotion, represents the mechanical aspects of an agent's movement. By separating this layer from the steering layer, it is possible to utilize the same steering behaviors for completely different types of locomotion with little modification.

The locomotion layer translates the steering behavior's "intentions" into movements that are subject to physically imposed constraints such as the limitation of forces that can be applied by an agent. For this prototypical crowd simulation, a simple idealized locomotion model is used based on a point-mass approximation.

The agent's physics are based on forward Euler integration. At each simulation step the desired steering force (clipped to the maximum force) acts on the agent's point mass. This results in an acceleration equal to the steering force divided by the agent's mass. The acceleration and the current velocity are accumulated and limited to the maximum speed. The resulting velocity vector is then added to the agent's current position.

B. Pathfinding

Due to the potentially complex nature of the environments geometry/layout, the agent needs the capability to navigate on paths that have to be determined before traveling from point A to B. To navigate, an agent needs an internal topological representation of the environment that considers places and relations between them. One of the most commonly used topological navigation strategies uses graphs to represent the network of connections an agent may use to move around its environment. An agent's movements are not restricted to moving along the graph edges from node to node, but it can use the navigation graph to negotiate its environment and plan paths between points and traverse them.

To find the shortest paths in an euclidian graph, the edges' costs must be taken into account. Dijkstra's algorithm has

been chosen as the algorithm to realize the prototypical path-planning capability, because it is very efficient and a number of implementations of this algorithm already exist.

An agent's minimum path planning capability has to enable it to plan a path from its current position to any valid position it decides to move to. In certain configurations, the target node is within *line-of-sight* (LOS) of the current agent's position. A simple LOS test is a very fast operation. Hence, the path planner tests for LOS before planning the path, so that in case of LOS-visibility, the costly path planning can be skipped.

A second problem arises from a potentially significant number of agents that inhabit *dense environments*, so that the agents are likely to interfere with each other. Especially in panic simulations this results in the appearance of congestion at *bottleneck configurations*. Typical bottleneck configurations are narrow doorways or tight passages. Also, counterflow can force the agent to deviate from its path and get stuck. To avoid being trapped in such a situation, the agent needs the capability to plan a new path to its target. This is done by calculating the *estimated time of arrival* (ETA) for each node and replan if the time exceeds the ETA.

C. Messaging

Events are broadcasted upon occurrence to all relevant objects in the simulation that in turn can take appropriate action. When endowed with the power to send, handle, and respond to events, it is easy to design complex behavior like panic propagation through screaming or team coordination for squad members.

A message is an enumerated type that has a shared meaning among all entities in the simulation, thus providing a common interface in which events can be communicated. For routing purposes a simple communication protocol was implemented that entails the sender's ID, the receiver's ID, the message and a container for extra information to keep the design flexible in a telegram.

In addition to telegrams, a message routing mechanism *message dispatcher* is responsible for receiving and transmitting messages. This way it is easy for an agent to send a message encapsulated in a telegram to any agent if it knows the agent's ID. Additional sending modes are broadcast and line-of-sight. Finally, it is possible for an agent to just send messages to its near neighbors in a certain range.

IV. Optimization

In terms of virtual reality environments, a slow frame rate can result in lower user performance or, in the worst case, the sense of an animated 3D environment begins to fail. Since it often is necessary to visually track animated objects, frame rates of between 25Hz to 60Hz are considered acceptable.

An embodied multi-agent simulation is a very expensive computation task. This is especially true if moving agents share the same environment and have to take other agents into account while updating themselves. This section introduces two different methods to optimize an existing multi-agent simulation system and then combines both methods to further optimize the simulation.

A. Multithreading

Because multi-core processors are already widely spread at present time, it is reasonable to let simulation take advantage of the chip's architecture by writing multi-threaded applications. The multi-agent system draws its dynamics from the simulation of each agent. Hence, the main work will be carried out by updating all agents in a simulation step. The program's hotspots are parallelized, since these are the sections of code that possibly yield to the highest speedup.

