Creature Smarts: The Art and Architecture of a Virtual Brain

Robert Burke rob, Damian Isla naimad, Marc Downie marcd, Yuri Ivanov yivanov, Bruce Blumberg bruce@media.mit.edu

Introduction: The Synthetic Characters Philosophy

Is it possible to build an artificial intelligence that's as clever, as adaptive and as captivating as the average dog?

That's the question that drives the Synthetic Characters group at the MIT Media Lab. We draw inspiration from animal behavior, experimental biology, and the brain and cognitive sciences, and apply lessons learned from those fields to the study of artificial intelligence. While there are many research groups that focus on a particular cognitive ability, we have found it instructive to build *complete systems* – creatures that can interact with each other and with human participants.

Our philosophy is influenced by Daniel Dennett's theory of the *intentional stance*, which states that if you know a being's beliefs about its world, and you also know its desires, then you can predict that being's actions [Dennett 1987]. Thus, if a virtual creature can represent beliefs about its world, and also has a capacity for desires, then we should be able to generate appropriate actions for that creature.

A complementary influence is Disney's notion of the *illusion of life.* As animators Frank Thomas and Ollie Johnson explain, "It is the change in shape that shows what the character is thinking. It is the thinking that gives the illusion of life. It is life which gives meaning to the expression." [Thomas 1981] If we provide characters with the *means to express their mental state,* an observer can infer their beliefs and desires.

Towards a Deeper Gaming Experience

We seek to build creatures that are:

- *Robust.* They can base decisions on imperfect knowledge of the world.
- *Reactive.* They can react appropriately to sudden changes in their environment.
- *Adaptable.* They can learn from their experience of the world.
- *Honest.* They possess enough perceptual integrity to be surprised when things don't go as planned.
- *Expressive.* They have personality, can express their surprise and remain in character.
- *Sensible.* They display some basic common sense, regardless of their personality.
- *Scalable.* How about a whole pack of creatures like that?

Over the past few years, the popularity of networked games has skyrocketed. These games are appealing partly because human opponents (and comrades) are so much more sophisticated – and thus offer a wider range of interactions – than any of the artificial intelligence seen in games today. At the same time, the gaming community has begun to explore the potential that exists for games that incorporate more intelligent and engaging creatures. AI will never *replace* the richness of human interaction, nor would we want it to. However, by building creatures with the characteristics listed above, we seek to *emulate* some of the richness that today is only found in interactions between humans. Interactions with synthetic characters could be just as rich, offering a new range of gaming experiences for which interaction with only humans would be not desirable. Who wants to play "man in background #4" anyway?

Imagine your average dog. Without any special effects or a sci-fi backdrop, a dog is interesting, fun and gives at the very least the *appearance* of understanding. You can form a relationship with a dog, and establish a common history. When you interact with a dog, you *know* you are in the presence of a fellow intelligence. How can we capture that fleeting quality? We might find out by studying the very creatures that exhibit it.

Why look at Natural Systems?

Existing biological systems (like ourselves) are able to sense, react to, and manipulate their environments. Many such systems also have a tremendous capacity for adaptation and learning. We look to natural systems for clues

and design principles. Animals do what they ought to do, and learn what they ought to learn. In short, we look to animal behavior, or ethology, because nature "does things right."

Intelligence is a highly multi-faceted discipline – each of ethology, psychology, neuroscience, experimental biology and “good ol’fashioned AI” approach it from a different angle, and each has its own unique tools and insights. But because the details of how biological systems work are still rather sketchy, our research has a dual purpose: during our attempts to advance artificial intelligence, we are simultaneously applying and exploring theories borrowed from these disciplines.

Why look at Dogs?

A problem with cartoon-like creatures is that they can behave however they want – there are no rules, and no way to ask, “how close did we get?” At the other extreme, attempting to honestly model all aspects of human intelligence would be setting the bar too high (at this stage in the game). Working on a dog provides a reasonable and interesting focus. And, from a behaviorist’s point of view, it provides us with a means for gauging our success. The gap between the intelligence exhibited by today's AI and a real dog is much greater than the gap between dog- and human-level intelligence.

There are at least two additional reasons why we choose to build dogs. The first is that everyone has an intuitive sense of how dogs should behave and react. And second, lest we anthropomorphize our canine friends, there is a vast amount of literature available on how dogs work. There are books that detail how to use operant conditioning techniques, like the "Clicker Training" paradigm that we model, to teach a dog to do tricks (see [Wilkes 1995]). There are books about how to train dogs to herd sheep and sniff for drugs at airports. In fact, there are almost as many books about dogs at amazon.com as there are books about sex. Enough said.

Paper (and lecture) Roadmap

We’ll begin with a discussion of our group’s history, which will bring us to *sheep|dog* and *Clicker,* the two installations built under c4, our most recent behavior architecture. We’ll then take Duncan, the highland terrier and star of those installations, as the primary case study for a detailed discussion of c4’s architecture. We will demonstrate how c4’s ethologically inspired design principles have led to intuitive behavior specification, a flexible and extensible system, and in the end a compelling cast of interactive creatures.

A Brief History of Characters

Over the past four years, the group has experimented with different approaches to creature design. We design, build, and then iterate.

The *Alive* project featured Silas T. Dog, our first ethologically inspired autonomous creature created by Bruce Blumberg. The participant could interact with Silas through the "magic mirror," a wall-sized display that presented a live video image of the participant, as well as the graphical creatures that co-exist with the participant in the world of the mirror. (See [Blumberg 1996].)

The next project, *Swamped*, featured an autonomous cartoon raccoon that was intent on stealing the eggs of a user-controlled cartoon chicken. The group coined the term “sympathetic interface” for the plush toy chicken the participant used to manipulate the corresponding virtual chicken. (See [Johnson 1999].)

(void)* featured three humanoids sitting at the bar in an all-night diner. Using the physical buns-and-forks interface invented by Charlie Chaplin in "The Gold Rush," the participant could "possess" a character and make it dance. The characters differed from puppets in that they had an emotional response to the interaction they were undergoing, and would change the style of their animation to reflect their emotional state.

Shortly before last year's Game Developers Conference, we re-designed our architecture. The new design is informed by lessons learned from our past efforts, and provides a new focus on learning. We presented the results of the system’s first iteration ("c1") at GDC 2000 in a demo called *The Isle of Man's Best Friend*. Three iterations later, we have arrived at c4, the system we discuss in detail here.

