CAROSA: A Tool for Authoring NPCs

Jan M. Allbeck

Department of Computer Science,
Volgenau School of Information, Technology and Engineering,
George Mason University, Fairfax, VA 22030

Abstract. Certainly non-player characters (NPCs) can add richness to a game environment. A world without people (or at least humanoids) seems barren and artificial. People are often a major part of the setting of a game. Furthermore, watching NPCs perform and have a life outside of their interactions with the main character makes them appear more reasonable and believable. NPCs can also be used to move forward the storyline of a game or provide emotional elements. Authoring NPCs can, however, be very laborious. At present, games either have a limited number of character profiles or are meticulously hand scripted. We describe an architecture, called CAROSA (Crowds with Aleatoric, Reactive, Opportunistic, and Scheduled Actions), that facilitates the creation of heterogeneous populations by using Microsoft Outlook®, a Parameterized Action Representation (PAR), and crowd simulator. The CAROSA framework enables the specification and control of actions for more realistic background characters, links human characteristics and high level behaviors to animated graphical depictions, and relieves some of the burden in creating and animating heterogeneous 3D animated human populations.

Keywords: Non-player characters, Virtual crowds, Agent behaviors

1 Introduction

As we journey through our day, our lives intersect with other people. We see people leaving for work or school, waiting for trains, meeting with friends, hard at work, and thousands of other activities that we may not even be conscious of. People create a rich tapestry of activity throughout our day. We may not always be aware of this tapestry, but we would definitely notice if it were missing, and it is missing from many games.

Most crowd simulations to date include only basic locomotive behaviors possibly coupled with a few stochastic actions. In the real world, inhabitants do not just wander randomly from location to location. They have goals and purposes that related to locations, objects, and activities. Military training simulations, virtual environments, games, and other entertainment enterprises, could benefit from varied, but controlled character behaviors to promote both plots and presence. Furthermore, the creation of these heterogeneous populations with contextual behaviors needs to be feasible. We do not want to demand that scenario

authors be expert programmers. Ideally, they would be storytellers and creative visionaries. Likewise, defining these populations should not be an arduous, never ending process requiring levels of detail beyond what is immediately important to the scene author. An author should be allowed to concentrate on what is most critical to furthering the effectiveness of their virtual world and easily able to explore different possibilities.

In this paper we present a framework for creating and simulating what we term functional crowds. These are populations of virtual characters that inhabit a space instead of merely occupying it. The CAROSA (Crowds with Aleatoric, Reactive, Opportunistic, and Scheduled Actions) framework includes parameterized object and action representations, a resource manager, role definitions, and a simple calendar program. These components are vital to creating contextual behaviors in a heterogeneous population without overburdening the scenario author. To further increase richness, the CAROSA framework also includes four different types of actions: Scheduled actions arise from specified roles for individuals or groups. Reactive actions are triggered by contextual events or environmental constraints. Opportunistic actions arise from explicit goals and priorities. Aleatoric or stochastic actions are random but structured by choices, distributions, or parametric variations.

2 Related Work

As noted by Bates, in games as in life people are a part of setting [1]. Virtual characters can help establish context, provide emotional elements, and help drive a storyline, but to do so their behaviors need to be reasonable and appropriate. To be practical for a game, the mechanisms for authoring these characters need to be simple enough for non-programmers, but also provide enough freedom for designers to drive the story. In [11], Sanchez-Ruiz et al. present a framework for using ontologies and planning to automatically construct solutions that game designers can incorporate into *Behavior Trees*, but the results are still complex trees that can be difficult to manage and debug.

In many games and simulations, populations of non-player characters (NPCs) are programmed to follow a path and perhaps perform a few behaviors. They might also react to a certain set of stimuli, but they do not generally interact with many objects in the environment. They lack context, often only existing when in the players field of view [3].