In addition, it is very important to take data dependencies that inhibit parallelization into consideration. Data dependence exists if the order of statement execution affects the results of a program. This might result when the same location of storage is used by different tasks (i.e., collision checks by different agents).

Since the agents' update will be done in the simulation within a single for-loop, automatic parallization was used. The simulation's hotspot is a loop itself, hence using automated parallelization with OpenMP led to significant improvements.

B. Space Partitioning

One of the highly important behaviors that is needed for a plausible human crowd model is collision detection and avoidance of neighboring agents. Because of the dynamic nature and the embodiment of the agents, each agent has to be aware of all other agents in each step of its simulation update. Since every agent has to complete the same task, the computational load for doing so will increase exponentially as the number of agents grows linearly[2]. For a moderate number of entities, this will only have a small impact on the simulation. While multithreading can theoretically double a simulation's performance the by using two threads (quadruplicated with four threads), computational costs of $O((n)^2)$ should be avoided anyway.

For the prototypical implementation, a space partitioning scheme based on bins is implemented to show the optimization potential that space partitioning offers in embodied multi-agent simulations. A grid is projected onto the simulation's environment and each agent is then associated to the cell it is positioned at. Only agents in the same cell or in the cell's neighboring cells can potentially collide. A bounding box around the agent is used to determine the neighboring cells that can be discarded from the agent's further consideration. Finally, only the agents that are located in the relevant cells are being tested for collision by the agent.

C. Results

The two previous subsections introduced two optimization techniques that can be used to improve the simulation system's performance. Consequently, the next step to further improve the performance is to combine these two techniques.

[2]This problem is referred to as the *all-pairs-test* problem. The number of tests is typically $O(nm)$, where n and m are the number of possibilities for each of the two parameters. In an agent simulation where $n = m$ the computational cost is $O((n)^2)$.

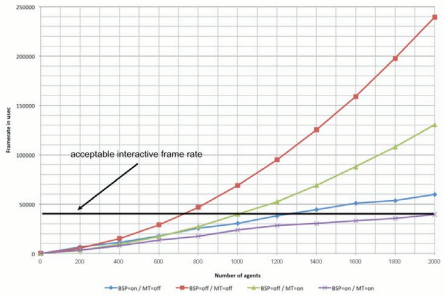

Fig. 4. Optimization results

Figure 4 illustrates a comparison of the frame rates achieved using the different optimization techniques in contrast to the non-optimized simulation. The horizontal axis indicates the number of agents that have been simulated and the vertical axis indicates the average time taken for a simulation cycle to finish. The vertical scale uses microseconds. A minimum update rate of 25Hz is considered acceptable for virtual reality environments. Taking this lower boundary into account, the maximum number of agents that can be simulated without optimization is some 730 agents. Using multi-threading on a duo-core processor increases this number to approximately 1,000 agents. Further improvements are achieved by using bin space partitioning, which allows for the simulation of 1,250 agents with an interactive frame-rate.

Finally, by combining both optimization techniques, significant improvements are obtained. This is due to the linear correlation of the agent number and the time needed for the all-pair algorithms and the parallelization of code execution that is achieved through multi-threading. This combination allows for the simulation of about 2,000 agents in real-time, which is roughly three times as many as without optimization.

V. Scenarios

A few different scenarios have been implemented to demonstrate the simulation's flexibility. Among these scenarios are pedestrian counter-flow studies in densely packed environments, theatre filling, fire-escape panic scenarios, pedestrian simulation in cityscapes and a simple ancient infantry battle scenario.

The fire-escape scenario investigated the most likely-to-be-used evacuation routes visitors of a simple shopping mall might choose in case of a fire emergency. In addition, it could be investigated how these routes emerge depending on the location of the fire source. Also, different agent configurations were used to reveal how the agent's communication capability effects panic propagation (see fig. 5a)).

