

Planning for Animation

Norman I. Badler Bonnie L. Webber Welton Becket
Christopher Geib Michael Moore Catherine Pelachaud
Barry D. Reich Matthew Stone

1 Introduction

If there is any one truth of computer animation that has emerged from numerous efforts in recent years, it is that *simple procedural techniques cannot capture the complexity of human motion, interaction, and communication*. As one technique improves the veracity of one aspect of a movement simulation, other aspects become distorted or suppressed, and new problems appear. This is because human movement is the product of so many factors: physics, capabilities, emotion, experience, context, and interactions with other physical objects and agents. We should expect the world to require highly interconnected processes to improve simulation quality while retaining acceptable levels of specification burden and computational resources: it is rather unlikely that a complete behavioral simulation at the sensory-motor level would be feasible. Nor should we expect it to be provided by an expert animator. Indeed, the latter has spawned the quest for more automated animation techniques [Badler et al. 1991, Badler et al. 1993b, Cohen, 1992, Liu, 1994, Ngo and Marks, 1993, van de Panne and Fiume, 1993, Sims, 1994, Tu and Terzopoulos, 1994, Witkin and Kass, 1988]. While such techniques can potentially change the field in major ways, their modest results to date have led the community to turn increasingly to “performance based” animation – *i.e.*, directly sensed motion of actors – as a viable method to produce realistic-quality human motion at low (programming or expert skill) cost [Badler et al. 1993a, Robertson, 1994, Williams, 1990].

“Performance based” animation, however, does not solve the problem of generating realistic and appropriate human movements in any of the following situations:

- Highly-variable situations, where one cannot *a priori* acquire the range of movements that people will make. Such situations can arise in real-time simulation activities, where the situation and actions of other agents cannot be predicted ahead of time.
- Dangerous situations, where one cannot involve a human actor.

- Time-critical situations, where the amount of time needed to program behaviors excludes programming as an option.

Such situations require immediate, situationally appropriate (reactive) behavior on the part of a synthetic agent.

Under such conditions, a synthetic agent must be made to sense its environment and respond to it in terms of reflexes, higher level intentions, expectations, and available skills. In this paper, we argue that an effective way of doing this is through the integration of a rich collection of interacting techniques, organized in a principled, structured representation. These techniques include planners and parallel transition networks (Section 3.2) to aid in overall task control, and goal-based sensing, response, and (as necessary) physics-based, kinematic or inverse kinematic behaviors to achieve environmentally-appropriate movements. Together they simplify:

- the expression of local environmental influences without complicating their expression at the higher levels and
- the expression of situational awareness and influences at a higher level without the added complexity of managing all potential lower level variability,

while admitting:

- the aggregation of the intentions and expectations associated with individual tasks;
- the interaction of multiple agents, wherein agents can sense and react to the behavior and perceived intentions of other agents, as well as to the environment;
- the distinction between expected alternatives in the way a situation may develop and seizure of opportunity or total surprise at circumstances;
- the independent but coordinated control of a human agent's locomotion, upper body motions, visual attention, and communication systems (speech and facial expression);
- the embedding of persistent data structures for cognitive knowledge or spatial maps as needed to remember or mark as known visible parts or characteristics of the environment;

- the emergence of human-like qualities as a consequence of executing the agent’s representational models.

After setting out a framework of definitions and describing the representational structures used in our approach, we will illustrate how they are embodied in four different systems. Not only have these systems evolved in parallel with the approach we advocate, but they embody it to an ever-increasing degree: “Stepper” (Section 4), “Hide and Seek” (Section 5), and “Gesture Jack” (Section 6).

2 Definitions

There are essentially several interrelated concepts – that of **agent**, **activity**, **goal**, **role** and **context** – that underlie our approach to automating the animation of environmentally-appropriate and task-appropriate agent behavior. Understanding these concepts should set the stage for understanding “Hide and Seek” and “Gesture Jack”.

2.1 Agents

An **agent** is an object that is able to take action by virtue of having a sense-control-action (SCA) loop to produce locally-adaptive behavior (Section 3.1). An agent with only SCA loops determining its behavior is considered a purely “reactive” agent. Deliberative agents (such as human agents) also have higher-level control structures that affect more global, planned, or cognitive aspects of behavior, which in turn can affect the immediate formulation and parameters of an SCA loop (Section 3.2). (An object which causes events through purely physical forces is not considered an agent.) Because our interest is in behavioral realism rather than visual realism, we focus on human agents, whose behaviors we have the greatest familiarity with and expectations about, thus providing a grounding for judging realistic behavior. The definition, however, allows for other agents acting under non-human drives and motivations. Agents can, of course, sense and respond to the presence and actions of other agents.

2.2 Activities, Goals, and Behaviors

Simply stated, **activities** are anything one would describe an agent as doing. What is more important to the automated production of animated simulations is how activities are specified. In a Natural Language (NL) like English, activities are specified as verbs (“walk”), verb phrases (“walk to the store”) and even whole paragraphs (*e.g.*, “Walk to the store. Make sure you take you raincoat, and avoid taking Main Street because there’s a parade today.”). In our approach, activities are specified through the Parallel Transition Networks (PaT-Nets) described in Section 3.2. The important things to note about activity specifications in both NL and PaT-Nets is that:

- activities can be specified at many levels of abstraction, from *look for a hider* to *move FINGER*;
- activities can be specified either individually or as structures of more basic activities;
- activities are essentially *processes* that are terminated when one or another condition holds. Activity specifications can embody termination conditions intrinsically (*e.g.*, “play the Moonlight Sonata” is intrinsically terminated when the player reaches the end of the sonata), or the termination conditions can be specified explicitly (*e.g.*, “play poker for an hour or until you lose your shirt”).

Crucial to the successful animation of human agents is the notion that activities have executable implementations. Appropriate structures for activities will assist construction of their computational analogues.

GOALS What an agent tries to achieve through its activities. Like activities, goals can be specified at different levels of abstraction – *e.g.*, from learning to bluff at poker to getting to a particular location.

TASKS Usually a combination of one or more goals and, optionally, the activities that contribute to achieving them. At a high-level, instructions can specify the task(s) an agent should carry out or goal(s) it should achieve. At a low-level, instructions can specify activities.

SCHEMAS The “program” an agent follows while executing tasks. Parallel transition networks (PaT-Nets) are the schema “programming language.” The task(s) may be produced by an explicit scripted procedure written as a PaT-Net schema, or the schema may itself be generated or modified by the actions of a planner or other decision-making system. The “outputs” of the schema are local variable or memory changes at the PaT-Net level, a set of behavioral instantiations to be executed at the SCA level, and any necessary communication links between the two.

BEHAVIORS In general, behaviors are considered “low level” capabilities of an agent, such as locomoting [to], reaching [for], looking [at], speaking [to], listening [to], *etc.*

2.3 Roles

An agent can adopt **roles**, each of which results in a set of tasks (a “working set”) becoming relevant. The tasks associated with a role needn’t be instantiated immediately upon its adoption: for example, in the context Hide-and-Seek, the role of *hider* is associated with a set of tasks including hiding, watching for the *seeker*, stealing home, and running from the seeker. The activities a hider is actually engaged in depends on the phase of the game and the seeker’s (perceived) activity: s/he hides during the counting phase of the game, s/he watches for the seeker during the seeking phase of the game (and may also steal home during that phase), and s/he runs from the seeker during the pursue-and-tag phase of the game.

The working set concept, though primarily designed for computational efficiency, actually leads to the reasonable notion of deliberate “context-switching” when external events or internal desires force adoption of new roles. Thus if the dinner call is heard, the hider can drop one set of activities for another without waiting for current activities to be completed. The agent’s organization into schemas and an SCA loop means that the abandoned activity’s behavior set will be replaced by the “go to one’s house” activity’s behavior set: the transition will be seamless and natural. The agent will not continue to look for hiding places on the way home.

There are many circumstances in which multiple tasks may be active, and the agent must adopt behaviors which are selections or combinations or those possible. For example,

it is not difficult to execute both locomotion and arm motions simultaneously, but the agent needs to decide whether to grasp for a new object while engaged in holding another. Some of these decisions can be made at the schema level (essentially by allocating and locking resources), while others are managed at the SCA level (such as walking and talking at the same time).

2.4 Contexts

A **context** may be the result of natural or other outside forces or the result of multiple agents agreeing that the context will hold until some later point. In the latter case, each agent initially adopts a mutually-accepted role in the context (*e.g.*, *speaker* and *listeners* in the context of a conversation, *seeker* and *hidiers* in the context of hide-and-seek, *etc.*), as well as accepting the rules and/or conventions for exchanging roles within that context and any mutually-accepted conditions under which the context will be dropped.

Contexts are not necessarily mutually-exclusive, so an agent may easily be in several contexts and therefore in several roles simultaneously. The activities the agent is engaged in at any one time may come from any of these activities, which may or may not be affected by the other contexts simultaneously at play. For example, two agents, both in the role of *hidiers* in a game of hide-and-seek, may decide to engage in conversation, thereby adopting additional roles of *speaker* and *hearer*. Because, as hidiers, they are trying to avoid detection, their conversation may be whispered or carried out through gesture or sign language. As another example, an agent in the role of *hider* in the context of hide-and-seek is also usually in the context of self-preservation. Therefore, s/he will avoid hitting obstacles as s/he moves to a hiding place, tries to steal home or runs away from the seeker.

