Virtual Actors and Avatars in a Flexible User-Determined-Scenario Environment

Dan M. Shawver
Sandia National Laboratories
Albuquerque, NM 87185-0978
shawver@vris.sandia.gov

Abstract

VRaptor, a VR system for situational training that uses trainer-defined scenarios is described. The trainee is represented by an avatar; the rest of the virtual world is populated by virtual actors, which are under the control of trainer-defined scripts. The scripts allow reactive behaviors, but the trainer can control the overall scenario. This type of training system may be very useful in supplementing physical training.

1. Introduction

This paper presents VRaptor (VR assault planning, training, or rehersal), a VR system for situational training. VRaptor lets the trainer define and redefine scenarios during the training session. The trainee is represented by an avatar; the rest of the virtual world is populated by virtual actors, which are under the control of trainer-defined scripts. The scripts allow reactive behaviors, but the trainer can control the overall scenario.

VRaptor supports *situational training*, a type of training in which students learn to handle multiple situations or scenarios, through simulation in a VR environment. The appeal of such training systems is that the students can experience and develop effective responses for situations they would otherwise have no opportunity to practice. Security forces and emergency response forces are examples of professional groups that could benefit from this type of training. A hostage rescue scenario, an example of the type of training scenario we can support, has been developed for our current system and is described in Section 3.

Since control of behaviors presupposes an appropriate representation of behavior and means of structuring complex behaviors, we survey related work on behavior simulation in Section 2.

In the Virtual Reality / Intelligent Simulation (VR/IS) lab, our basic VR system [16] allows multiple human participants to appear in embodied form (as avatars) within a common, shared virtual environment. The virtual environment may also contain virtual actors. Using this infrastructure, we have developed the VRaptor system. VRaptor adds oversight and session control by a trainer, through a workstation interface. This interface, described in Section 4, allows selection of roles and actions for the individual virtual actors, and placement of them in the scene.

In Section 5 we present the architecture of the simulation component of VRaptor, and in Section 6 discuss the representation of scenarios in terms of scripts and tasks.

2. Related work

Since our focus in this research is on the scripting and control of virtual actors, we survey work toward building animations or behaviors which are either automated or reactive, and especially work which offers hope of allowing realtime implementations.

2.1. Behavioral animation

Behavioral animation has developed from the early work of Reynolds [15], on flocking and schooling behaviors of groups of simulated actors; recent work in this vein includes that of Tu and Terzopoulos [17]. Systems that deal with smaller groups, or individual behaviors, are reviewed in the following sections.

2.2. Ethologically-based approaches

Ethologically-based (or biologically-based) approaches deal with *action selection mechanisms*. Since

DISCLAIMER

intelligent behavior should emerge naturally in this approach, some form of *reactive planning* may be used. An approach that included reactive planning in a system providing simulation capabilities was developed by Maes [7], and subsequently extended into a distributed form in the work of Zeltzer and Johnson [18, 19]. Maes has demonstrated a system called ALIVE which provides simulated actors responding to users' gestures (see Maes *et al* [8]). Blumberg [3] describes a ethologically-based system which is embedded in the ALIVE framework.

2.3. Other approaches

Alternative approaches for simulation of reactive, situated actors have also been developed by Bates and Loyall [6], Becket and Badler [2], the Thalmanns and their group [11], and Booth et al [4]. The system of Bates and Loyall does not do any actual planning, although it does allow a range of actions to be reactively invoked, and supports the implementation of simulated simple actors that have an extensive repertoire of behaviors and include simulated emotional states. The system appears to make programming action sequences, as behavior segments, relatively straightforward. The system of Becket and Badler uses a network of elements (PaT Nets) to get reactivity. There is a higher-level, nonreactive planning component. The Thalmanns have explored some behavioral features in conjunction with synthetic actors, and they use a reactive selection of (fine-grain) strategies in association with synthetic vision in the cited work.

The work of Booth et al proposes a design for a *state machine engine*, which hierarchically combines state machines and constraint resolution mechanisms. This mechanism is described more fully in Ahmad et al [1].