Many entertainment companies, including producers of movies and games, use software such as Massive to build background characters [8]. Massive^{TM} provides mechanisms for creating and executing rules that govern the behaviors of characters. Creators can then replicate these behaviors and characters to produce very large populations. While creating and refining these rules still takes time and skill, the software makes construction of relatively homogeneous crowds much easier. There can even be some statistical variations in the models and the portrayals of the behaviors. While this is well suited to big battle scenes where the troops and their behaviors are all similar and object interactions are limited

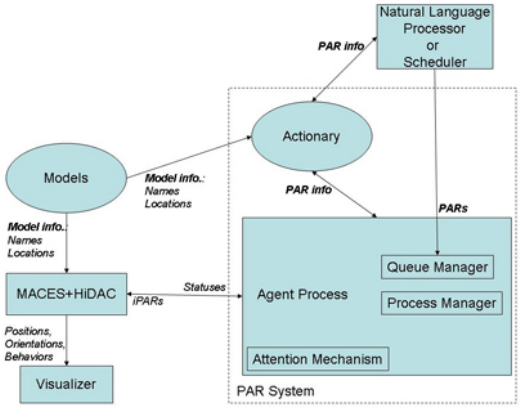


Fig. 1. System diagram

to weapons attached to the characters, scenes that require functional, contextual characters are not feasible.

Much of the research in crowd simulations has been focused on maneuvering characters through a space while avoiding collisions with each other and the environment [10]. Other crowd simulation researchers have examined population diversification. McDonnell et al. have been studying how perceptible variations in character appearance and motion qualities are to human observers with the ultimate goal of simulating more visibly interesting, reasonable populations [9]. Lerner et al. use real world data to fit behaviors into pedestrian simulations [7]. In their work, character behaviors, such as two adjacent characters chatting, are based on the current spatial configuration of the characters in the environment. Other researchers have created variations in behavior by defining regions in the environment where certain behaviors would be displayed [4, 6, 12]. For example, a theater is labeled with sit and watch behaviors. However, within these regions behaviors are uniform. While these research enterprises have achieved more visually interesting populations, the character behaviors still lack larger purpose and context. Decision networks have been used for action selection [13], but they require crafting the prior probabilities of each action in context.

3 System Overview

The CAROSA framework includes many components common to game pipelines (See Figure 1). We are currently using the open source graphics engine, OGRE, for visualization. We are using HiDAC+MACES as an underlying crowd simulator [10]. HiDAC+MACES allows us to navigate characters from one location to another while avoiding collisions and providing us with notification when objects and other agents are perceived. Calls to playback animation clips also filter through HiDAC+MACES to the OGRE figures.

CAROSA's less common components include an action and object repository called the *Actionary*, *Agent Processes*, a *Resource Manager*, and a *Scheduler*. Scenario authors can use the *Scheduler* to schedule actions found in the *Actionary* and link them to locations or specific objects. These actions are, of course, also associated with the agents or characters that are to perform them. *Agent Processes* receive and process these actions, using the *Resource Manage*r to allocate or bind any object parameters needed by the actions. The next few sections will describe some of these components in a bit more detail and discuss how they facilitate the creation of functional, heterogeneous populations.

4 Parameterized Representations

A key component of the CAROSA framework is the PAR system [2]. PARs include both parameterized actions and objects and mechanisms for processing them. The database of actions and objects, termed the Actionary, is a repository for general definitions or semantic descriptions. Both actions and objects are defined independent of a particular application or scenario. They are meant to be reusable building blocks for simulations. The Actionary currently contains more than 60 actions and nearly 100 object types.

We will concentrate on a few key aspects of PARs that help ground character behaviors in proper contexts. First, action definitions include *preparatory specifications*. These parameters provide a simple form of backward chaining that greatly reduces the burden on the scenario author. For example, in the simulation of a functional population, nearly every instantiated action is tied to an object participant (e.g. *sitting in a chair, using a computer, eating food, drinking coffee, etc.*). The character performing an action needs to walk to the associated object before the action can be performed. Requiring a game designer to specify these walk actions and other actions handled by preparatory specifications would more than double the amount of work required.