3 An Intelligent Agent Architecture

An intelligent agent must interleave sensing, planning, decision-making, and acting. Accordingly, it is desirable to create an architecture that permits specification and exploration of each of these processes. Planning and decision-making can be accommodated through incremental, symbolic-level reasoning. When the agent decides to act, the symbolic actions must be instantiated in executable behaviors. Most behavioral systems use

either state controllers or numerical feedback streams, but not both. By using both it is possible to obtain maximum flexibility and maintain appropriate levels of specification [Badler et al. 1993b, Becket and Badler, 1993].

We can characterize these two control levels as PaT-Nets and SCA loops.

- PaT-Nets are parallel state-machines that are easy for humans and automatic planning systems to manipulate. They are also good at sequencing actions based on the current state of the environment or of the system itself. They characterize the tasks in progress, conditions to be monitored, resources used, and any temporal synchronization.
- The SCA loop performs low-level, highly reactive control involving sensor feedback and motor control.

In this paradigm, the agent can instantiate explicit PaT-Nets to accomplish certain goals (*e.g.*, go to the supply depot and pick up a new motor), while low-level control can be mediated through direct sensing and action couplings in the SCA loop (*e.g.*, controlling where the agent’s feet step and making sure that s/he doesn’t run into or trip over any obstacles). Since the sensors can establish what the agent can perceive, the agent is able to react through the SCA loop, and if desired, use this information to confirm, adopt, or select higher-level (cognitive) actions: for example, if an obstacle cannot be surmounted, the current PaT-Net might need to be reconsidered. Since PaT-Net state transitions are explicitly represented, statistical distributions of alternative behaviors may be easily embedded.

PaT-Nets may instantiate themselves when certain events or conditions occur [Firby, 1992, Georgeff and Lansky, 1990], or they may be invoked or invented by high-level planning processes [Weber et al. 1995]. They depend on the required action or mission, the agent’s immediate intentions, and overall behavioral doctrine. For example, a PaT-Net to traverse a partly wooded area to achieve a target location safely may be based on a “locomote to target via suggested route” plan, but mediated through an SCA loop that uses sensor readings on the terrain type (don’t walk through trees or into holes), and global policies (try not to place yourself in the view of any known hostile agents). With each step, the agent’s action is affected by all these conditions.

The rest of this discussion is organized into a more comprehensive description of SCA loops and PaT-Nets. We will then have enough tools in place to illustrate the interactions between planning, PaT-Nets, and SCA behaviors in three domains: “Stepper,” “Hide and Seek,” and “Gesture Jack.”

3.1 Low-Level Control: Sense-Control-Action Loops

The SCA or *behavioral loop* is a continuous stream of floating point numbers from the simulated environment, through simulated sensors providing the abstract results of perception, through control decisions independently attempting to solve a minimization problem, out to simulated effectors or motor actions (walking, *e.g.*), which enact changes on the agent or the world. This loop continuously operates, connecting sensors to effectors through a network of nodes which for descriptive convenience are divided into *sense* (S), *control* (C), and *action* (A) phases.

The behavioral loop is modeled as a network of interacting SCA processes connected by arcs across which only floating point messages travel. An individual, conceptual path from sensors to effectors is referred to as a *behavioral net*. It is analogous to a complete behavior in an “emergent behavior” architecture such as Brooks’ *subsumption architecture* [Brooks, 1986], except that nodes may be shared between behaviors, and arbitration (competition for effector resources) may occur throughout the behavioral path and not just at the end-effector level. The behavioral loop is modeled as a network with floating point connections in order to allow the application of low-level, unsupervised, reinforcement learning in the behavioral design process. (This is being developed in [Becket, 1995]). Since our main use of SCA loops to date has been in locomotion reasoning, the remaining discussion will be in these terms.

3.1.1 Sensory Nodes

Sensory nodes model or approximate the abstract, geometric results of object perception. They continuously generate signals describing the polar coordinate position (relative to the agent) of a particular object or of all objects of a certain type within a specified distance and field of view. A few of the sensors used are:

object sensors: These provide the current distance from the agent and angle relative to the forward axis of the agent of a particular object in the environment. While we find it sufficient for a sensor to abstract over object recognition, a more sophisticated approach would simulate the high-level results of vision of the agent using Z-buffering hardware to create a depth map of what the agent can see [Renault et al. 1990, Reynolds, 1988].

range sensors: A range sensor collects all objects of a certain type within a given range and field of view, and performs a weighted average into signals giving the distance and angle of a single abstract object representing all detected objects. Signals into the sensor define the range, field of view, and weighting parameters (defining relative weights of distance and angle) and may be altered continuously in order to focus the sensor.

terrain mapper: This sensor perceives an internal map of the terrain as if it were an external entity. This is discussed in detail in Section 4.

human sensor: If an object is a human, that information is detected by this sensor.

Other sensors can be developed and embedded in the architecture as needs arise.

3.1.2 Control Nodes

For locomotion reasoning we use two simple control nodes loosely based on Braitenberg's *love* and *hate* behaviors [Braitenberg, 1984], but formulated as explicit minimizations using outputs to drive inputs to a desired value (similar to Wilhelms' [Wilhelms and Skinner, 1990] use of Braitenberg's behaviors). Control nodes typically receive input signals directly from sensory nodes, and send outputs directly to action nodes, though they could be used in more abstract control situations. Our two control behaviors are:

attract: Create an output signal in the direction of the input signal, but magnified according to distance and angle scalar multipliers and exponents. This node works only when input signals exceed a threshold distance or angle.

avoid: Create an output signal in the opposite direction of the input, magnified according to scalar multipliers and exponents, whenever inputs fall below a threshold distance or angle.

These nodes incorporate both scalar multipliers and exponents, to allow modeling the nonlinearities typically observed in animal responses to perceived inputs [Reynolds, 1987].

3.1.3 Action Nodes

Action nodes connect to the underlying human body model and directly execute routines defined on the model (such as walking, balance, hand position, and torso orientation) and arbitrate among inputs, either by selecting one set of incoming signals or averaging all incoming signals. An example is the *walk controller*, which decides where to place the agent's next footstep and then connects to the locomotion generator [Badler et al. 1993b] to achieve the step.

3.2 High-Level Control: PaT-Net Schemas

Low-level control is designed to connect to a general symbolic reasoning process, including a model of parallel automata called *Parallel Transition Nets* (PaT-Nets) [Becket, 1994] and various planners. A sample PaT-Net is shown conceptually in Fig. 1. Each net description is a class in the object-oriented sense and contains a number of nodes connected by arcs. Nodes contain arbitrary Lisp expressions to execute as an *action* whenever the node is entered. A transition is made to a new node by selecting the first arc with a true condition (defined as a Lisp expression). Nodes may also support probabilistic transitions where the probability of a transition along an arc is defined rather than a condition. *Monitors* are supported that, regardless of which state the net is in, will execute an action if a general condition evaluates to `true`.

A running network is created by making an instance of the PaT-Net class. Because a running net is actually an encapsulated, persistent object, it may have local state variables available to all actions and conditions, and may also take parameters on instantiation. The running PaT-Net instances are embedded in a Lisp operating system that time-slices them into the overall simulation. This operating system allows PaT-Nets to spawn new nets, kill