In general, systems such as those developed by Zeltzer and Johnson, Bates and Loyall, and Becket and Badler assume an underlying stratum that deals with continuous, feedback-controlled domains, and provides a set of constituent actions (perhaps constituted from smaller primitive actions). The set of constituent actions are invoked by the reactive planning component. That is, these authors separate the creation of single, continuous actions from the selection and invocation of those actions. Nilsson [9, 10] combines both aspects of action in one formalism, called *teleo-reactive programs*. Multiple levels of more detailed specification are provided through procedural abstraction.

2.4. Individual behaviors and expressive movement

Recent work by Perlin [12, 13] has shown that to an interesting extent, relatively simple kinematic techniques can create movement that is both natural and expressive, the latter being made apparent through the example of a dancer figure animated by his techniques. More recent work by Perlin and Goldberg [14] has extended their work into multiple figures using a distributed system.

3. Testbed scenario

Hostage rescue, our testbed scenario, is the sort of operation an organization such as the FBI Hostage Rescue Team is called upon to perform. For a simple initial capability, we assume the rescue should take place in a single room. This type of operation is called a *room clearing*. Traditionally, training of response teams for such scenarios involves the use of a "shoothouse", a physical facility that models typical rooms and room arrangements, and is populated with manikins or paper cartoon drawings for the adversaries. Such facilities lack the flexibility and limit the degree of interesting interaction (the manikins may move only in simple ways, if at all). Our shoothouse scenario exhibits an alternative in which figures can move through a range of programmable actions. In addition, the physical facility is rather expensive to operate; our VR system should provide a more cost effective training option. (However, we do not foresee entirely replacing the physical shoothouse with a virtual one in the near future.)

3.1. Components of a room clearing operation

A room clearing operation proceeds in the following steps:

1. Breach through door(s) or wall to create an entry into the room.
2. Toss a *stun grenade* (or *flashbang*) into the middle of the room. This creates a diversion, and as the name implies, stuns the inhabitants of the room with blast and light.
3. Forces enter the room in pairs, each member of the pair to cover either the left or right side of the room from the breached opening. Each steps into the room along the wall and then forward. Thus each can clear his own section of the room.
4. Commands are given to the room occupants to "get down", and not resist.

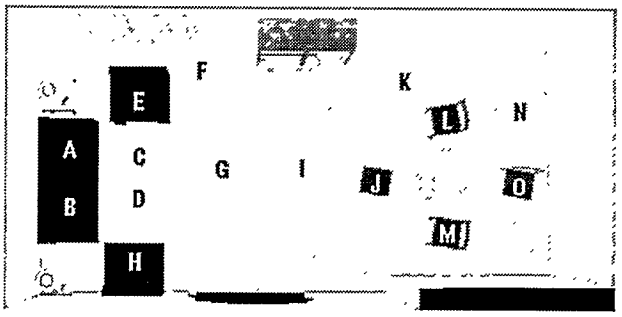


Figure 1. Allowed Virtual Actor Locations

5. Shoot armed adversaries.

The total attack time may be only a few seconds for a single room.

3.2. Training for a room clearing operation using VR

There will be one or more trainees who will be practicing the room clearing operation; these will be the intervention forces. The trainees will be using immersive VR.

The trainers will control the training session by setting up scenarios and monitoring the trainees' performance. The trainers will use a multiple-windows workstation display that provides a 3D graphics overview of the virtual environment (i.e. the room) and a user interface to define the scenario and start the session.

The room occupants will be simulated using virtual actors. These actors will carry out roles and actions assigned by the trainer, subject to reactive changes as the scenario proceeds, such as an actor getting shot.