Like actions, the objects in the Actionary are stored in a hierarchy where children inherit parameter values from their parents. For objects, this hierarchy is driven by types (e.g. *Furniture, Person-Supporting-Furniture, Sofas, Chairs, etc.*). General action definitions, those that have not been instantiated for execution, then reference these types as object participant parameters. Hence, a general definition of *Sit* includes *Person-Supporting-Furniture*. Incorporating these types of commonsense items again lessens the work of scene authors who are not required to specify them. In the next section, we will discuss how the *Resource Manager* can be used to instantiate specific objects of the needed type. Our objects also include parameters such as sites that further specify how characters interact with them. They indicate, for example, where a character should stand to operate the object or how to grasp the object.

To increase the richness of simulations, we have extended the PAR representations to include different action types. Scheduled activities arise from specified roles for individuals or groups (See Section 6). Scheduled actions provide structure and control to simulations. Through scheduled actions, characters have

purpose. Scheduled actions can be used to establish a daily routine or drive a storyline. For example, the game designer may schedule a meeting between two characters to show that they have an association or are a part of the same group (e.g., a gang or terrorist cell). We believe, however, that scheduled actions alone would result in characters that are too structured or robotic.

Reactive actions are triggered by contextual events or environmental constraints. Many of these behaviors arise from the underlying crowd simulator. For example, reactive behaviors include a character altering its heading or slowing to avoid collisions. Reactive actions, such as acknowledging someone as they pass in the hallway, are not handled by the crowd simulator. These reactions are specified and recognized in a ruled based *Attention Mechanism*. We believe that reactive actions help to add life to simulations. Perceiving and interacting with a dynamic environment indicates that the characters are not focused solely on achieving a specific goal. Reactive actions might also be used to create character flaws and emotional moments (e.g. mugging anyone walking alone, cowering in fear whenever they see a dog, avoiding police officers, etc).

Opportunistic actions arise from explicit goals and priorities. These need fulfilling behaviors are akin to the hill-climbing behavior choices of characters in the video game The Sims. While our opportunistic actions are similar, the implementation is different. In The Sims current proximity to resources is heavily weighted when choosing an action, and time is ignored entirely. We take into account time and future proximities. For example, a character may be working in his office and have a non-emergent energy need and a meeting to attend in a few minutes. The character could then attempt to address the need by stopping by the lunch room for a cup of coffee on the way to the meeting. This requires them to leave a couple of minutes early and to know that the lunch room has proximity to the path to the meeting room. Finding an object resource is essentially done through a depth-limited depth first search where the depth limit is based on the need level. As the need level increases, so does the distance the agent is willing to go out of their way to fulfill the need. In fact, as the need increases, so will the priority of the action that will fulfill it. If the need becomes great enough (higher priority than other actions), all other actions will be preempted or suspended. Opportunistic actions add plausible variability. Character begin each new run with random need levels and therefore fulfill their needs at different times creating emergent behaviors by reacting to the different stimuli they will encounter. These actions can also be used to establish character flaws and promote a storyline (e.g. characters that require large amounts of sleep, excitement, attention, alcohol, drugs, etc).

Aleatoric actions are random but structured by choices, distributions, or parametric variations. Take for example, working in an office. The aleatoric or stochastic nature of the behavior stems from what the sub-actions might be and their frequency of occurrence. For example, many professions entail working in an office for large portions of each day. Working in an office might include, using a computer, speaking on a telephone, and filing papers (See Figure 3(b)). These tasks are interspersed through out the day. A game designer could schedule each

of these tasks during different segments of the day, but for most scenarios the exact timing of each sub-action is not important. The overarching *WorkInOffice* action should just look plausible. With aleatoric actions, the author need only specify *WorkInOffice* and the sub-actions will be performed in reasonable combinations. Each time a scenario is run, different combinations of actions will be chosen creating reasonable variations for the player encounter.