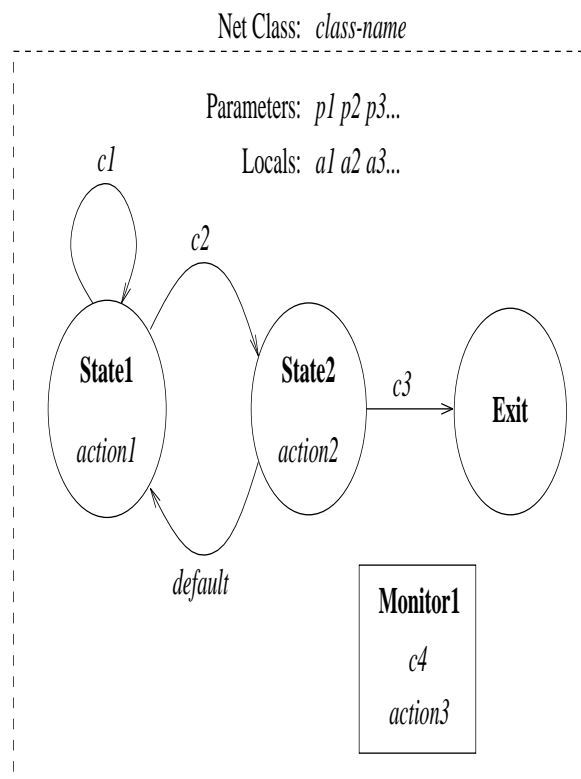


Figure 1: A sample PaT-Net shown graphically

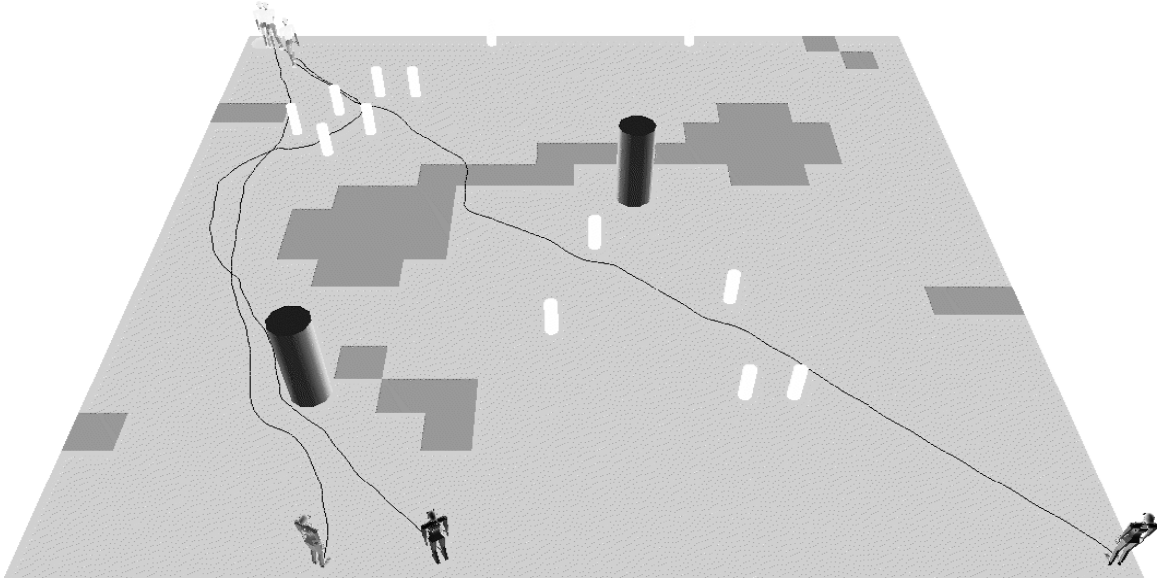


Figure 2: Attraction, avoidance, and terrain awareness

other running nets, communicate through semaphores and priority queues and wait (sleep) until a condition is met (such as waiting for another net to exit, for specific time in the simulation, or for a resource to be free). Running nets can, for example, spawn new nets and then wait for them to exit (effectively a subroutine call), or run in parallel with the new net, communicating if necessary through semaphores. Because PaT-Nets are embedded in an object-oriented structure, new nets can be defined that override, blend, or extend the functionality of existing nets. Several different PaT-Nets will be illustrated in the remaining sections.

4 Stepper

Stepper is an instance of the two-level (PaT-Net and SCA loop) architecture providing locomotion reasoning and control for simulated human agents in simulated environments [Reich et al. 1994]. Locomotion reasoning determines the characteristics of an agent's locomotion: *i.e.* what types of attractions, avoidances, and posture changes will achieve the goal (Fig. 2).

At the low level, *Stepper* uses an SCA loop to generate human locomotion. A set of influences (combinations of simulated sensors and attraction or avoidance) determine an

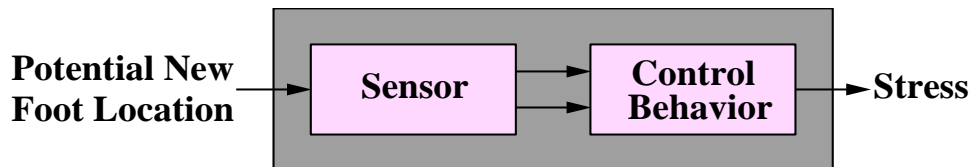


Figure 3: An Influence

agent’s behavior. At the high level, a set of PaT-Nets schedule and control these influences.

4.1 The Sense-Control-Action Loop for Human Locomotion

Stepper makes use of *Jack*’s framework for general object locomotion, which in turn makes use of an SCA loop that performs “anticipatory sensing”. That is, in the sense phase, sensors “anticipate” the environment at each potential next foot position, in order to determine in the control phase where the agent should step. The agent takes the chosen step in the action phase. A human steps at discrete positions in a continuous space, and cannot change the targeted step location while a step is in progress. The advantage of anticipatory sensing over some kind of averaging is described in Section 4.3.

Individual *influences* bound to the agent effect anticipatory sensing. An influence captures both what aspects of the environment are relevant to monitor and the degree to which they should attract or repel the agent from a particular position. An agent can be influenced by natural features of the terrain (*e.g.*, muddy ground, bodies of water), man-made features of the environment (*e.g.*, sidewalks), locations for which it is headed, *etc.*

An influence determines how an agent acts. In our system, the combination of a sensor and a control behavior (attraction or avoidance) is an influence (Fig. 3). An influence maps a foot position to the *stress* of stepping there. From among the possible choices, the control phase of the SCA loop leads the agent to take the least stressful step.

An influence activates when “bound” to an agent. While active, its output contributes to the stress calculations. Influences may be bound or unbound at any time during a simulation, and hence activated or deactivated. Locomotion is performed by binding influences to humans. The SCA loop, constantly monitoring the environment, immediately initiates the appropriate locomotion.

4.2 Architectural Overview

Each iteration of the SCA loop includes sensing, control, and action. Three phases compose sensing: (1) generation of possible steps, (2) polling of active influences, and (3) calculation of overall stress. The first phase determines a set of points to which the agent can next step. Then, in the second phase, polling of active influences determines stress values for each point; stress is proportional to an agent’s desire not to step at a given point. Finally, the sum of the stress values for each point determines the overall stress for that point, given the active influences.

The control step selects the least stressful potential point and directs the action step to execute the locomotion.

Generation of New Foot Positions The first part of sensing generates candidate foot positions. Three parameters determine the set of possible positions:

- Minimum Step Length
- Maximum Step Length
- Maximum Turning Angle (θ)

Given the agent’s position and orientation, these parameters define a fan-shaped region in space (Fig. 4). The agent can step anywhere within this region. The set of states, \mathcal{P} , that the agent can be in next corresponds to the set of points to which the agent can step next. The set of points is the intersections of concentric arcs and uniformly spaced radii. The user determines the number of arcs and radii. The user may also add the agent’s current location to set of points being considered. If this point is selected as the point to which to step, then the agent doesn’t move. Care must be taken when choosing this option to avoid situations where the agent may get stuck in a local minimum.

Polling Influences For each state $p \in \mathcal{P}$, the system simulates p and then polls the influences. Each influence returns a stress value. The higher the stress value, the less desirable the state. The stress value is multiplied by a user-defined scalar and raised to

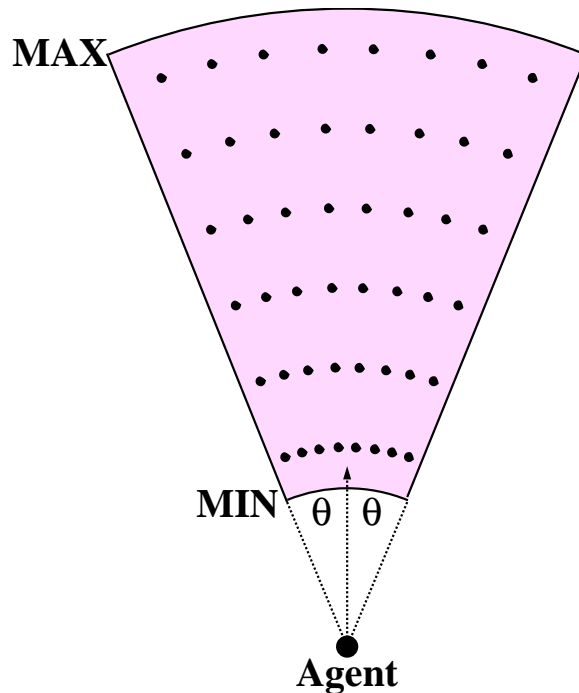


Figure 4: Calculating the next foot position

a user-defined power. This gives the user control over the relative contributions of the influences.

Influences have access to the entire environmental database and are effectively omniscient. The designer of the influence is responsible for limiting the amount or type of knowledge extracted.

Sometimes an influence computes two stress values: one based on the agent's position and one based on its orientation. In this case, two sets of user-defined constants are needed.

Calculation of Overall Stress The overall stress of a particular state is the sum of the weighted stresses output by each active influence. An influence is active if bound to the agent; influences may be bound and unbound at any time during a simulation.

Stepping Once each state's overall stress has been computed, the system selects the state with the lowest overall stress. Invocation of the locomotion system causes the agent to step to the appropriate point. When the step is completed, the agent enters a new state and the

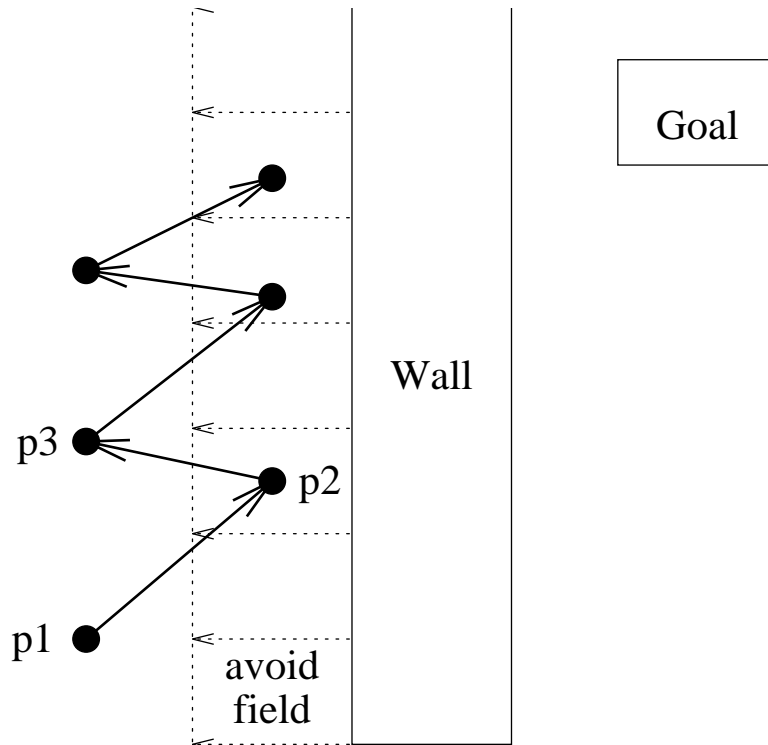


Figure 5: Sawtooth path due to potential field discontinuities

SCA loop continues.