Enter c4

Some of the constraints on c4's design were completely practical: it needed to support graphics with at least a 20Hz frame rate, input devices (mice and microphones as well as more exotic interfaces), and distributed rendering (the ability to use multiple PCs to render multiple views of the world). It also needed to be scalable enough to support a reasonable number of autonomous creatures all sensing and reacting to each other and the world.

But more importantly, it needed to let us make smart creatures that learn the same kind of things that animals seem to learn. Intelligence is often seen as the combined effect of many individually unintelligent components (see [Minsky 1985]).[1] The architecture needed not only to support those components individually, but also to allow them to coexist and communicate coherently within one brain.

Finally, the system needed to make it easy to build creatures with ever-increasing behavioral complexity. No one would build creatures with this system if it were impossible to work with.

c4 is implemented in Java, and sits on top of the OpenGL-based Magic scenegraph, an open-source library implemented in C++ (see [Eberly 2001]). sheep|dog's creatures and worlds were modeled and animated using 3DStudio MAX. We use seamless mesh rendering techniques for the creatures.

Case study: *sheep|dog* and *Clicker*

We have implemented two significant projects to date under c4. The first is *sheep|dog*, an interactive installation in which a user plays the role of a shepherd who can interact through a series of vocal commands with Duncan, a virtual sheep dog, to herd a flock of sheep. This system demonstrated some of the basic reactive, perceptual and spatial abilities of the creatures built using c4.

The other project is *Clicker*, in which the user trains Duncan to perform a variety of tricks using the same "clicker training" technique used with real dogs. This system demonstrated the learning abilities of the creatures built using c4.

Figure 1: Duncan the highland terrier, his shepherd, and their sheep, in *sheep|dog: Trial By Eire*.

[1] See especially Minsky's *The Society of Mind,* in which he describes how a mind might be composed of many "agents," each simple and understandable, but through their interaction able to exhibit complex behavior. c4's Systems are analogous to Minsky's Agents.

System Architecture

c4's single World object contains the *World Model*, or the virtual world as it exists on a given timestep. The World maintains the list of creatures and objects, and acts as an event blackboard for the posting and distribution of world events. As well as network synchronization and render management.

World events take the form of *DataRecords*, perceptual nuggets that can represent anything from an acoustic pattern to a visual or symbolic event. The flow of DataRecords through the system is shown in Figure 2. As we'll discuss below, whether and how each creature interprets each DataRecord depends entirely on the sensory and perceptual abilities of that creature.

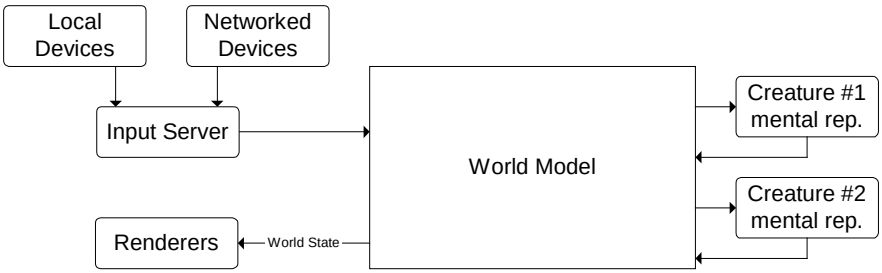

Figure 2: Flow of information within the system. Unmarked arrows indicate the flow of DataRecords, or perceptual nuggets, within the architecture. Formal abstractions exist between the world model and the creatures' mental representations. No links exist between the mental representations of different creatures; instead, creatures communicate by posting DataRecords to the world's event blackboard.

In a single execution of the update loop, DataRecord events are fed to each of the creatures and objects and, in turn, each is prompted to update. Most creatures and objects will themselves produce events that the World will harvest and make available in the next timestep. Pre- and post-update tasks, including UI updating, network coordination and rendering, are also performed by the World.

Cognitive Architecture

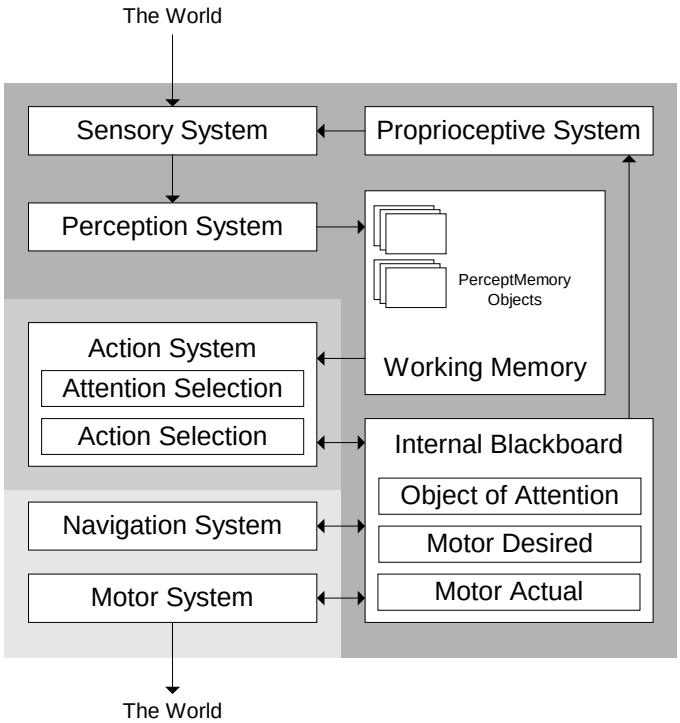

Figure 3: The c4 brain architecture, arranged into a "perception and memory" section, an "action selection" section, and a "motor and navigation" section. The systems are processed serially, in roughly top-to-bottom (or dark-to-light) order relative to this figure.

The “brain” of each creature implemented with c4 is organized into a collection of discrete *systems* that communicate through an internal blackboard. Any part of the brain can write to or read from specific “slots” in this blackboard that are differentiated by string labels. The canonical brain, depicted in Figure 3, is highly modular, with few essential subsystems and many opportunities for expansion. Duncan, the shepherd and each sheep use this brain structure.

The systems are processed serially, in roughly top-to-bottom order relative to the figure. We have arranged the systems by function into three subgroups: those that deal with how a creature *represents the world,* those that handle aspects of the creature’s *action selection*, and those that manage *navigation and motor control*.