4. VRaptor user interfaces

4.1. The trainer's interface

The user interface for the trainer consists of a 3D viewing window of the virtual environment and a set of menus. Using the menus, the trainer can control the placement of the actors in the room, assign them roles of either terrorist or hostage, and select scripts for each actor. The scripts are subject to constraints of applicability to the current position and pose of the figure. The menu choices adjust dynamically to reflect the current actor placements and scenario. Fig. 1 shows possible starting locations for the virtual actors. Views of the actors from within the room are shown in Figures 2 and 3. Typical menu choices for the actors' responses when the shooting starts are:

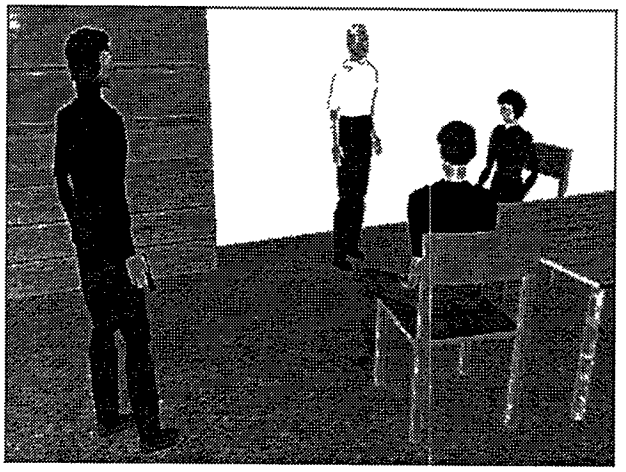


Figure 2. Virtual Actors in Room

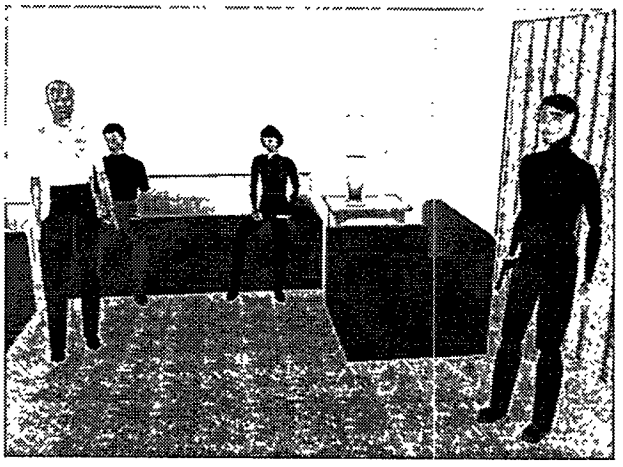


Figure 3. Another View of Actors

- give up and put hands in air, then on head
- dive for the floor and give up
- do nothing – i.e. dazed
- fight (if adversary)

Except where noted, the actor may be either a hostage or an adversary.

4.2. The trainee's interface

The trainee is immersed in the scene. The trainee is provided with a Head-Mounted Display (HMD)[1] and views the scene from the eye point of the appropriate avatar. The trainee holds a weapon which is currently a Baretta 9mm replica instrumented to detect trigger pulls and clip insertion or removal. This weapon provides the weight and feel of a real Baretta, but is lacking the recoil. The headmount and gun each have an electromagnetic tracker mounted on it, and in addition, electromagnetic trackers are mounted on the hand not holding the gun, as well as the lower back.

5. Virtual actor system

The virtual actor simulation is a distributed set of cooperating components. There are two types:

1. An actor/scenario controller component
2. A puppet server component

The simulation requires one actor/scenario component for the application, and one puppet/server component for each virtual actor. Basic supporting behaviors are installed in the lower-level ('puppet server') support modules. Higher-level behaviors appear as tasks dispatched on an actor-specific basis (see Sec. 6).