4.3 Advantages of Anticipatory Sensing

In calculating the stress values of all possible next foot positions, the system looks one step into the future. As an alternative to this anticipatory sensing approach, a velocity vector resulting from the blended output of all active influences could be used to determine the next footstep. However, this would result in severe instability around threshold boundaries. This occurs because we allow thresholds in our sensor and control behaviors and as a result the potential field space is discontinuous. Taking a discrete step based on instantaneous information may step across a discontinuity in field space. Consider the situation in Fig. 5 where a goal on the opposite side of the wall attracts the agent, who avoids the wall up to some threshold distance. If the first step is scheduled at position $p1$, the agent will choose to step directly toward the goal and will end up at $p2$. Now within the threshold distance for

walls, the agent will step away from the wall and end up at $p3$, outside the threshold. This process then repeats until the agent clears the wall, producing an extremely unrealistic sawtooth path about the true gradient in the potential field. Anticipatory sensing, by considering and choosing among all possible next steps, avoids this problem.

Currently, anticipatory sensing only looks one step into the future and calculates the resulting stress, based on the present environment. One extension under development moves beyond this simplification by projecting moving objects (including humans) into the future. Instead of avoiding an object based on its present location, an object will be avoided based on where it is likely to be after the current step. Probabilistic methods will be used to determine the likely location of the object.

A second extension under development projects the agent and objects several steps into the future. The contribution of a step to the stress calculations will be greatest for the first step and decrease for each additional step. This should produce paths that better reflect humans' patterns of locomotion.

4.4 PaT-Nets for Human Locomotion

PaT-Nets introduce decision-making into the agent architecture. They monitor the SCA loop (which may be thought of as modeling instinctive or reflexive behavior) and make decisions in special circumstances. For example, the observed behavior resulting from the combined use of different influences can sometimes break down. The agent may get caught in a dead-end or other local minimum. Actions sometimes fail and unexpected events sometimes occur. PaT-Nets can recognize these situations, modify the agent's behavior by binding and unbinding influences, and then return to a monitoring state. During a simulation, PaT-Nets bind and unbind influences in Stepper, thereby altering agent behavior.

Consider an example with Tom chasing Jerry. The ChaseNet shown in Fig. 6 begins in *state 1*. An attraction to Jerry binds to Tom. As Tom begins to run toward Jerry the net passes to *state 2*; the ChaseNet enters the monitoring state. When Jerry ceases to be visible to Tom (Jerry may have run around a corner or behind an object), the net enters *state 3*. An attraction to the location where Jerry is most likely to be found, generally Jerry's last known location, binds to Tom. Tom begins to run toward this location as the ChaseNet transitions to *state 4*. If Tom arrives at this location and does not see Jerry, the ChaseNet

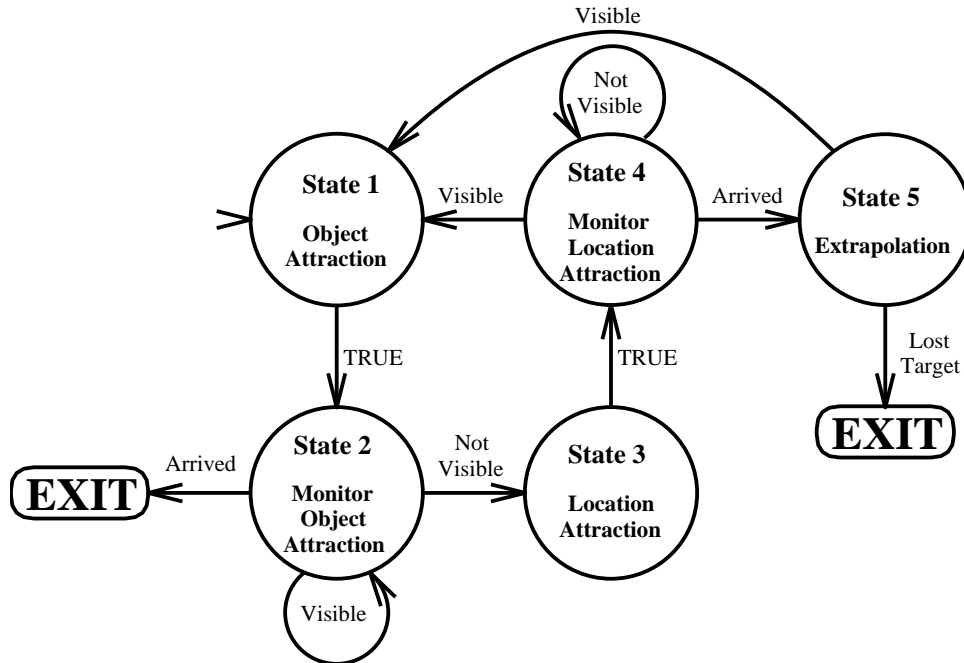


Figure 6: ChaseNet State Diagram

transitions to *state 5* and Tom searches in the direction Jerry was last known to be heading.

Clearly, chasing requires reasoning and decision-making beyond the scope of the SCA loop alone. PaT-Nets provide this reasoning, and schedule and control the low-level influences to direct the agent to act in the desired manner.

5 Hide and Seek

In [Moore et al. 1995] we describe a planning system for players in a game of “Hide and Seek” and its vertical integration into a system called ZAROFF that selects reactive behaviors to execute in an animated simulation. By interleaving planning and acting, the players dynamically react to changes in the environment and changes in information about where the other players may be hiding. Adaptivity is also supported through least-commitment planning, as the planner only looks ahead one action at each level of its abstraction hierarchy. The implementation follows the two-level agent architecture: the Intentional Planning System (ItPlanS)[Geib, 1995] interacts with a Search Planner[Moore, 1993] to perform the

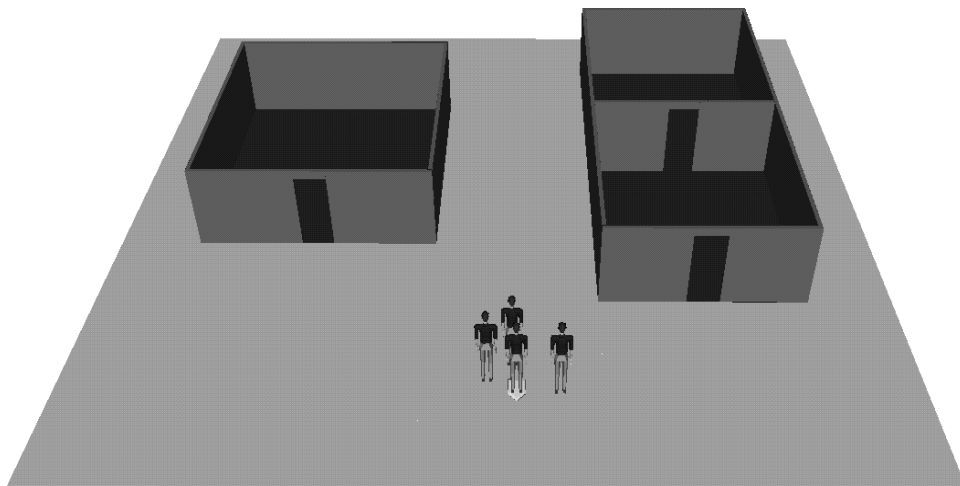


Figure 7: The start of a game of hide and seek

“high-level” reasoning for the system, and these two components in turn interact with a set of “low-level” SCA nodes based on Stepper (Figs. 7 and 8).

5.1 System Architecture

Our division of the control of a player between a *planning component* and a *reactive behavior component* reflects a distinction between deliberative actions (ones requiring non-local reasoning about the past, the present, and possible futures) and non-deliberative actions. In this sense, keeping track of where you are located in a complex environment and what hiding places have been checked requires deliberate effort, while walking from one place to another generally does not. Together, these two components create realistic animations of human decision-making and locomotion while playing hide and seek.