How Creatures Represent the World

In order for a creature to have intentions, and thus perform actions on the basis of its beliefs and desires, the creature requires a representation of those beliefs and desires. A creature’s beliefs are necessarily distinct from the actual state of the world. The World knows that the sheep is at coordinates (32, 32, 0), but Duncan only knows where the sheep is relative to his own head. Therefore, creatures can’t simply be allowed direct access to the world model, or aspects of other creatures’ internal state. Instead, as indicated in Figure 2, *a formal division must exist between the world model and creatures’ mental representations.*

Why go to such great lengths to enforce this division? A creature should only be able to act on information that its “sensory apparatus” is able to observe. If there is an aspect of the world’s state that is important to a creature, the creature should create and maintain its own internal representation of that state. Though this may seem inefficient, in the following sections we’ll describe how this allows us to model a wide variety of mental phenomena that otherwise would be impossible to produce, and contribute greatly to a creature’s realism.

Sensory honesty (Sensory System)

The Sensory System marks the single entry point into a creature’s brain. All sensory input from the world must pass through the Sensory System before it can be processed by the rest of the brain. The primary job of the Sensory System is to act as the enforcer of *sensory honesty.* As such, it processes each DataRecord so that it appears as it would from the creature’s point of view. Sometimes that means removing it completely – for example, culling VisualDataRecords outside of the visual field – or otherwise transforming it into the appropriate reference frame.

Because the Sensory System offers a single entry point, it allows us to provide more or less uniform treatment to the many types of DataRecords that a creature may be called upon to process. But most importantly, it ensures that the input to a creature’s mental representation is biologically inspired. The alternative – allowing a creature to potentially see through walls, or through the back of its head – has a devastating effect on the illusion of life!

Classification hierarchy (Perception System)

Once the stimulus from the world has been “sensed,” it can then be “perceived.” The distinction between sensing and perceiving is important. A creature, for example, may “sense” an acoustic event, but it is up to the perception system to recognize it as an instance of a specific type of acoustic event that has some meaning to the creature. A sheep would interpret an UtteranceDataRecord as just another noise, but Duncan should classify the utterance as the word “sit,” and then sit down expecting to get a cookie. Thus, it is within the Perception System that each creature assigns a unique “meaning” to events in the world.

The Perception System takes the form of a *Percept Tree* (Figure 4). A *Percept* is an atomic classification and data-extraction unit that models some aspect of each DataRecord passed in by the Sensory System. It does this by returning a match probability or *confidence.* The SheepShapePercept, for example, will return the probability that a DataRecord represents the experience of seeing a sheep. If the match is above a threshold, a Percept also returns a piece of extracted data. The HeadLocationPercept, for example, returns the location of the perceived DataRecord relative to the coordinate frame of the creature’s head.

Percepts are organized hierarchically in terms of their specificity. For example, a child of a ShapePercept will activate on the presence of any kind of shape, whereas one of its children may activate only on a specific type of shape (e.g. a SheepShape.) This allows DataRecords to perform efficient partial traversals of the Percept Tree: if the ShapePercept returns a match probability of 0.0, we have no need to ask if its children, like the SheepShapePercept, will match the DataRecord.

Many Percepts are adaptive, using statistical models to characterize and refine their response properties. These Percepts can not only modulate their "receptive fields" (the space of inputs to which they will respond positively) but also, in concert with the Action System, *modify the topology of the tree itself*, dynamically growing a hierarchy of children in a process called *innovation*. As will be described below (see Action Selection), the process of innovation is behavior-driven. Only Percepts that allow the creature to make better decisions are prompted to innovate.

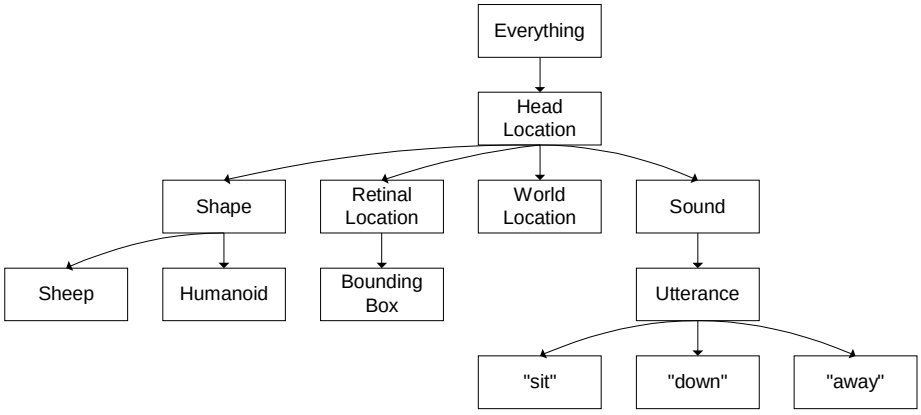

Figure 4: The Percept Tree. The EverythingPercept at the root of the tree returns a match probability of 1.0 for every DataRecord. The Head Location Percept is derived from the root of the tree because, at present, all events have locality. (A separate proprioceptive percept tree, also derived from the Everything node, is in the works.) Retinal location and shape apply only to visual events, just as the SoundPercept and its children only apply to auditory events.

The Percept provides a useful level of abstraction for reducing the dimensionality of incoming sensory information. An arbitrarily complex mechanism tucked into a Percept can determine whether or not it matches a particular DataRecord. Under the hood of the UtterancePercept, for example, is an utterance classifier that helps its child Percepts to classify different utterances. The rest of the creature's brain need not have any knowledge of how that classifier works. The classifier can easily be replaced with a different mechanism (as it was during debugging, when we elected to click a button rather than mumble "sit" into a microphone for hours on end).

Proprioception

Real animals use sensors that detect muscle extension and tendon strain to determine the position of their limbs. This type of self-awareness, or "self-perception," is called *proprioception* by biologists. We have found it useful to emulate proprioception and extend it to include many forms of self-awareness, including awareness of emotional state and of self-action.

The Proprioception System is the module responsible for this function. It provides a blackboard mechanism, analogous to the World event blackboard, where the creature can post events to itself that will not be visible to other creatures in the world. Other internal mechanisms can post aspects of their state in the form of ProprioceptiveDataRecords to the blackboard, and the Sensory System receives the contents of the Proprioceptive System's blackboard on the next timestep. This mechanism provides a common pathway for all sensory data, internal and external, that will be processed by the creature on a timestep.