The same mall layout was later on used for the investigation of emerging traffic routes for agents travelling inside the mall on their individual "shopping-tours" (see fig. 5b)). In this scenario an agent would respond to its desires (drink, eat, medical treatment etc.) based on the behavior model. The colored agents try to get to the corresponding colored areas, the red arrow shows the most frequented path. Additional

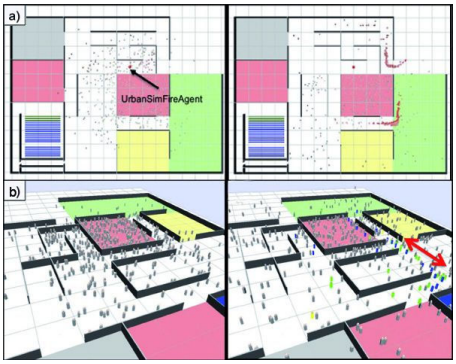

Fig. 5. a) Fire-escape scenario. Agents send fire messages to other near agents in line of sight. b) Busy mall scenario. Agents are getting hungry or thirsty and plan paths to the foodstands. Colored agents try to get to the corresponding colored areas.

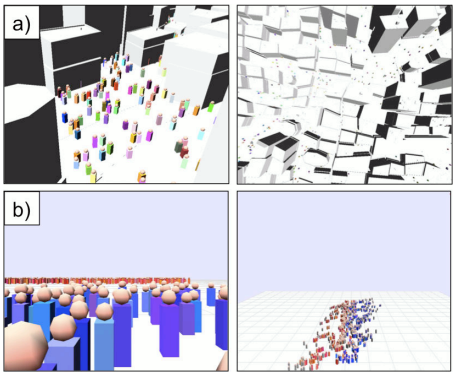

Fig. 6. a) Pedestrian simulation b) Ancient battle scenario

screenshots of the other implemented scenarios are shown in (see fig. 6) where the definition of the agents' FSMs resulted in completely different behavior. In the battle scenario agents frequently send out hit-messages in a close range. Hitted agents get colored grey and try to get back to their base while enemy agents chase them.

The results show how easily and quickly new behavior can be modeled and tested with the prototype. It is possible to

Fig. 7. High quality real-time visualization of the simulation in a city environment

model and simulate a wide range of different behaviors which is not supported by most existing crowd simulation applications. Furthermore, the basic validity of the layered agent approach could be shown as its steering behaviors and the underlying locomotion layer resulted in commonly observed crowd phenomena. In particular, the observed phenomena are line-formation (street-crossing scenario) and combustion and the resulting geometry of crowd bulks at bottleneck configurations (theatre-filling scenario).

VI. Conclusion

In this paper, a flexible and extensible finite state machine for modeling the agent's cognitive behavior was designed and implemented. It offers a high degree of flexibility of behavior representation that allows the simulation to be readily employed in many simulation domains. The use of the singleton software design pattern significantly enhanced the efficiency, since it avoids memory allocation and deallocation. This pays off better, the higher the number of simulated agents becomes. It could be shown, that the complexity of the behavior that can be created with finite state machines is immense. Also, the agents behavior resulted in well known crowd phenomena, that can be observed in real-life accordingly.

Some modules that are used by the human agent prototype were introduced. These were the agent's path planning module and its search algorithm. A powerful event-handling system was designed and put in place, that, in addition to dispatching general events, provides communication capability for the agents. This could be used by agents to interact with each other and thus elevate the modeling flexibility of the human behavior a great deal. The implemented classes are part of this prototypical implementation. Consequently, it is certainly possible to switch the implemented modules with new modules yielding additional functionality.

Finally, to gain maximum scalability, the simulation was optimized using multi-threading and cell space partitioning. Both techniques were investigated and implemented and their results were compared in relation to using no such techniques. The combination of both techniques, resulted in further improvements that accelerated the simulation by a factor of three. It was possible to simulate 2000 agents in real-time. The multi-threading results are based on an Intel E6550 2,33GHz machine; yet, better results would be expected using quad- or other multiple-core machines.