Fig. 9 depicts information flow in ZAROFF. To control the player in the role of seeker, the system starts by initializing the plan with the input goal (finding a hiding human), populating the database with the initial locations of all the objects and human figures in the simulation, and creating a partial map from what the player can see around him. ItPlanS and the SCA loop start processing simultaneously. The planner queries the state of the database through the Filtered Perception module to decide how to elaborate the plan and select an action. If necessary, the Search Planner is consulted to assist in planning

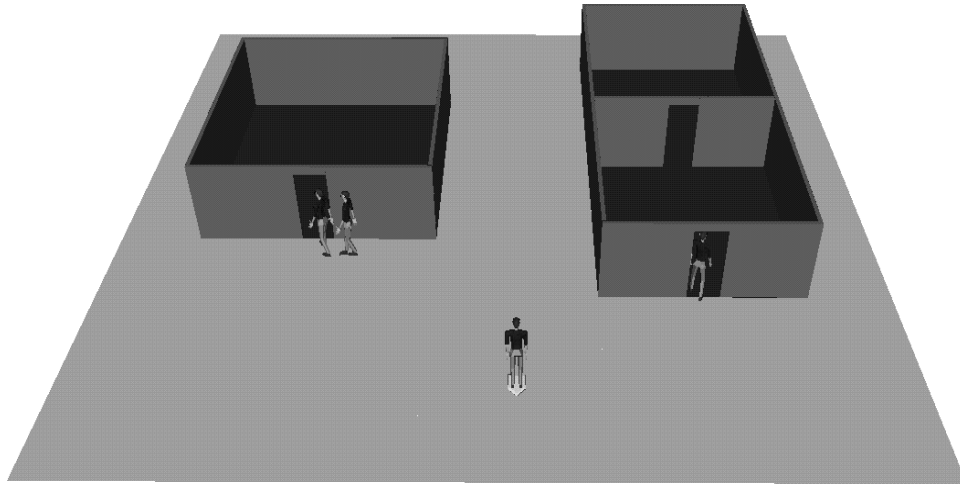


Figure 8: Hiders hiding – seeker counting

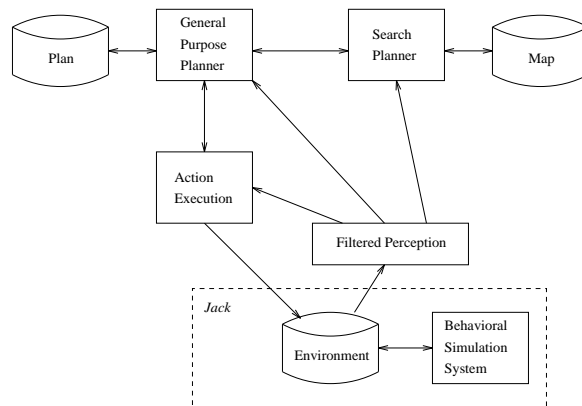


Figure 9: Information flow in ZAROFF

how to find things. When ItPlanS decides on an action, it instructs Action Execution to carry it out. Further planning is suspended until the action has terminated (successfully or unsuccessfully).

In making decisions about what to do next, each component makes use of its own internal simulation, which differs from the graphical animation of the environment. ItPlanS uses abstract descriptions of the effects of each action to choose one which will move closer to the specified goal. The Search Planner simulates the movements of an agent on its internal map of the environment. Stepper simulates taking the next step in several alternate locations. At each level of decision making, an internal simulation is used *at an appropriate granularity*.

5.2 Action in ZAROFF

Actions chosen by ItPlanS are carried out by an Action Execution module (see Fig. 9). Both components are well matched to the dynamic environment in which ZAROFF acts: the planner quickly selects the next action to perform based on comparison between the perceived world state and a partial hierarchical plan that is regularly revised. The action execution module controls locomotion via the Stepper system, enabling it to react to unexpected events such as moving obstacles, changing terrain, or a moving goal [Reich et al. 1994].

5.3 Planning in ZAROFF

ITPLANS [Geib, 1992] is a hierarchical planner, in which hierarchical expansion only takes place to the degree necessary to determine the next action to be carried out. It consists in an incremental expansion of the frontier of the plan structure to successively lower levels of abstraction. The incremental nature of the plan allows the system to make commitments at the appropriate level of detail for action while not committing the system to future actions that might be obviated by changes in the world. The close coupling of ITPLANS with the environment manifests itself in two ways:

First, the traversal and pruning process the planner follows at each interval relies on being able to determine the actual state of the world and compare that with its goals. During the expansion process ITPLANS examines the state of the world and its memory to determine if any of the goals within its plan have been satisfied. When a goal has been

satisfied serendipitously, it can be pruned out of the plan structure, and the system can move on to consider its next goal.

Second, ITPLANS “leans on the world” [Agre, 1988] when predicting the results of its actions. Rather than maintaining a complete model of the world and the state that results from executing the action, ITPLANS uses a simpler method based on associating conditional add and delete lists with each action. ITPLANS assumes that a given proposition is true in the state that results from the action if (1) the proposition is explicitly added by the add list or (2) the proposition is true *now* in the world and it is not contained on the delete list. By this method, ITPLANS can make predictions about the results of executing an action without modeling the entire world state.

5.4 Search planning

A consequence of limited perception is the occasional need to find objects. Our approach is to isolate this reasoning in a specialized module, a Search Planner that translates information acquisition goals to high-level physical goals to explore parts of the environment.

Searches are planned by first identifying known regions where an object may be located and systematically exploring this space. A plan is developed for exploring each region in turn. After such an exploration plan is executed, the environment is observed to determine whether the agent can see an object with the desired properties. During this observation phase, new potential regions may be seen by the agent. These new regions are considered for future exploration as needed.

5.5 Distinctions between the Upper and Lower Level

ZAROFF’s stratification into higher-level and lower-level components is a reflection of differences in informational needs. For example, one common low-level behavior is responsible for causing a figure to locomote from one position to another. Locomotion decisions require very detailed information – *e.g.*, foot positions, distances and angles.

In contrast, ITPLANS is responsible for sequencing locomotion actions with actions to open doors in order to explore various hiding places within the game field. The information needed to build plans at this level is at a different level of abstraction. ITPLANS needs to know “Is the door open?”, “Am I at the door?”, *etc.* While such information can be derived

from lower-level information, neither of the modules has need of the information that the other uses.

Differences in information needs provide one reason to separate higher-level and lower-level components, but there are other differences as well. One is level of commitment to future action. In locomotion, while each foot step brings the agent close to its current goal, no commitment is made to steps that it has yet to take. In contrast, `ITPLANS` makes commitments to act in specified ways in the future. For example, when the agent decides to walk to the door, it has already committed to opening the door once it has gotten there. While such commitments are tempered by possible changes in the environment, as long as a commitment remains active, it can affect the agent’s decision-making and behavior.

While both the higher-level and lower-level components of `ZAROFF` are environmentally responsive, another difference motivating the separation of these two levels is the “granularity” of their reactivity, both in the form and time-scale of the environmental feedback they depend on. On the one hand, lower-level behaviors are locked into an `SCA` loop and are thus tightly coupled to the environment. On the other hand, the higher-level system only receives feedback from the world “between actions”, when the lower-level system chooses to relinquish control. Thus, `ITPLANS` only receives feedback whenever there is no low-level behavior executing. This means that the time intervals over which the two different components receive feedback differ in magnitude. This discussion has noted a number of differences between the high level components of the `ZAROFF` system and the lower level control system. We now argue why they are valuable within the system architecture.

Separating low-level motor control from high level planning decisions is valid: a symbolic planner is inappropriate for making decisions about foot placement, and likewise local potential field calculations are inappropriate for making long-range plans. While the benefits of adding a reactive controller to a planner are well known, the relationship between these two components is symbiotic. While the reactive controller adds flexibility and an ability to handle local disturbances to the plan, if properly constructed the high level planning can result in the reduction of the complexity of the problem that the controller must solve.

Our solution to the problem of path planning illustrates this: path planning is done by navigating from one “landmark” to another. That is the search planner breaks any extended locomotion task into a series of smaller locomotion tasks. These “way points” are

then passed one at a time to the locomotion controller to move the system to the desired location. Thus by performing symbolic path planning in the form of identifying waypoints for the trip, each of the individual problems that the lower level system must solve are reduced in size.

6 Gesture Jack

“Gesture Jack” is a demonstration system that consists of two embodied agents holding a conversation where one agent has a specific goal and the other tries to help achieve it [Cassell et al. 1994]. All parts of the conversation have been automatically synthesized and animated: intonation, gesture, head and lip movements, and their inter-synchronization. Gesture Jack combines a dialogue planner (which moderates the communicative acts between the agents) with PaT-Nets (which control the speaker/listener roles and various non-verbal aspects of the intercourse). Motor actions drive the face, head, lips, and eyes. PaT-Net schemas control head and eye movements, as these relate directly to the agent’s role in the conversation. The face and lips are controlled directly from behavior inputs to the SCA loop, but the absence of direct sensory inputs means that the SCA loop is much simplified in comparison to ZAROFF.

The test scenario consists of two agents: Gilbert, a bank teller, and George, a patron who asks Gilbert for help in obtaining \$50. The dialogue is unnaturally repetitive and explicit in its goals because the dialogue generation planner lacks the conversational inferences that allow human conversationalists to follow leaps of reasoning. Therefore, the two agents have to specify in advance each of the goals they are working towards and steps they are following (see Section 6.2).

Gilbert: Do you have a blank check?
George: Yes, I have a blank check.
Gilbert: Do you have an account for the check?
George: Yes, I have an account for the check.
Gilbert: Does the account contain at least fifty dollars?
George: Yes, the account contains eighty dollars.
Gilbert: Get the check made out to you for fifty dollars
and then I can withdraw fifty dollars for you.
George: All right, let's get the check made out to me
for fifty dollars.

When Gilbert asks a question, his voice rises. When George replies to a question, his voice falls. When Gilbert asks George whether he has a blank check, he stresses the word “check”. When he asks George whether he has an account for the check, he stresses the word “account”.

Every time Gilbert replies affirmatively (“yes”), or turns the floor over to Gilbert (“all right”), he nods his head, and raises his eyebrows. George and Gilbert look at each other when Gilbert asks a question, but at the end of each question, Gilbert looks up slightly. During the brief pause at the end of affirmative statements (always George here) the speaker blinks. To mark the end of a question, Gilbert raises his eyebrows.