5.1. The actor/scenario controller

The actor/scenario controller manages all the actors and tracks the state of the simulated world. Higher-level behaviors are programmed as tasks in this component. These tasks are determined by a trainer using the menu system. Each actor is represented in the controller component by an object, which communicates to the appropriate puppet server for that actor. The controller sends commands to the puppet server, which carries out the command by animating the figure of the actor appropriately. Figure 4 illustrates this concept. This figure shows two actors, but in general

[1]We have been using the O1 Products PT-O1 HMD

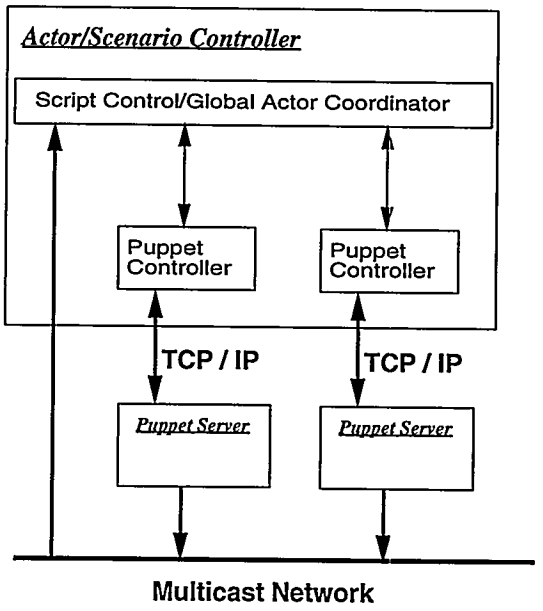


Figure 4. Virtual Actor Components

there can be many. The appropriate components (and processes) would be replicated for each actor. The actor/scenario controller implementation uses the Umbel Designer[2] environment. This environment allows an object-oriented design approach.

The actor/scenario controller contains a component which evaluates the gun position and orientation at trigger pull event time to determine which (if any) actors were hit. When an actor is hit, the actor/scenario controller overrides the current activity of that actor to force an appropriate response to the hit; e.g. the actor falls dead in a manner appropriate to its current position.

5.2. The puppet server

The puppet server component uses the NYU `kpl` language interpreter modified to provide I/O that is compatible with the VR/IS system (see Sec. 7.3). It runs `kpl` code rewritten to extend Ken Perlin's original "dancer" code [12, 13] with new behaviors and with techniques for building more elaborate behaviors through chaining simple behavior elements. Commands are sent from the actor/scenario controller by TCP/IP connections to the specific puppet server through an intermediate proxy for that puppet server (not shown in Fig. 4). This indirect route accomodates a lower-level menu interface to the individual puppet server for development of new basic behaviors. (Per-

[2]A product of Inflorescence, Inc.

lin's original interface creates tcl/tk menus; essentially the same kind of code interfaces with the proxy.)

6. Scripts and multitasking

Central to our research is provision of user-manipulatable scripting. To provide this, we use the *task* abstraction at the actor/scenario controller level. The mapping of script to tasks is one-to-many; multiple concurrent tasks may be required in general to realize all aspects of a particular script. For simple cases, one task may do.

There are also once-per-timestep condition checks taking place. These checks are a type of callback procedure registered with the simulation control mechanism of the actor/scenario controller. These check procedures can set variables, suspend or terminate a task, or signal a semaphore to wake up a task. An example of a task is given in figure 5.

6.1. Tasks and threads of control

We use Umbel Designer to provide a simulation-time task capability. Tasks have the ability to consume simulated time, while procedures are (conceptually at least) instantaneous. This task abstraction allows for both sequencing actions and pausing for either a specified delay time or until some condition is satisfied. One task can call another, which causes the calling task to wait for completion of the called task. In addition, tasks can be started so that they run asyncronously with the caller. Generally when a task terminates, at the end of its code block, the thread of control running that task terminates. In the case that the task was called from another task, the calling task resumes.

Tasks are implemented in terms of simulated time, but we constrain the simulated time to match real time. Obviously this can only be done if the real time required to do the tasks' computation is not too great. Thus runtime efficiency can be a major issue. This is somewhat alleviated in our architecture by having the division into large-grain high level control on the part of the actor/scenario controller and the fine-grain control on the part of the puppet servers. The latter run in parallel with the tasking computation.