5 Resource Management

As we have stated, creating functional populations whose behaviors are contextual, involves associating actions with related objects. Making these associations by hand would overwhelm a game designer. We have implemented a *Resource Manager* that is used to automatically allocate objects.

Resources can be initialized from the environment definition using defined spaces and object model placement. This automated process also instantiates PAR objects and stores them under their corresponding types in the *Actionary*. Additionally it sets up location-content relationships. Object locations are set as the room that they are placed in and likewise the objects are listed as contents in those room objects. These relationships are then used to initiate resource management. There are different layers of resources. The objects in a room are resources of that room. Rooms themselves are also resources. A resource group is created for each room and all of the objects contained in that room are added as resources. Also, every room is added to the room resource group. Finally, all of the objects in the environment are added to a free objects list. This list is used to allocate objects for actions that do not have a location specified. For example, a housekeeper might be asked to clean anything. A random object would be chosen from the free objects list and used as a participant of the action. It is also possible to specify the type of object needed. This list is automatically updated as the Resource Manager allocates and de-allocates objects to agents. The entire resource allocation process itself is done automatically as a consequence of processing PAR actions and does not require game designer intervention.

Resources can be allocated according to type or a specific object instance can be requested. Furthermore, a preference function can be used to determine the best object to allocate to the character. This function argument could be used to indicate that particularly shady characters prefer to sit in the back corners of a restaurant, for example. It should also be noted that objects are not allocated to characters until the characters are at least in the target location or room of the action. While resources could be allocated at the time the PAR actions are assigned to the character, we feel this method leads to more natural behaviors including failures. For example, a character should have to walk to a room to know that there are no more chairs available.

6 Roles and Groups

Peoples functions or purposes through the course of a day are highly correlated to their roles: Students attend classes, Professors work in their offices and lecture, Housekeepers clean, Researchers research, etc. Creating roles for game characters provides them focus and purpose. Defining groups of characters with common roles means reduced work for game designers.

Roles also give us a plausible starting point for enabling the assignment of default behaviors and locations that can lessen the load of an author. When they have nothing else to do, professors work in their offices, researchers conduct research in a lab, housekeepers clean, etc. This means that even if they have no scheduled activities, their behaviors will at least look plausible. Note that all of these actions, like most, require object participants. The professor needs an office to work in, the researchers need a lab desk, and the housekeeper needs something to clean. The question is how do we determine or allocate objects for default behaviors. Certainly we do not want to demand that a game designer assign such objects for every character in the population. We also do not want to just randomly choose objects whenever any of the default behaviors are to be performed. This would lead to inconsistent scenarios. Professors do not randomly choose offices to work in throughout a day. We have created a couple of mechanisms for automatically choosing appropriate object participants providing a more consistent context.

First, when defining a role, one can specify objects that the role should possess. For example, a professor should have an office. An author does not need to specify an instance of an object, though they are permitted. It is sufficient to specify a class like *Office*. Possessions are transitive. If you have an office, you also have the objects in the office. When a role has a possession specified, every character with that role is allocated (by the *Resource Manager*) a possession of that type during the initialization. Whenever a character initiates the performance of her default behavior, she first checks to see if an object of the type needed is located in her possessions. If it is, she uses that object. If it is not, the next method is tried.

This method also uses the association of object types with default actions. For example, researchers in our scenario need laboratory desks in order to do their research. As we did with professors, we could say that all researchers should possess a lab desk, but in a setting where resources are limited it might be more likely for researchers to take whatever desk is available. We do this by indicating that the action *Research* requires an object participant of the type *LabDesk*. When a researcher initiates this default action, an object of type *LabDesk* is allocated to her. If no object can be allocated, a failure is produced and the action will not be performed. This might lead to the character performing a different action or just standing still (or idling as a default action) until an appropriate object can be allocated.