VII. Outlook and future work suggestions

In the field of crowd simulation many crowd models do exist, which cannot be modified or extended to integrate new factors, modules or code from other models. For future development in the different fields of human simulation, better, standardized interfaces are required to benefit from the diversity of achievements from the various related fields.

On the practical side a few improvements considering the implemented prototype are certainly possible. To begin with, algorithms to automatically generate navigation graphs from existing geometry could be utilized. A possible implementation could incorporate the generation of a *shortest-path roadmap* as proposed by LaValle [17] to accelerate a simulation's setup, as the graph's points of visibility don't have to be placed manually anymore and connecting edges would be identified on the fly.

Finally, it would be interesting to incorporate *partial topographical knowledge* of an agent's environment. That is, at a simulation's beginning an agent does not have access to the whole navigation graph of an environment and needs to explore the world first to obtain that knowledge. Furthermore, the effect of agents communicating their individual knowledge would be an interesting topic to address.

References

[1] Erica D. Kuligowski and Richard D. Peacock, *A review of building evacuation models*. Technical report, US Department of Commerce, Fire Research Laboratory, 2005.

[2] ED Kuligowski, *Review of 28 egress models*. Workshop on Building Occupant Movement During Fire Emergencies, pages 68-90, 2005.

[3] Gabriel Santos and Benigno E. Aguirre, *A critical review of emergency evacuation simulation models.*. In Building Occupant Movement During Fire Emergencies, 2004.

[4] R Leggett, *Real-Time Crowd Simulation: A Review*. 2004.

[5] Hubert Ludwig Kluepfel, *Models for crowd movement and egress simulation*. Traffic And Granular Flow'03, 2005.

[6] Christian J. E. Castle, *Guidelines for assessing pedestrian evacuation software applications*. Technical report, University College London, Centre for Advanced Spatial Analysis, 2007.

[7] N. Pelechano, J. M. Allbeck, and N. I. Badler, *Controlling individual agents in high-density crowd simulation*, In SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation. pages 99?108, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[8] Barry G. Silverman, Michael Johns, Jason Cornwell, and Kevin O'Brien, *Human behavior models for agents in simulators and games: part i: enabling science with pmfserv*. Presence: Teleoper. Virtual Environ., 15(2):139-162, 2006.

[9] Legion Ltd., *Legion Studio 2006*. Website, 2003. Available online at:http://www.legion.com/ ; visited on 2008-11-26.

[10] Jeffrey M. Bradshaw, Mark Greaves, Heather Holmback, Tom Karygiannis, Wayne Jansen, Barry G. Silverman, Niranjan Suri, and Alex Wong, *Agents for the masses?*. IEEE Intelligent Systems, 14(2):53?63, 1999.

[11] Y. Labrou, T. Finin, and Y. Peng, *Agent communication languages: The current landscape*. In IEEE Intelligent Systems and their Applications, number 12, pages 45-52, 1999.

[12] Milind Tambe, W. Lewis Johnson, Olph M. Jones, Frank Koss, John E. Laird, Paul S. Rosenbloom, and Karl Schwamb, *Intelligent agents for interactive simulation environments*. AI Magazine, 16:15-39, 1995.

[13] Simon R. Goerger, Michael L. McGinnis, Rudolph P. Darken, and Michaeil J. Kwinn, *A validation methodology for human behavior representation models.*. Technical report, Operations Research Center of Excellence, 2005.

[14] Craig W. Reynolds, *Steering behaviors for autonomous characters*. Game Developers Conference 1999, 1999.

[15] Bruce M. Blumberg and Tinsley A. Galyean, *Multi-level direction of autonomous creatures for real-time virtual environments*. In SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and inter-active techniques, pages 47-54, New York, NY, USA, 1995. ACM.

[16] M Buckland., *Programming game AI by example*. Plano, Texas: Wordware Pub., 2005.

[17] Steven M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.