In saying the word “check”, Gilbert sketches the outlines of a check in the air between him and his listener. In saying “account”, Gilbert forms a kind of box in front of him with his hands: a metaphorical representation of a bank account in which one keeps money. When he says the phrase “withdraw fifty dollars,” Gilbert withdraws his hand towards his chest.

6.1 Gesture Jack Structure

In the Gesture Jack system, we have attempted to adhere to a model of face-to-face interaction suggested by the results of empirical research [Scherer, 1980]. In particular, each conversational agent is implemented as an autonomous construct that maintains its own representations of the state of the world and the conversation, and whose behavior is determined by these representations. (For now, the two agents run copies of the same program, initialized with different goals and world knowledge.) The agents communicate with one

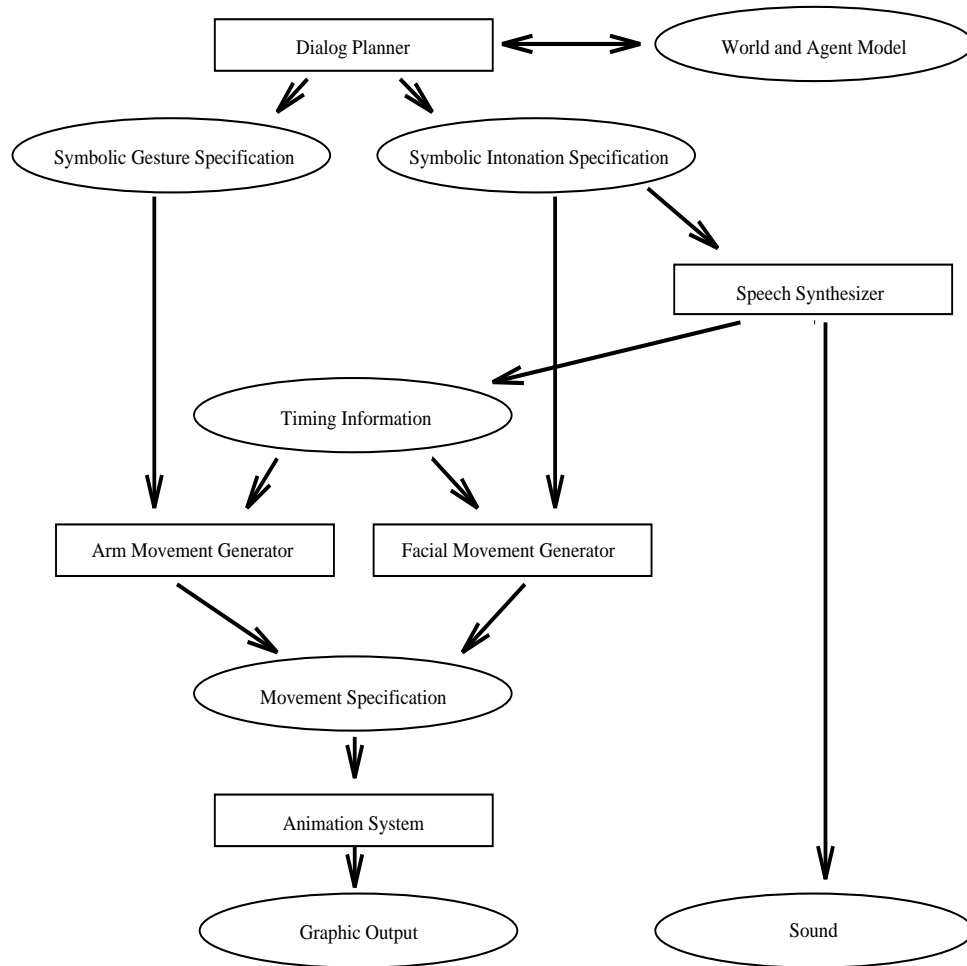


Figure 10: Architecture of each conversational agent

another only by the symbolic messages whose content is displayed in the resulting animation. (If their SCA loops were fully modeled, they would actually be able to *interpret* the speech and gestures of the other agent.) The architecture of a conversational agent is shown in Fig. 10.

In this section, we provide an outline of how each agent decides what to say, determines the contribution of this content to the conversation, and uses the resulting representations to accompany speech with contextually appropriate intonation, gesture, facial expression and gaze.

6.2 Dialogue Planner

The selection of content for the dialogue by an agent is performed by two cascaded planners. The first is the domain planner, which manages the plans governing the concrete actions which an agent will execute; the second is the discourse planner, which manages the communicative actions an agent must take in order to agree on a domain plan and in order to remain synchronized while executing a domain plan.

The input to the domain planner is a database of facts describing the way the world works, the goals of an agent, and the beliefs of the agent about the world, including the beliefs of the agent about the other agent in the conversation. The domain planner executes by decomposing an agent's current goals into a series of more specific goals according to the hierarchical relationship between actions specified in the agent's beliefs about the world. Once decomposition resolves a plan into a sequence of actions to be performed, the domain planner causes an agent to execute those actions in sequence. As these goal expansions and action executions take place, the domain planner also dictates discourse goals that an agent must adopt in order to maintain and exploit cooperation with their conversational partner.

The domain planner transmits its instructions to take communicative actions to the discourse planner by suspending operation when such instructions are generated and relinquishing control to the discourse planner. Several stages of processing and conversational interaction may occur before these discourse goals are achieved. The discourse planner must identify how the goal submitted by the domain planner relates to other discourse goals that may still be in progress. Then content for a particular utterance is selected on the basis of how the discourse goal is decomposed into sequences of actions that might achieve it.

When the dialogue is generated, the following information is saved automatically: (1) the timing of the phonemes and pauses, (2) the type and place of the accents, (3) the type and place of the gestures [Cassell et al. 1994, Prevost and Steedman, 1993, Steedman, 1991]. This speech and timing information is critical for synchronizing the facial and gestural animation.

6.3 Symbolic Gaze Specification

In the current version of the program, head and eye behaviors are not differentiated, and follow the same movement pattern. Gesture Jack uses four types of gaze (defined in terms of head motion), related to dialogic function [Argyle and Cook, 1976, Collier, 1985] (Fig. 11).

planning: corresponds to the first phase of a turn when a speaker organizes her thoughts. A speaker tends to look away at this point. Then, during the execution phase, a speaker looks more at a listener. For a short turn (duration less than 1.5 sec.), speaker and listener establish eye contact (mutual gaze) [Argyle and Cook, 1976].

comment: accompanies and comments speech, by occurring in parallel with accent and emphasis. Accented or emphasized items are punctuated by head nods; the speaker looks toward the listener. A speaker also gazes at a listener more when asking a question, looking up at its end.

control: controls the communication channel and functions as a synchronization signal: responses may be demanded or suppressed by looking at the listener. When a speaker wants to relinquish her turn, she gazes at the listener at the end of the utterance. When a listener asks for the turn, she looks up at the speaker.

feedback: is used to collect and seek feedback. A listener can emit different reaction signals to the speaker's speech. Speaker looks toward the listener during grammatical pauses to obtain feedback on how utterances are being received. This is frequently followed by the listener looking at the speaker and nodding. In turn, if the speaker wants to keep her turn, she looks away from the listener. In this and other ways, listener and speaker react to each other's behavior.

6.4 Using PaT-Nets in Gesture Jack

Interaction between agents and synchronization of gaze and hand movements to the dialogue for each agent are accomplished using PaT-Nets, which allow coordination rules to be encoded as simultaneously executing schemas. Each agent has its own PaT-Net: probabilities and other parameters appropriate for an agent are set for the PaT-Net, given its