A key aspect of this paper is facilitating the creation of heterogeneous populations. To achieve this, we need to provide a way to define groups as well as individuals. An author can name a group and provide the number of members

in it, or an author can create individual characters and assign them to groups. In both cases, the information is stored in the *Actionary*. As needed, agents are created to fulfill the indicated membership numbers. If the name of a group happens to correspond to the name of a role, then all of the members of the group are automatically assigned that role and inherit all of the default behaviors and specifications for it. There can be more than one group per role and there may be groups that do not indicate a role. Entire groups can be assigned actions (of any type) as easily as a single individual.

7 Game Authoring

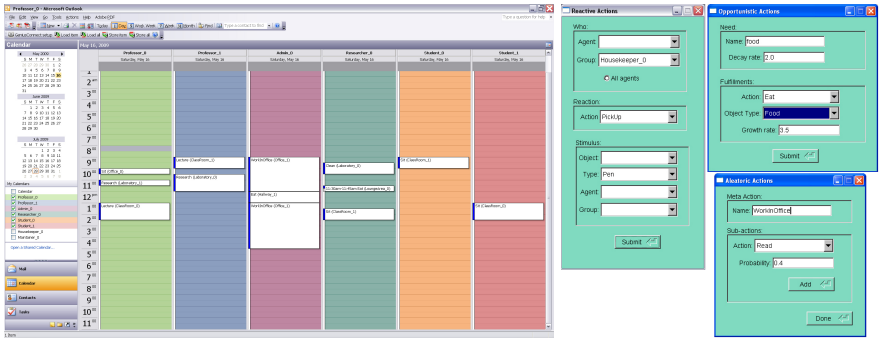
Beyond creating functional, heterogeneous crowds, our goal for the CAROSA framework was to facilitate authoring by non-programmers. Certainly to simulate a virtual environment, one will still require models and animations, but once a repository of these items has been created and a populated *Actionary* obtained and linked to them, a game designer should be able to use these building blocks.

When scheduling actions, an author only needs to specify the PAR action that should be performed, which character or group of characters should perform it, what objects or locations might participate in the action, and when it should be performed. In fact, the actions can either be simple PARs or complex lists of PARs composed together. A specific object participant can be specified, such as sit in *Chair_4* or a location can be given, such as sit in *ClassRoom_1*. If a location is specified, a chair instance will be allocated by the *Resource Manager* when the character arrives in the specified location. If the required resource cannot be allocated, a failure is reported to the *Agent Process* and the action is removed from the queue. In the future, a planner could be used to attempt to acquire the resource needed.

We are currently using the calendars of Microsoft Outlook®as a scheduling interface (See Figure 2(a)). It is a common, well-known software package that does not require users to be programmers. Calendars can be created for groups of characters or individuals as desired. Activities, locations, and times are then simply entered on the calendars. Then with a simple click of a button, a software package called GeniusConnect [5] links these calendars to tables in our MySQL database (i.e., the *Actionary*). These activities are read from the database and assigned to the appropriate character or characters by placing them on the appropriate *Action Queues* of the *Agent Processes*.

We have constructed simple custom Graphical User Interfaces (GUIs) for authoring reactive, opportunistic, and aleatoric actions (See Figure 2(b)). These GUIs are directly connected to the *Actionary*. The drop-down lists are populated from table entries in the database and submitting new actions of these types are written directly to database tables. These newly created actions can then also be referenced from Microsoft Outlook to, for example, schedule aleatoric actions.