Memory

Psychologists distinguish between different kinds of memory. Procedural Memory is the label given to skill learning – for example, learning to play piano, or throw a baseball. Another type of memory is Declarative Memory, which allows us to remember facts, such as, "my name is Warren," or, "Lake Titicaca is 4200 meters above sea level," or, "the pen is on the table." Declarative Memory is further broken down into Long-term Memory, which stores important events and facts, and Working Memory, which tracks the state of the environment that is relevant to the immediate task. (See [Reisberg 1997] for a summary of the different memory classifications.)

c4's *Working Memory* structure is meant to mirror this psychological conception of Working Memory. c4's Working Memory maintains a list of persistent *PerceptMemory* objects that together constitute the creature's "view" of the current context. That "view," however, is informed by more than just direct perception. Instead, it is a patchwork of perceptions, predictions and hypotheses. Any component of c4 that has something to say about

how the world is (or might be) can modify PerceptMemory objects or post new ones. It is on the basis of these objects, whether directly perceived or not, that action-decisions will be made, internal credit for reward will be assigned, motor-movements will be modulated, and so on.

PerceptMemory Objects

Although Percepts are useful for reducing the dimensionality of incoming sensory information, and transforming that information into a form that is meaningful to a creature, alone they are still not quite enough to capture all that we wish to know about the state of the world. Consider a Percept that detects "redness." Though that Percept would register activation on any timestep in which red was detected somewhere out in the world, this perception is of limited utility without knowing *what* about the world was red. In other words, the redness of a sensory event should not be separated from the other perceptual qualities of that event. This is especially necessary if, for example, the redness percept extracts a number representing the "degree of redness." If there are multiple red objects in the world, then there are various pieces of data that are relevant to different objects in the world.

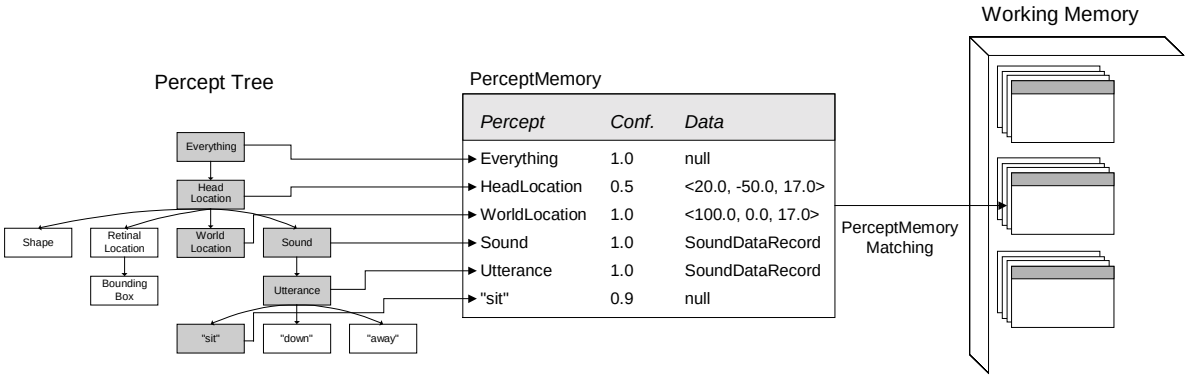

Figure 5: An illustration of PerceptMemory formation. Here an utterance is being recognized as the word "sit". A DataRecord representing the utterance comes in from the world, causing certain Percepts in the Percept Tree (left) to activate. The Perception System caches the confidences and data corresponding to these Percepts in a PerceptMemory object. This object then matches itself with the most similar existing PerceptMemory in Working Memory (right), and adds its data onto the history of data that it maintains.

When a DataRecord comes in from the world, all the percept activity that it causes is stored together in a PerceptMemory. PerceptMemory objects are a convenient way of bundling perceptual information as histories of percept activations and data, each of which corresponds to an object in the world. Significantly, this is the *only* form of object-representation in c4. To a c4 creature, a sheep is not a "sheep," but rather a "thing" that is "sheep-shaped" (it activates the creature's "sheep-shape detector percept"), white, and frequently issues an acoustic pattern that a human would characterize as a "baaah." To a creature, *an object is the set of perceptions it imparts.*

When a new DataRecord is pushed through the Percept Tree, each Percept that registers a positive match caches its confidence and data in a table in the PerceptMemory. This PerceptMemory is then passed off to Working Memory (See Figure 5). It is then "matched" against existing PerceptMemory objects stored there to determine if it is truly novel, or rather a continuation of an existing PerceptMemory (for example, is it the same red ball as the red ball that we saw an instant ago in the same place). In the case of visual events, matching is done on the basis of shape, or on the basis of location when shape is not enough to disambiguate incoming visual events (as is often the case with two nearby sheep). This matching mechanism also allows events of differing modalities to be combined. If there is good indication that an acoustic event and a visual one belong together (for example, they originate in more or less the same region of space) then they may be matched together in Working Memory, presenting both sight and sound information through a single PerceptMemory and thus giving the impression that, for example, it was the shepherd who said, "sit." In either case, if a match is found, the data from the new PerceptMemory is added to the history being kept in the old one. The new confidence is also added to a history of confidences. On timesteps in which new information for a given Percept is not observed, its confidence level is decayed. The rate of decay is determined in part by the Percept itself (confidence in another creature's location might decay rapidly without observation, but confidence in its shape probably would not.)

PerceptMemory objects are especially convenient because they allow us to query our memory in useful ways. Behaviors can be triggered by asking questions like, "is there food near me?" Action-targets can be picked by finding the "red object that is making the most noise." Much of the reasoning that a creature will typically perform is object-based reasoning. "Find an object that is humanoid-shaped and go to it" implies that you can extract both

shape and location information out of a structured object-based memory. This is what PerceptMemory objects allow.

Prediction and Surprise

The stream of sensory data coming from an object will often be interrupted because the object is out of the creature's visual field, or because it is occluded by another object. Our brains are uniquely adept at filling in these sensory gaps, often without our even realizing it. The existence of a "blind spot" in our visual fields is an oft-cited example: the area of our retina that has no light receptors is ascribed likely information. Another classic example, considered by the psychologist Piaget, is the ball-rolling-behind-the-wall example. When we see a ball disappear behind an occluder, the ball does not disappear for us conceptually. In fact, we maintain a fairly accurate estimate of the ball's likely location given its last observed velocity, and our other "common sense" knowledge of physics. When the ball fails to come out from behind the other side of the wall at the correct time, it's a surprise. This ability to maintain a model of an object even when the object is not directly observable is known as *object persistence*.