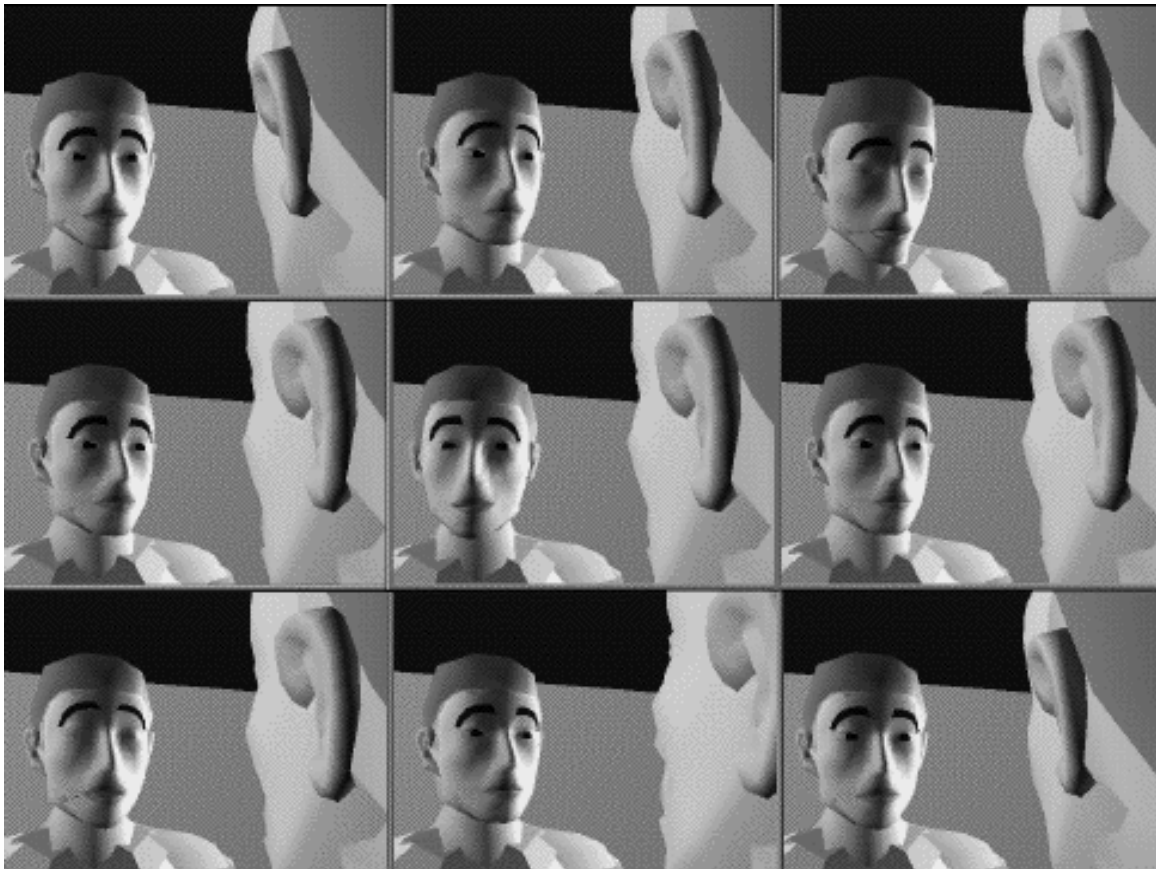


Figure 11: Facial expressions and gaze behavior corresponding to: “All right. <pause> You can write the check”.

current role as listener or speaker. Then as agents' PaT-Nets synchronize the agents with the dialogue and interact with the unfolding simulation, they schedule activity that achieves a complex observed interaction behavior.

The Pat-Net input file is a list of lists of the form:

(Speaker Listener Utterance Start-time)

The PaT-Net parses each "utterance-input" and schedules actions when condition are true. It stops when no phonemes remain in the utterance. At this point, the agents' Gesture and Gaze PaT-Nets send information about timing and type of action to the animation system. The animation itself is carried out by *Jack*.

The Gaze and Gesture PaT-Net schedule motions as necessary, given the current context, in semi-real time. Animation is performed as the input utterances are scanned, which allows for interactive control and system extensibility.

6.5 Gaze PaT-Net

Each of the four dialogic functions (planning, comment, control and feedback) appears as a sub-network in the PaT-Net, represented by a set of nodes, a list of conditions and their associated actions. Each node has an associated probability, based on an analysis of two-person conversations, noting where and when a person is gazing, smiling and/or nodding. Each of these signals is binary-valued – *e.g.*, gaze is equal to 1 when a person is looking at the other person, 0 when looking away. The conversation is annotated every tenth of a second. Six turn-states are considered, three per agent. When an agent hold the floor she can be speaking while the other agent is pausing (normal turn) or speaking (overlapping talk or a backchannel signal), or they can be pausing simultaneously. For each of these turn-states, we compute the co-occurrence of signals (nod, smile and gaze) and their probability. Separate PaT-Net nodes correspond to each different turn-state and signal occurrence; for example, the occurrence of a "within-turn signal" corresponds to the action: agent1 looks at the agent2 while having the floor and pausing.

Fig. 12 shows the Gaze PaT-Net (node names in bold, actions in italic and conditions specified on the arcs) and two sub-PaT-Nets: Head-Start and Gaze-Start. The latter support independent control of head and gaze (*e.g.* gazing at or away from the other agent,

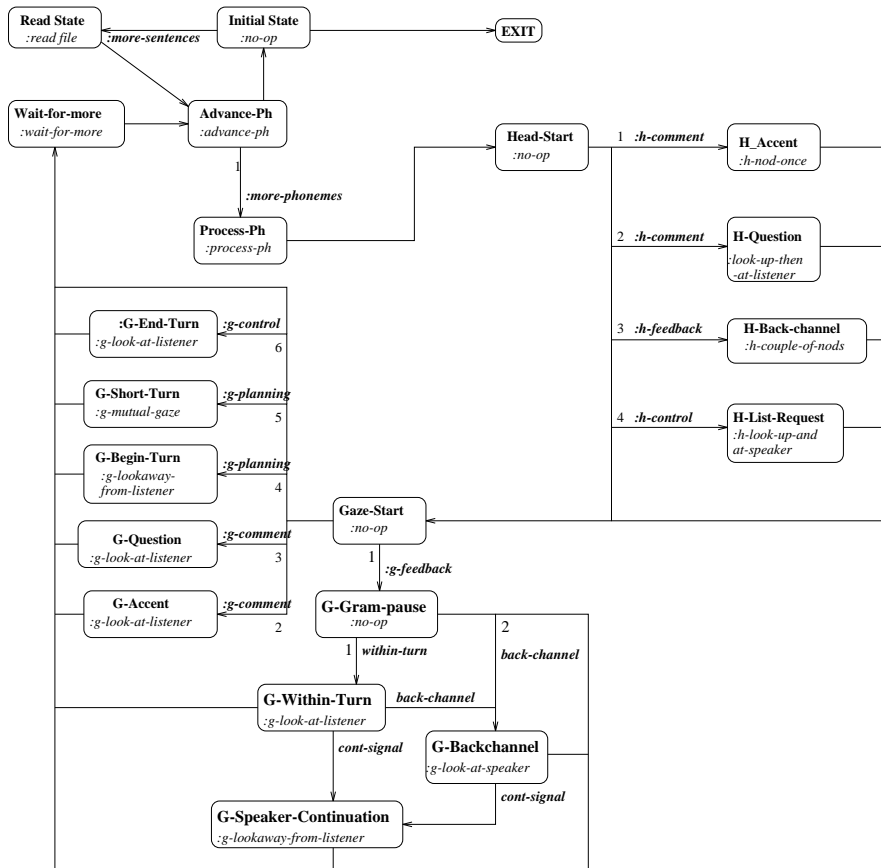


Figure 12: Gaze PaT-Net

and other head movement such as nodding, shaking, etc).

Each dialogic function is associated with a set of nodes. A node is entered only if the input condition is true, with its action(s) occurring probabilistically. For example, the dialogic function **planning** is defined by the following nodes and conditions: if a short turn is detected, the node G-Short-Turn is entered, with probability P of the speaker and listener looking at each other. The other node corresponds to begin-turn (comprising all phonemes between the first one and the first accented segment), where the speaker looks away.

6.6 Gaze Animation

For each phoneme, the Gaze PaT-Net is entered and a decision made regarding head movement. If no action is performed on a phoneme, the PaT-Net waits for the next available

phoneme.

Some actions performed by an agent influence the behavior of the other agent. In the case of the feedback node, different branching is possible depending on whether the speaker performs a “looking at the listener” action. The probability of a back-channel occurring (“listener looking at speaker”) is smaller if the speaker does not look at the listener and greater otherwise.

Head motions are performed by tracking an invisible object in the current environment. All swing, nod, turn movements are obtained by giving new coordinates to the object. A head movement is performed by giving the new position of the object, the starting time and duration of the movement. The head velocity has a smooth easy-in/easy-out pattern, that is it accelerates at the beginning and decelerates before stopping. A nod is simulated by moving the invisible object in the vertical plane, while swing and turn are executed by moving it in the horizontal plane. Each of these displacements takes as parameters the number of times to perform the move (simulation of multiple nods or swings), and distance (execution of large or small head nods). Varying these parameters allows one to use the same function for different conditions. For example, when punctuating an accent, the speaker’s head nod will be of larger amplitude than the feedback head nods emitted by the listener. A gaze direction is sustained until a change is made by another action.

We illustrate a PaT-Net execution with the following example:

```
Gilbert:  Get the chEck made OUt to you for fifty dollars <pause> And  
thEn <pause> I can withdrAw fifty dollars for you.
```

planning: This is a long utterance, so the node `short-turn` is not entered. For the first few phonemes (“Get the ch”), the node `beginning-turn` is entered, but the low probability action `speaker gazes away` does not occur. Therefore the speaker (Gilbert) keeps his current gaze direction (looking at George).

comment: On accented items (“chEck”, “thEn” and “withdrAw”), the node `accent` of the function `comment` is reached. In the first two cases the probability allows the actions `speaker gazes at the listener` to be performed by Gilbert, while `nod-once` by Gilbert results on “withdraw”.

control: At the end of the utterance (*i.e.*, the phonemes between the last accented segment and the last phonemes – here, “fifty dollars for you”), speaker and listener perform an action: **speaker gazes at listener** from the node **end of turn** and **listener gazes at the speaker** and up from the node **turn request**.

feedback: The two intonational phrases of our example (*get the check made out to you for fifty dollars* and *and then*) are separated by a pause; this corresponds to a within-turn situation. The node **G-feedback** is entered. If the probability allows it, the action **speaker gazes at the listener** is performed. If not, the arc going to the node **back-channel** is immediately traversed without waiting for the next phonemic segment. After a program-specified delay (0.2 sec.) the probabilities associated with the actions is checked once more. If allowed the node **back-channel** is reached and the action can happen: **listener gazes at the speaker**. In either case, the final step corresponds to the reaching of the node **speaker-continuation** after some delay. The action **speaker gazes away from the listener** is then performed.