The PAR representation of objects and actions provide semantic foundations that are referenced when authoring through any of these interfaces. The *Resource Manager* provides a means of filling in information and tying the behaviors to



(a) (b)

Fig. 2. (a) Microsoft Outlook®calendars are used as an interface for scheduling actions. (b) Custom GUI's for authoring Reactive, Opportunistic, and Aleatoric Actions.

the environment. This allows a game designer to concentrate on aspects of the characters that are directly relevant to her storyline including the heterogeneity and functionality of the population. The precise nature of these interfaces is not important. What is important is that a limited, manageable amount of data is required to specify the behaviors of the characters. All of the required data could just as easily be specified in and then read from a spreadsheet or a simple text file.

8 Example Simulations

As an initial test-bed for the CAROSA framework, we simulated a university building environment. The environment is based on one floor of an engineering building at a university. It includes laboratories, classrooms, hallways, offices, a restroom, and a lounge (See Figure 3). The occupants of the building include professors, students, researchers, administrators, housekeepers, and maintenance personnel. Actions in this environment include working in an office, lecturing, attending class, lounging, cleaning, inspecting, researching, waving, eating, drinking, going to the restroom, picking up objects, as well as others. There is also collision-free locomotion.

Characters in the simulation adhere to their schedules as created by the scenario author through the *Scheduler*, but they also greet each other as they pass by and attend to their needs through opportunistic actions. If a portion of their day is unscheduled, the characters revert to their default behaviors which are in many cases aleatoric actions that provide ongoing reasonable behavior.

We ran many simulations of this environment and noted several emergent behaviors. For example, students tended to gather in the same areas because of the resources available and therefore tended to walk to classes together. Furthermore when groups were instructed to react to other group members by waving,

(a) (b)

Fig. 3. Sample scenario based on activity in a university environment.

students would greet each other before a class which seems like reasonable behavior. Students also tended to go directly for food and coffee after classes. Because need levels are currently all checked together, needs tend to be fulfilled at the same time. For example, a character might go to the restroom and then to get coffee. Again this emergent behavior seems quite plausible.

We have also authored a scenario where a human commander and a robot team are tasked with clearing a building of weapons (See Figure 4(a)). This scenario includes searching the environment, picking up and dropping weapons, and alerting the commander to the presence of other beings. Reactions include alerting the commander to objects of certain types and other characters. For the robots, the main need is battery power, which is fulfilled by charging. Aleatoric actions can be used to vary the behaviors of the robots (including navigation routes) decreasing predictability that may be exploited by hostiles.

In addition to the interfaces for scheduling actions and authoring other action types, we constructed a GUI for giving the robot teams immediate high priority instructions. Through the interface depicted in Figure 4(b) a user chooses from drop-down menus the agent she is instructing, an action to be performed, and either an object participant or a location and object type. A PAR instance is then automatically created and added to the character's *Action Queue* with a very high priority. Having a high priority ensures that it will suspend or preempt any actions the robot is currently performing and start execution immediately.

9 Conclusions and Future Directions

In this paper, we have described the CAROSA framework that we have constructed to facilitate the creation of plausible, functional NPCs. Our characters do more than navigate from one location to another without colliding with each other or obstacles. Characters have roles that provide them purpose. Their behaviors are performed in proper context with the object participants. In fact, because actions reference PAR objects and not coordinates directly, the environmental setting can be completely rearranged without requiring additional work

(a) (b)

Fig. 4. Simulation of a robot, commander team tasked with clearing a building of weapons.

from the game designer. Objects can even be added or removed and the characters will still perform. Note, however, some actions may not be performed if there is a shortage of resources.

Through the use of the PAR representations, a resource manager, and definitions of roles and groups, we have a created a framework in which actions can be scheduled in forms analogous to the scheduling of real people. Game designers can then use these character behaviors to create more reasonable backdrops, motivate back-stories and drive storylines To add additional richness and more reasonable behaviors, we have also implemented reactive, opportunistic, and aleatoric actions that are equally easy to author and modify. These types of actions can be used to design flawed and emotional characters that provide additional depth and interest.