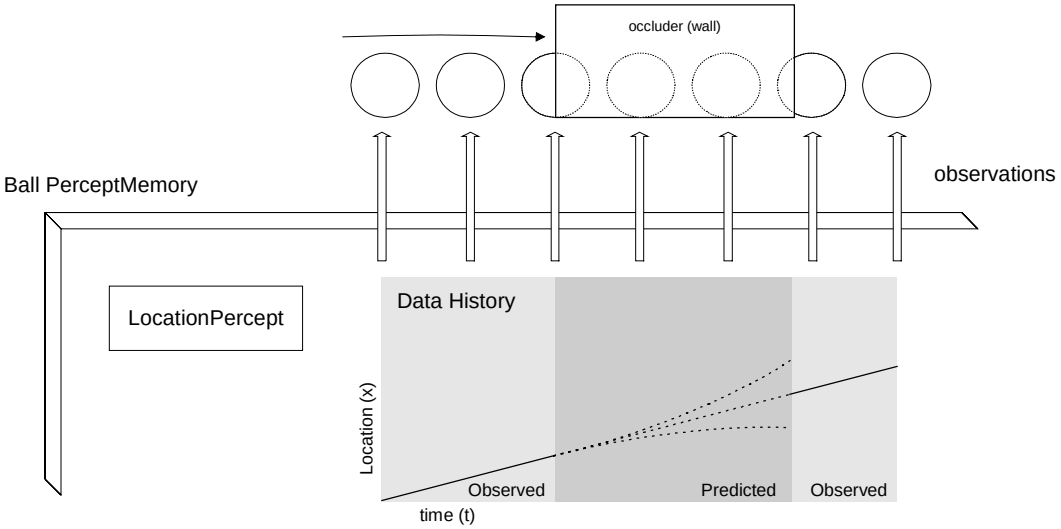

Figure 6: An example of object persistence. As the ball (top) rolls to the right behind the wall, the gap in the perception of its location is filled in by the creature's prediction mechanisms (dark gray area). Note that the range of possible locations in the LocationPercept's data history widens the longer the ball has remained unobserved, but narrows again when the ball becomes visible. A perception that behaves this way is called a *fluent*. See [Funge et al. 1999].

Part of the task of the Memory System is to provide some form of object persistence as an important component of basic physical common sense. When a Percept's data is not observed, its value is instead predicted and made available to the rest of the system *as if* it had been observed. That way, the creature can act on the position of the ball whether it is behind the wall or not.

How are these predictions made? It depends on the kind of Percept we are dealing with. Since PerceptMemory objects contain histories of percept data, it is possible, if the data is a vector or scalar, to use function approximation techniques to extrapolate its value. In other cases, proprioception can inform location predictions – a creature can predict an object's egocentric location based on its own movement and the object's last known world position (or extrapolated position, as described above). This technique is critical if the object has gone out of the creature's line of sight. Though it cannot see the object, it can still maintain an accurate idea of where the object is. It should also be emphasized that location is not the only form of percept data that can be predicted. Well-designed prediction mechanisms can detect and model all sorts of regularities over all sorts of percepts. If a certain stimulus is active for a brief period every *n* seconds, that regularity can be gauged by a good predictor. If the stimulus is important for determining appropriate behavior (sitting down whenever a light is on gets the creature food, say) then the creature can react to the stimulus *before* it occurs.

Predictions – their occasional deviation from the actual state of the world and the magnitude of that deviation – also provide a basis for *surprise*. Surprise is an excellent method for focusing perception, and a PerceptMemory that has just provided a surprising stimulus is an excellent candidate for the creature's object of attention. More importantly, however, prediction is a huge contributor to common sense and hence to the believability of a

creature. Predictions imply expectations about the world, and a creature that acts in anticipation of a future event can more truly be said to have *intentions*. There are a host of emotions associated with expectation that purely reactive creatures cannot possibly display, many of the arising from situation in which, for one reason or another, expectation failed to match reality. Creatures with expectations can be surprised, relieved, disappointed, confused, frightened, tricked, teased, misled, taunted, and so on.

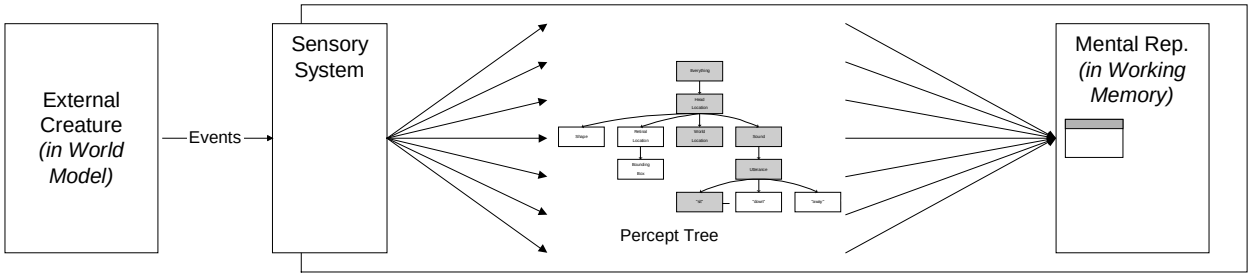

Figure 7: Each creature takes the world model, wrangles it through their Sensory and Perception Systems, and then re-constructs a mental representation of the world as PerceptMemory objects in Working Memory.

Why bother with all this stuff?

We conclude this section on Perception and Memory with a grand, "why bother?" We have gone to great lengths to enforce sensory honesty and emulate a psychological conception of memory. The situation is essentially that of Figure 7, which shows how incoming data from another creature in the world is split up, processed by a multitude of individual Percepts and then reassembled through a computationally intensive PerceptMemory matching mechanism. The question is, if we (as designers of the simulation) know that all that data is coming from a certain object in the world, then why bother splitting it up only to put it back together? What does "sensory honesty" buy us after all?

It actually buys us a lot:

- *Honest mistakes.* There are good mistakes and bad mistakes. Good mistakes are the kind of mistakes that we ourselves would make in confusing situations. They are the mistakes that make a creature believable. If a creature reacts to the monster before it has jumped out of the closet we sigh and roll our eyes. If the creature is surprised and shows it, then we empathize and say, "The poor thing didn't see it coming." In other words, we begin to ascribe to it human-like attributes! *What we don't know is as important as what we do know.*

- *Learning generalizable concepts.* Individual perceptual traits need to be pulled out of data through the Percept Tree because it allows general lessons to be drawn from the world. "Redness" is a trait shared by many things. If treating one red thing a certain way works, then maybe treating *all* red things the same way *also* works. Furthermore, the Percepts and the Percept Tree can themselves learn, gradually refining the creature's perceptual experience in both data-driven and reward-driven manners. This increased sensitivity towards certain aspects of perception (perhaps a creature becomes very good at distinguishing between shades of red if many things it sees are red) can then be applied to all objects it encounters, not just the particular object that taught it that sensitivity. As concepts are pulled apart and reconstructed, the creature can find rules of behavior that generalize across many objects, including ones it has never seen.