7 Conclusion

We have demonstrated through these three systems that the combination of high level control through planners and PaT-Nets with low level SCA loops yields interesting, human-like, “intelligent” behaviors. Removing any one of these three components would incapacitate performance in notable and indeed crippling ways.

For example, if the planners were removed from Hide and Seek, then all decision-making and actions would need to be encoded in PaT-Nets, including the opportunistic generation of new goals (go to a found hider rather than the location the seeker was heading toward) and a hider’s choice of where to hide and a seeker’s choice of where to seek next. In the case of the dialogue planner in Gesture Jack, its symbolic reasoning (such as backward chaining or question generation) to determine a series of intermediate steps toward an overall goal would have to be encoded in PaT-Nets. Overloading PaT-Nets with these sorts of reasoning and planning would require a full programming language capability to specify arc transitions and a loss of locality that would be an end to any perspicuity. Indeed, the burden would once again be returned to the animator who would need to virtually “program” the entire

agent to select actions and anticipate any contingencies based on whatever features the immediate environment presented.

If the PaT-Net schemas were omitted, the planners would be forced to do too much work to see that sequential or coordinated tasks were carried out. Rather than making the planner disambiguate overlapping or interleaved activities, the schemas can manage resource allocation and coordinated activities. Thus the PaT-Nets in Gesture Jack can manipulate the head nods and eye movements needed for speaker/listener turn-taking without imposing a load on the actual dialog content planner.

If the SCA loop were omitted, the burden of managing all the environmental complexity must be foisted off onto some higher-level (symbolic reasoning) controller. It appears unrealistic to expect that a symbolic planner worry about where to place a foot during locomotion (Stepper). Likewise, a PaT-Net should not be used to explicitly manage the sensory feedback and decision-making that can check for and avoid hazardous terrain features or other obstacles. The low level SCA loop provides a kind of “reflex” action which can adapt to situations without requiring cognitive overhead.

We believe that ongoing research into embodied human-like simulated agents will find, as we have, that this architecture of high level schemas and planners combined with low level SCA loops will achieve increasing success in producing intelligent and realistic behavior.

8 Acknowledgements

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory; ARPA AASERT DAAH04-94-G-0362; DMSO DAAH04-94-G-0402; ARPA DAMD17-94-J-4486; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; DMSO through the University of Iowa; and NSF CISE CDA88-22719.

References

[Agre, 1988] Agre, P. (1988) “The Dynamic Structure of Everyday Life.” Technical Report 1085, MIT Artificial Intelligence Laboratory.

- [Argyle and Cook, 1976] Argyle, M. and Cook, M. (1976) *Gaze and Mutual Gaze*. Cambridge University Press.
- [Badler et al. 1991] Badler, N.I., Barsky, B.A. and Zeltzer, D. (editors) (1991) *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*, Morgan-Kaufmann, San Mateo, CA.
- [Badler et al. 1993a] Badler, N.I., Hollick, M.J. and Granieri, J. (1993) “Real-Time Control of a Virtual Human using Minimal Sensors.” *Presence* 2(1), pp.82-86.
- [Badler et al. 1993b] Badler, N.I., Phillips, C.B. and Webber, B.L. (1993) *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press, New York, NY.
- [Becket and Badler, 1993] Becket, W. and Badler, N.I. (1993) “Integrated Behavioral Agent Architecture.” In *The Third Conference on Computer Generated Forces and Behavior Representation*, Orlando, FL.
- [Becket, 1995] Becket, W.M. (1995) *Reinforcement Learning for Reactive Navigation of Simulated Autonomous Biped*s. PhD thesis, University of Pennsylvania.
- [Becket, 1994] Becket, W.M. (1994) The *Jack* Lisp API. Technical Report MS-CIS-94-01, University of Pennsylvania.
- [Braitenberg, 1984] Braitenberg, V. (1984) *Vehicles: Experiments in Synthetic Psychology*. The MIT Press, Cambridge, MA.
- [Brooks, 1986] Brooks, R.A. (1986) “A Robust Layered Control System for a Mobile Robot.” *IEEE Journal of Robotics and Automation*, pp.14-23.
- [Cassell et al. 1994] Cassell, J., Pelachaud, C., Badler, N., Steedman, M., Achorn, B., Becket, W., Douville, B., Prevost, S. and Stone, M. (1994) “Animated Conversation: Rule-Based Generation of Facial Expression, Gesture and Spoken Intonation for Multiple Conversational Agents.” *Computer Graphics*, Annual Conference Series, pp.413-420.
- [Cohen, 1992] Cohen, M.F. (1992) “Interactive Spacetime Control for Animation.” *Computer Graphics*, 26(2), pp.293-302.

- [Collier, 1985] Collier, G. (1985) *Emotional Expression*. Lawrence Erlbaum Associates, NY.
- [Firby, 1992] Firby, R.J. (1992) “Building Symbolic Primitives with Continuous Control Routines.” In *Artificial Intelligence Planning Systems*.
- [Geib, 1992] Geib, C.W. (1992) Intentions in means-end planning. Technical Report MS-CIS-92-73, Department of Computer and Information Science, University of Pennsylvania.
- [Geib, 1995] Geib, C.W. (1995) *The Intentional Planning System: ItPlanS*. PhD thesis, University of Pennsylvania.
- [Georgeff and Lansky, 1990] Georgeff, M.P. and Lansky, A.L. (1990) “Reactive Reasoning and Planning.” In *Readings in Planning*, J. Allen, J. Hendler, and A. Tate (editors), Morgan Kaufmann, San Mateo, CA, pp.729-734.
- [Liu, 1994] Liu, D., Gortler, S.J. and Cohen, M.F. (1994) “Hierarchical Spacetime Control.” *Computer Graphics*, Annual Conference Series, pp.35-42.
- [Moore, 1993] Moore, M.B. (1993) Search plans. Technical Report MS-CIS-93-56, Department of Computer and Information Science, University of Pennsylvania.
- [Moore et al. 1995] Moore, M.B., Geib, C.W. and Reich, B.D. (1995) “Planning and Terrain Reasoning.” In *AAAI Spring Symposium on Integrated Planning Applications* (also, Technical Report MS-CIS-94-63, Computer and Information Science, University of Pennsylvania.)
- [Ngo and Marks, 1993] Ngo, J.T. and J. Marks, J. (1993) “Spacetime Constraints Revisited.” *Computer Graphics*, Annual Conference Series, pp.343-350.
- [van de Panne and Fiume, 1993] van de Panne, M. and Fiume, E. (1993) “Sensor-Actuator Networks.” *Computer Graphics*, Annual Conference Series, pp.335-342.
- [Prevost and Steedman, 1993] Prevost, S. and Steedman, M. (1993) “Generating Contextually Appropriate Intonation.” In *Proc. of the Sixth Conference of the European Chapter of the Assoc. for Computational Linguistics*, Utrecht, pp.332-340

- [Reich et al. 1994] Reich, B.D., Ko, H., Becket, W. and Badler, N.I. (1994) "Terrain Reasoning for Human Locomotion." In *Proceedings of Computer Animation '94*, Geneva. IEEE Computer Society Press, pp.996-1005.
- [Renault et al. 1990] Renault, O., Magnenat-Thalmann, N. and Thalmann, D. (1990) "A Vision-Based Approach to Behavioral Animation." *The Journal of Visualization and Computer Animation*, 1(1), pp.18-21.
- [Reynolds, 1987] Reynolds, C.W. (1987) "Flocks, Herds, and Schools: A Distributed Behavioral Model." *Computer Graphics*, 21(4). pp.25-34.
- [Reynolds, 1988] Reynolds, C.W. (1988) "Not Bumping into Things." SIGGRAPH Course 27 Notes: Developments in Physically-Based Modeling, pp.G1-G13.
- [Robertson, 1994] Robertson, B. (1994) "Caught in the Act." *Computer Graphics World* 17(9), pp.23-28.
- [Scherer, 1980] Scherer, K.R. (1980) "The Functions of Nonverbal Signs in Conversation." In *The Social and Physiological Contexts of Language*, H. Giles and R. St. Clair (editors), Lawrence Erlbaum, New York, pp.225-243.
- [Sims, 1994] Sims, K. (1994) "Evolving Virtual Creatures." *Computer Graphics*, Annual Conference Series, pp.15-22.
- [Steedman, 1991] Steedman, M. (1991) "Structure and Intonation." *Language*, 67, pp.260-296.
- [Tu and Terzopoulos, 1994] Tu, X. and Terzopoulos, D. (1994) "Artificial Fishes: Physics, Locomotion, Perception, and Behavior." *Computer Graphics*, Annual Conference Series, pp.43-50.
- [Weber et al. 1995] Webber, B., Badler, N., Di Eugenio, B., Geib, C., Levison, L. and Moore, M. (1995) "Instructions, Intentions and Expectations." *Artificial Intelligence Journal*, to appear.

- [Wilhelms and Skinner, 1990] Wilhelms, J. and Robert Skinner, R. (1990) “A ‘Notion’ for Interactive Behavioral Animation Control.” *IEEE Computer Graphics and Applications*, 10(3), pp.14-22.
- [Williams, 1990] Williams, L. (1990) “Performance-Driven Animation.” *Computer Graphics*, 24(4), pp.235-242.
- [Witkin and Kass, 1988] Witkin, A. and Kass, M. (1988) “Spacetime Constraints.” *Computer Graphics*, 22(4), pp.159-168.