6.2. Task dispatching

Tasks must be dispatched based on both the particular actor involved and his assigned script. In addition, overall scenario control may require one or more tasks to control scenario startup and monitor progress through the scenario. For an example, see Figure 5.

```
task terrorist_sitting_fight (a: actor);
    var i: integer;
begin
    { Assume have initially action_sit_relax }
    { flashbang has already occurred, so cringe: }
    choose_puppet_action( a.puppet,
                          action_cover_face_sit );
    delay( 1.5 {secs} );
    choose_puppet_target( a.puppet,
                          target_snl_human_1 );
    choose_puppet_attention_mode( a.puppet,
                                  attn_looking );
    delay( 0.25 {secs} );
    choose_puppet_action( a.puppet,
                          action_sit_shoot );
    while an_avatar_lives do
        for i := 1 to num_rounds_terrorist_has
            while an_avatar_lives do
                begin
                    delay( 0.5 {secs} );
                    actor_fires( a );
                end;
    choose_puppet_action( a.puppet,
                          action_sit_relax );
    delay( 0.45 {secs} );
    choose_puppet_attention_mode( a.puppet,
                                  attn_alone );
end;
```

Figure 5. Simple Task Example

The task `terrorist_sitting_fight` can be part of an actor's assigned script. It is called only after the main simulation task has caused the flashbang to occur. Hence the timing in this task is relative to that occurrence. (The procedure calls that refer to the actor's puppet send control messages to the puppet server for this actor.) Should the actor controlled by this task be shot, the task will be not be allowed to continue controlling the actor, and an appropriate dying action will be invoked from the puppet server for the actor.

7. VR environment modules

Our current VR environment combines different types of simulation modules with specialized display and sensor-input modules in a distributed architecture. The term *modules* here means separate executables, with each typically running as a single Unix process, but frequently with multiple threads of control. The module types include the following:

1. The VR Station display
2. Polhemus tracker input module.

3. An avatar driver

4. Virtual actor modules as described in Sec 5.

The first three types of modules above will be described in more detail in the following sections. The VR environment consists of multiple instances of these types of modules.

7.1. The VR Station

The VR Station is the display driver module for the user. It provides an immersive view of the world, with remotely-driven real-time updates of the positions and orientations of objects and subobjects in the world. Typically, there are multiple instances of the VR Station running on separate CPUs, each with its own graphics pipeline hardware (typically an SGI Crimson with Reality Engine, or Onyx with Reality Engine 2). A VR Station instance is used by a participant in the scene (with an avatar), who in our testbed system would be a member of the intervention forces. VR Stations can also be used by observers who have no visible representation in the simulated world (*stealth observers*). The trainer's view is of this type.

7.2. The avatar driver and tracker input

The avatar driver is based on that described in Hightower [5], modified to accomodate placement of the right hand tracker on the gun held by the trainee. This placement of the tracker maximizes accuracy in evaluation of the aim of the weapon. There are also trackers on the left hand, the small of the back, and the head. An auxiliary module acquires the tracker data and sends it to both the avatar driver and the VR Station instance that supplies the HMD view for the participant. There is an avatar driver instance and a tracker input module instance for each trainee.

7.3. Communication from avatar and actors to the VR Station

All of the VR Station instances "see" the same world, although each VR Station can show a different view of it. Thus, the communication from the figure drivers (avatar driver and the puppet server modules) to the VR Station must allow this sharing. This requirement is met in the current Ethernet implementation using multicasting of UDP datagrams.

Each VR Station instance independently loads data files that describe the world and the figures in it. Each figure driver (avatar or actor) loads a corresponding file that describes the part of the world that it controls. The major output data from the figure drivers is transforms for the figure's joints and placement in the world. Thus figure drivers can move the figures that they control simultaneously in all views.

8. Summary and future work

This paper has presented VRaptor, a VR system for situational training, that lets the trainer define and redefine scenarios during the training session. Trainees are represented by avatars; the rest of the virtual world is populated by virtual actors, which are under the control of trainer-defined scripts. The scripts allow reactive behaviors, but the trainer can control the overall scenario.