- *The same thing means different things to different creatures.* The arrows to the left of the Percept Tree in Figure 7 are the same for every creature, but the ones on the right will almost definitely be very different. This is because the particular Percepts that exist in a creature's brain will depend on the type of creature it is, as well as the creature's experience. Some Percepts, it will have learned, are important to carefully track and predict (and incidentally spend a lot of computation on). Other Percepts it will lack entirely (an acoustic event will pass unnoticed by a creature with no ears). Perhaps a more intelligent creature will perceive more, or be able to predict things better. Whatever the case, the arrows to the right of the Percept Tree in Figure 7 represent *subjective experience*, and thus by nature are different for each creature.

Action Selection

Given a set of PerceptMemory objects that detail the perceived state of the world, the creature must decide which action(s) it is appropriate to perform. This process is known as Action Selection. There are three essential issues to be addressed when designing an Action Selection mechanism. First, what is the fundamental representation of action used by the system? Second, how does the system choose which actions to perform at a given instant? Many types of decision-making processes are possible here. Third, how can the choice of action be modified to allow the creature to learn from experience?

Representing Action: ActionTuples

Any representation of action should address what we call the '4 Big Questions":

- When to do it?
- What to do and how to do it?
- What to do it to?
- For how long?

As an example, consider a rule for responding to the "away" command in the sheep-herding installation. In prose, the rule would look like: When you perceive the utterance "away," gallop around the closest group of sheep, circling so as to keep the sheep on your left side until you are on the opposite side of the flock from the shepherd." It is easy to pick out the four answers to the questions above. A frequent mistake is to assume that you can get away without answering all four. Our experience has proven otherwise.

The fundamental representation of action used by c4 is called an *ActionTuple*. As you can see from Figure 8, the structure of an ActionTuple closely mirrors the "4 Big Questions." This is done not only for computational reasons but also to make it easier for behavior designers to see how to use them.

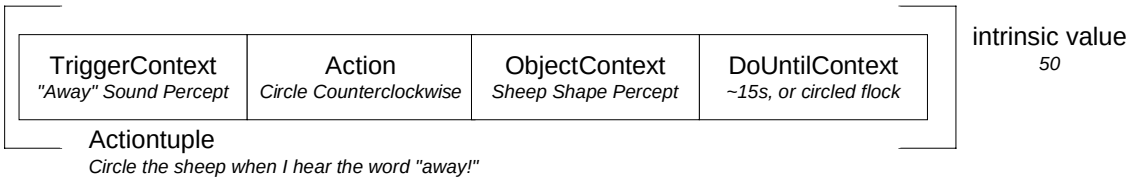

Figure 8: Example of an ActionTuple, the fundamental representation of action used by c4.

With the exception of learning, most of the functions of an ActionTuple are in fact handled by sub-components. For example, the *TriggerContext* of an ActionTuple is responsible for evaluating the relevance of the ActionTuple when it is inactive. TriggerContexts typically rely on specific Percepts in conjunction with the contents of Working Memory to arrive at their value. Figure 8 shows an ActionTuple that triggers upon the creature perceiving the specific utterance, "Away!" In practice, the trigger is usually a direct reference to a Percept. Thus, the ActionTuple in Figure 8 triggers in the presence of a PerceptMemory in Working Memory for which the AwaySoundPercept is active. The *DoUntilContext* associated with an ActionTuple is responsible for evaluating the continued relevance of the ActionTuple once it is active. The actual conditions may either be temporal ("continue for some period of time"), or state-based ("continue until some state is achieved"). An optional *ObjectContext* is utilized by ActionTuples whose actions are performed on or relative to some object, and specifies the conditions that object must meet. (For example, the ObjectContext of the ActionTuple that captures the notion of "stalk sheep" requires that the object being stalked exhibit the "SheepShapePercept")

When active, an ActionTuple's set of primitive actions typically modifies the contents of the creature's internal blackboard. These blackboard postings act as messages to other parts of the system. For example, the MOTOR_DESIRED posting holds the name of a physical behavior requested of the Motor or Navigation Systems. The ActionTuple also outputs a PerceptMemory representing the creature's current or desired focus of attention to the OBJECT_OF_ATTENTION posting. The PerceptMemory chosen for the posting is the one that best satisfies the conditions imposed by the ActionTuple's ObjectContext. This PerceptMemory will be used to direct the creature's gaze, as well as serving as the target for approaches, orientations and other targeted behaviors.

As will be discussed later in the paper, by virtue of the inclusion of an underlying navigation system, an ActionTuple can essentially specify “approach this object and, when within this distance and orientation, do X,” and rely on the navigation system to handle the mechanics of the actual approach. Since so much of behavior falls in the category of “approach and do” or “avoid and do,” this is a big win.

Selecting Actions: ActionGroups

Given a representation of action, the next issue is to decide which action(s) to perform at any given instant. It is important to note that typically a creature will be performing more than one action at a time. For example, while the dog may be engaged primarily in one action (such as charging around the sheep in response to the “Away” command), it is still performing a host of actions: visually tracking its target, choosing a gait to use, choosing how fast to wag its tail, and whether or not to bark. On the other hand, there are certain classes of actions that are mutually exclusive: running around the sheep and sitting down, for example. Thus, a good action selection mechanism must make it easy to organize mutually exclusive actions in such a way that only one is active at a time, but also make it easy to have multiple actions active simultaneously.

The approach taken in c4, as in many other systems, is to rely on the representation of action (in our case, the ActionTuple) to calculate its relevance – that is, how important it is that it be active, based on the state of the world. Actions are then typically organized into groups that are responsible for arbitrating among competing actions. In the simplest case where actions may run simultaneously, the group simply goes through each action, and if it has a non-zero relevance (i.e. value), it is allowed to run. The more complicated case is when the actions are mutually exclusive. In c4, *ActionGroups* are responsible for arbitrating among competing ActionTuples. Essentially, an ActionGroup relies on each of its ActionTuples’ evaluation of its instantaneous value given the relevant context. If inactive, the relevant context is its *TriggerContext*; if active, the relevant context is its *DoUntilContext*. The evaluated value, or *e-value*, of an ActionTuple is the product of the value of the relevant context and the ActionTuple’s intrinsic value.