Additional development is still needed for the CAROSA framework. In particular, we would like to increase the scale of the populations. Our university simulation can run 30 characters in real-time on a standard PC. The most costly algorithm in the CAROSA framework is the calculation of opportunistic actions. Scheduling the fulfillment of a need can require searching through time for a gap in the character's schedule as well as through the environment to find a resource near a path the character will be traveling. We are considering caching these paths or locations or perhaps using stored way-points that act as road signs for resources.

Ideally, we would also like to include inverse kinematics for the characters. This would enable us to further showcase the PAR representation through more detailed object interactions. This would require additional semantic labeling of the objects in the environment (i.e. creation of sites), but objects can be reused in many simulations, so we believe the rewards would be worth the initial effort.

Finally, we would like to explore additional scenarios including outdoor environments. We are also working to build social and psychological characteristics into the characters. In particular, we are examining representing and simulating

personalities and culture. As these characteristics are developed we would like to simulate their impact on character interactions and team behaviors such as cooperation, coordination, and competition. In particular, we would like to look at character communication in both verbal and non-verbal forms.

Acknowledgements

Partial support for this effort is gratefully acknowledged from the U.S. Army SUBTLE MURI W911NF-07-1-0216. We also appreciate donations from Autodesk.

References

1. Bates, B.: Game Design. Course Technology PTR (2004)
2. Bindiganavale, R., Schuler, W., Allbeck, J., Badler, N., Joshi, A., Palmer, M.: Dynamically altering agent behaviors using natural language instructions. In: Autonomous Agents. pp. 293–300. AAAI (2000)
3. Brockington, M.: Level-of-detail ai for a large role-playing game. In: Rabin, S. (ed.) AI Game Programming Wisdom, pp. 419–425. Charles River Media, Inc., Hingham, MA (2002)
4. DePaiva, D.C., Vieira, R., Musse, S.R.: Ontology-based crowd simulation for normal life situations. In: Proceedings of Computer Graphics International. pp. 221–226. IEEE, Stony Brook, NY (2005)
5. GeniusConnect: http://www.geniusconnect.com/articles/products/2/3/ (Last visited May 2009)
6. Lee, K.H., Choi, M.G., Hong, Q., Lee, J.: Group behavior from video: A data-driven approach to crowd simulation. In: ACM SIGGRAPH / Eurographics Symposium on Computer Animation. pp. 109–118. San Diego (2007)
7. Lerner, A., Fitusi, E., Chrysanthou, Y., Cohen-Or, D.: Fitting behaviors to pedestrian simulations. In: Symposium on Computer Animation. pp. 199–208. ACM, New Orleans, LA (2009)
8. Massive Software Inc.: 3d animation system for crowd-related visual effects: http://www.massivesoftware.com (Last visited Aug 2010)
9. McDonnell, R., Micheal, L., Hernandez, B., Rudomin, I., O'Sullivan, C.: Eye-catching crowds: saliency based selective variation. In: In ACM SIGGRAPH. pp. 1–10. ACM, New Orleans, Louisiana (2009)
10. Pelechano, N., Allbeck, J., Badler, N.: Virtual Crowds: Methods, Simulation, and Control. Synthesis Lectures on Computer Graphics and Animation, Morgan and Claypool Publishers, San Rafael, CA (2008)
11. Sanchez-Ruiz, A., Llanso, D., Gomez-Martin, M., Gonzalez-Calero, P.: Authoring behaviour for characters in games reusing abstracted plan traces. In: Ruttkay, Z., Kipp, M., Nijholt, A., Vilhjalmsson, H. (eds.) Intelligent Virtual Agents. pp. 56–62. Springer (2009)
12. Sung, M., Gleicher, M., Chenney, S.: Scalable behaviors for crowd simulation. Computer Graphics Forum 23(3), 519–528 (2004)
13. Yu, Q., Terzopoulos, D.: A decision network framework for the behavioral animation of virtual humans. In: In Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 119–128. Eurographics Association, San Diego, California (2007)