Initial feedback from potential users is promising. Future work includes adding features and improving the trainer's control. We want to extend the trainer's interface to allow selection and juxtaposition of more basic behavior elements through icons, which would extend the trainer's control of scripts to a finer-grained form. For deployment in actual training, monitoring and logging the trainee's performance would be necessary. This would allow performance review with or without the trainee present, and allow the trainer to evaluate scenarios with respect to difficulty or need for improvement. Also, the system could be used in planning an assault, and this monitoring capability would then be one way of accessing competing plans of attack. We hope to eventually evaluate the VRaptor system for training effectiveness.

9. Acknowledgements

Jim McGee was one of our domain experts, and helped us define the application. The HTML menuing system was implemented in Umbel Designer by Denise Carlson. The gun interface and sound effects support was implemented by James Singer. The avatar support is by Ron Hightower, using a figure imported from the Jack(TM) system. Ron also created some of the low-level behaviors, and worked on improving the figure models. The multicast packet and network support libraries were implemented by Nadine Miner and Jim Pinkerton respectively. The room and virtual actor figure models were done by Monica Prasad. We are grateful to Ken Perlin for supplying his dancer code. We would also like to thank Wade Ishimoto for his help. The VR/IS lab team is led by Sharon Stansfield. This work was sponsored by Sandia's LDRD program and was carried out at Sandia National Labs, supported by DOE under contract DE-AC04-94AL85000.

References

[1] O. Ahmad, J. Cremer, J. Kearney, P. Willemsen, and S. Hansen. Hierarchical, concurrent state machines for behavior modeling and scenario control. In *Proceedings of 1994 conference on AI, Simulation, and Planning in High Autonomy Systems*, Gainesville, FL, December 1994.

[2] W. Becket and N. I. Badler. Integrated behavioral agent architecture. In *Recent Techniques in Human Modeling, Animation, and Rendering (SIGGRAPH 93 Course Notes 80)*, pages 4–39–4–50. 1993.

[3] B. M. Blumberg and T. A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 47–54. ACM SIGGRAPH, Addison Wesley, Aug. 1995. held in Los Angeles, California, 06-11 August 1995.

[4] M. Booth, J. Cremer, and J. Kearney. Scenario control for real-time driving simulation. In *Proceedings of the 4th Eurographics Workshop on Animation and Simulation*, pages 103–119, September 1993.

[5] R. Hightower. Active embodiment of participants in virtual environments: Sensor-driven avatars. In *Proceedings of the 1996 IMAGE VIII Conference*, 1996.

[6] A. B. Loyall and J. Bates. Real-time control of animated broad agents. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, pages 664–669, June 1993.

[7] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*. The MIT Press, 1990.

[8] P. Maes, D. Trevor, B. Blumberg, and A. Pentland. The ALIVE system full-body interaction with autonomous agents. In *Computer Animation '95*, Apr. 1995.

[9] N. J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, January 1994.

[10] N. J. Nilsson. Reacting, planning and learning in an autonomous agent. In K.Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence 14*. The Clarendon Press, 1995.

[11] H. Noser, O. Renault, D. Thalmann, and N. Magnenat-Thalmann. Vision-based navigation for synthetic actors. In *Recent Techniques in Human Modeling, Animation, and Rendering (SIGGRAPH 93 Course Notes 80)*, pages 1–31–1–36. 1993.

[12] K. Perlin. A gesture synthesizer. In J. F. Blinn, editor, *SIGGRAPH 94 Course Notes, Course 1*. 1994.

[13] K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.

[14] K. Perlin and A. Goldberg. Improv: A system for scripting interactive actors in virtual worlds. In *SIGGRAPH 96 Proceedings*, pages 205–216, 1996.

[15] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 25–34, July 1987.