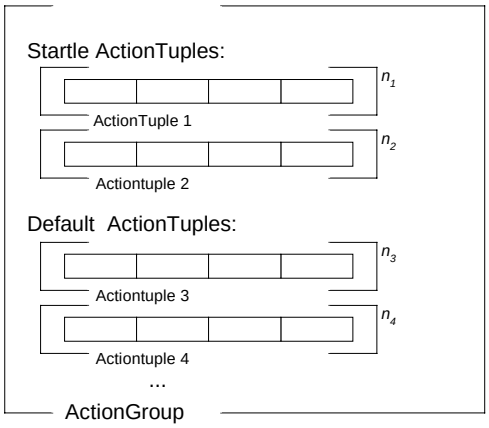

Figure 9: Anatomy of an ActionGroup. Startle ActionTuples become active immediately when their trigger value is non-zero. As suggested by the name, this mechanism is used for reactions like a startle to a sudden, loud noise. ActionTuples on the Default list compete probabilistically on the basis of their e-values.

When ActionTuples are added to an ActionGroup, they may be placed on either the *Startle* or *Default* list. The *Startle* list contains high priority ActionTuples. The ActionTuples on the Startle list compete deterministically: the one with the highest non-zero evaluated value wins. If no ActionTuples on the Startle list are relevant, then the ActionTuples on the Default list are given a chance. ActionTuples on the *Default* list compete probabilistically on the basis of their *e-values*. The use of both a probabilistic and deterministic action-selection mechanism reflects pragmatism as much as anything else. The problem with a purely probabilistic approach is that it is probabilistic. There are times when you really want the character to perform a particular action. Rather than fight probabilities, it is easier to create a separate mechanism for choosing actions that “should just happen.”

However, if an ActionTuple is currently active, then we use the heuristic that it should stay active until its DoUntil condition is met, unless the world has changed significantly from when the decision to activate the ActionTuple was made. This reflects the notion that the DoUntilContext is intended to specify the end condition, assuming nothing unexpected has happened in the meantime. In practice, this means that if an ActionTuple is currently

active, then only if an inactive ActionTuple's e-value has changed significantly from the last time "the dice were rolled" is it eligible to compete. If none pass this test, then the currently active ActionTuple remains active.

In c4's canonical Action System, there are two main ActionGroups. These are (in order of execution):

- *AttentionGroup*: Chooses the creature's focus of attention (things that are large, things that are moving fast, and so on). This decision will often be overridden by later-executed actions.
- *Primary ActionGroup*: The ActionGroup whose actions determine large-scale body motion.

In many creatures, additional ActionGroups are populated with ActionTuples that perform specialized functions. Duncan's Gait Selection ActionGroup uses specialized ActionTuples to perform gait selection using triggers based on internal variables and the results of his canonical ActionGroups.

There are several important advantages to the approach taken for Action Selection in c4. First, it works well in non-deterministic environments, because it is fundamentally probabilistic. The consequences of making a wrong decision are minimal since it is constantly re-evaluating what to do. At the same time, it reduces the amount of re-evaluation that takes place because the ActionGroup "trusts" the ActionTuples. Only if there is a significant change in the world will it force a re-evaluation. Second, the mixed use of a deterministic and probabilistic action-selection mechanism reduces the amount of "float fiddling" that one sometime encounters in purely probabilistic approaches. Third, since navigation is handled by another system, the complexity of the actions in the Action System is greatly reduced. Fourth, the system is general enough to support a hierarchical organization of action, although the need for deep hierarchies is mitigated by the presence of the Navigation System (see below) and indeed by the very structure of ActionTuples. Perhaps the most important advantage is that it supports learning of the sort that animals do, namely (1) learning the consequences of actions, and (2) learning the states of the world in which certain actions are particularly reliable in producing desirable consequences.

Learning in c4's Action System

The approach to learning that is taken in c4 borrows heavily from our understanding of learning in animals as well as from work in machine learning, most notably "reinforcement learning." The fundamental idea behind reinforcement learning is in fact rather simple: *Applying a reward after the occurrence of a response increases its probability of reoccurring, while providing punishment after the response will decrease the probability* [Thorndike 1911]. (See [Sutton 1998] for a more detailed review.)

Certain events in the world are perceived to have an intrinsic value on some nominal scale – for example, eating a cookie has a value of 100, and being attacked by a lion has rather large negative value. When the event occurs – say the dog eats a cookie – the action that immediately preceded the act of eating the cookie should receive *credit* for getting the dog into a state in which a cookie appeared and could be eaten. That is, the value of the action, "sitting," should be adjusted to reflect its likely consequences. Thus, if sitting reliably produces a cookie, the value of sitting should eventually approach 100, subject to a discount factor reflecting the fact that sitting is almost as good as getting a cookie but not quite. This process of assigning credit is called *credit assignment*. The process can be viewed recursively since actions that lead reliably to good things or bad things should also be given their due. In practice it is a bit more complicated than this, because the reliability of a given action in producing a given consequence typically depends on the state of the world, and this is something that must be learned as well. In other words, the system is trying to learn reliable state-action pairs.

It is important to note that it is essentially impossible to learn without variation of action and state. It is only through the process of varying action that the system can learn the relative value of the respective actions. Similarly, it is only through the process of performing actions in different contexts that it is possible to discover those aspects of the world that seem causal to the increased reliability of a given action in producing a desirable consequence. It is very difficult for a purely deterministic system to learn because there is no variation. However, the process of *exploration* must be balanced by *exploitation*. Sometimes the system should decide to go with a known, if possibly sub-optimal solution, instead of experimenting with a novel alternative that might possibly turn out to be a better solution in the long run. Thus, the variability of action needs to be guided by the system. Indeed, in anything but the most trivial of worlds, the state space is huge, and exploring it fully becomes impractical. Thus, careful attention is required to identify those portions of the state space that are most likely to be important. This is a very hard problem in general.

In our system, credit assignment occurs when one ActionTuple becomes active and another becomes inactive. During this process the value of an ActionTuple is adjusted to reflect the likely consequences of performing the

actions of the ActionTuple in the context of the state of the world indicated by its TriggerContext. This is done via a mechanism similar to temporal difference learning (as discussed in [Sutton 1998]).