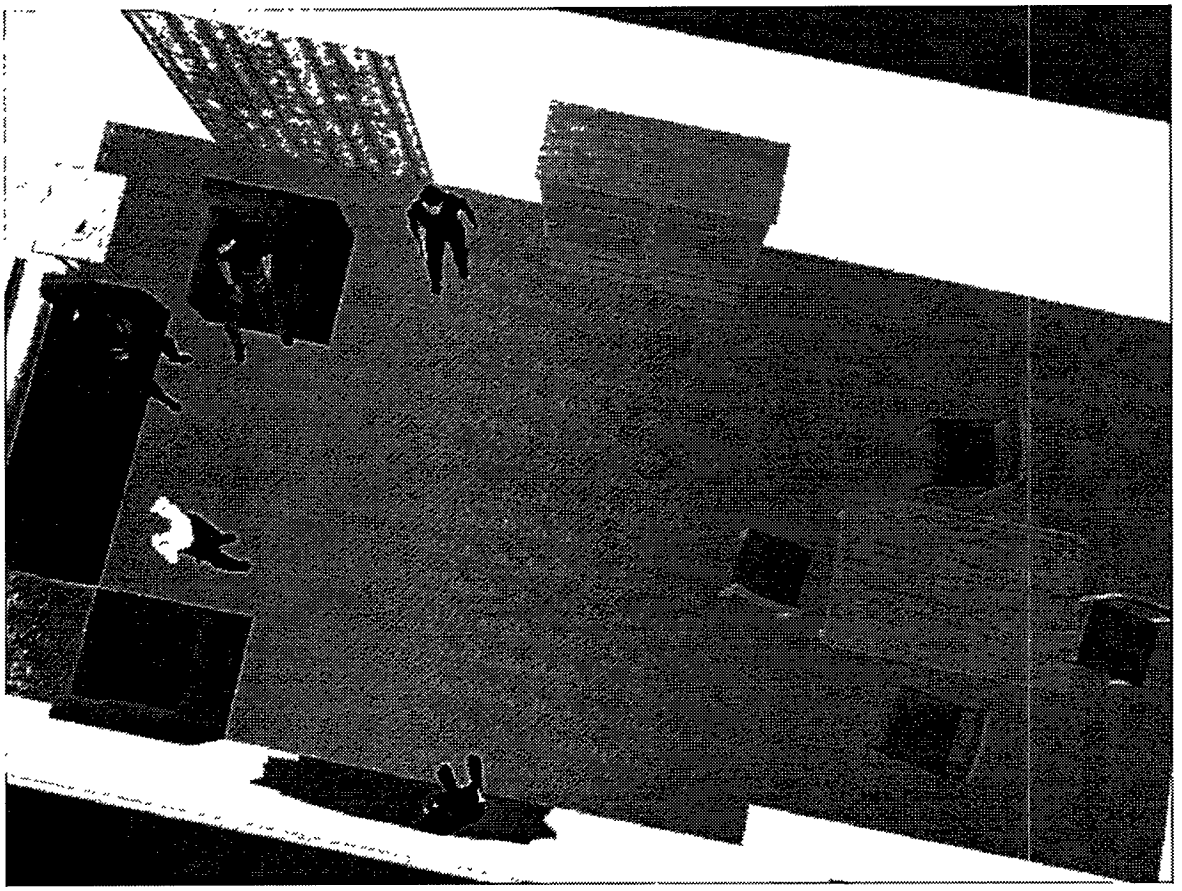
[16] S. Stansfield, N. Miner, D. Shawver, and D. Rogers. An application of shared virtual reality to situational training. In *VRAIS '95 Proceedings*, pages 156–161, 1995.

[17] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *SIGGRAPH '94 Proceedings*, pages 43–50, July 1994.

[18] D. Zeltzer and M. B. Johnson. Motor planning: An architecture for specifying and controlling the behaviour of virtual actors. *Journal of Visualization and Computer Animation*, 2:74–80, 1991.

[19] D. Zeltzer and M. B. Johnson. Virtual actors and virtual environments. In L. MacDonald and J. Vince, editors, *Interacting with Virtual Environments*, pages 229–255. John Wiley & Sons, Ltd., 1994.

To be submitted for inclusion as color plate in VRAIS '97 Proceedings:



Caption: Trainer's view of the shoothouse