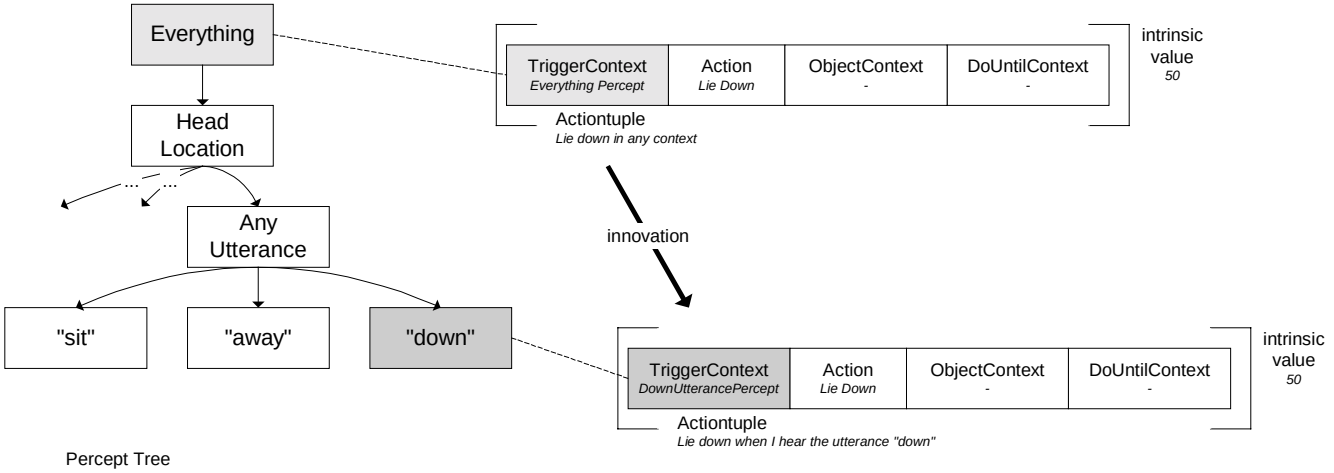

Figure 10: The process of ActionTuple innovation. On the left is the creature's Percept Tree. The AnyUtterancePercept (which serves as a classifier) has spawned a child, "down," that models an acoustic pattern that seems to occur reliably when the Down ActionTuple (top right) is active and subsequently rewarded. The Down ActionTuple is suboptimal in that it causes the creature to lie down in any context. Bottom right is the ActionTuple that is spawned by the ActionSystem to take advantage of the new Percept. Because its trigger is a more reliable predictor, the intrinsic value of that ActionTuple should continue to rise.

As part of the credit assignment process, statistics are kept that correlate the reliability of the ActionTuple with the state of its TriggerContext's Percept children, if they exist. If the ActionTuple's value goes above a threshold and there is a reliable correlation between the ActionTuple's activation and the state of one of its Percept's children, it creates a copy of itself. Then, it sets the TriggerContext to refer to the most reliable child percept, and adds this modified copy of itself to the list of children. Therefore we grow a tree of ActionTuples in which top-level nodes have very general triggers and low-level nodes have very specific ones. This process, illustrated in Figure 10, is referred to as innovation and is similar to the process proposed by Drescher (in [Drescher 91]). When an ActionTuple with children is asked to evaluate its value, it returns the maximum of its own evaluation and that of its children. If the maximum evaluation is associated with one of its children, then that child is the one that becomes eligible for expression. Thus, Duncan can learn that "sitting" in general is a good thing, but "sitting in response to hearing the utterance 'SIT'" is an even better thing.

Credit assignment also guides innovation within the Percept Tree. In this case, children percepts take the form of statistical models built from observations that capture those aspects of the state space that seem correlated with the reliability of a given action. An important observation is that the consequences of the action can be used to discriminate between good and bad examples for the purpose of updating a percept's model. For example, in the case of an UtterancePercept that is building a model of "SIT," if the sit action is rewarded in the presence of an utterance, then it is safe to assume that the utterance must be a good example of "SIT" and so the model can be updated using this new example. If the sit isn't rewarded, then it may be that the utterance wasn't a good example of "SIT." Perhaps it was one of the acoustic patterns that rhyme with "sit," and thus the example should not be used to update the model.

This process of innovation illustrates the intimate coupling between perception and action selection. The creature only bothers to refine senses that ultimately allow it to make better decisions about what to do.

Navigation and Motor System

The final section of a creature's brain is responsible for producing motion. We have divided the various motion-tasks that a creature can perform into two general categories, *Navigational* and *Motor* tasks. As we will see, both categories, and the systems that manage them, make important contributions to a creature's general spatial competence.

Intelligence and motion are also closely coupled in our interpretation of movement. In *The Illusion of Life*, Thomas and Johnson explain there is no such thing as *just* a "walk cycle." [Thomas 1981] There is a happy walk, a sad walk, an excited walk, and so on.[2] This notion – that an animation consists of both a *Verb* and an *Adverb* – is captured in the work of Rose et al [Rose 1999] that inspired our Motor System design.

Navigation System

The deceptively simple act of "eating the food" involves a host of problems: the creature must be near the food, be oriented toward it and, if approaching it is necessary, avoid physical obstacles along the way. The Navigation System provides such large-scale spatial competencies, usually involving *locomotion* around the environment.

The Navigation System typically functions by overriding the motor commands passed down by the Action System. In some cases this command is for an explicit Navigation task, such as "APPROACH." In other cases, the command is directed to the Motor System but with extra approach and orientation conditions specified which the Navigation System must work to satisfy. In either case, the original decision of the Action System is overridden with a more immediately appropriate motor command. If the Action System requests an "APPROACH," the Navigation System might decide that the best way to implement that request is through a "GALLOP." If the Action System requests a "BEG" but the Navigation System is instructed to orient the creature first, that command might be replaced with a "TURN." The "BEG" will continue to be overridden until the orientation condition is satisfied.

The Navigation System allows a convenient level of representation in the Action System, because it relieves the Action System of the burden of implementing the decisions it makes. "Approaching" as an essentially physical behavior may indeed precede each "eating," but behaviorally, both should be part of a single "eating" act – especially from the point of view of any learning that takes place. Ultimately, the majority of animal behaviors follow the "approach, orient and do" model, and the Navigation System allows these behaviors to be represented with high-level atoms.

Motor System

Motor systems and action selection mechanisms are closely coupled in nature. Real creatures display a remarkable "common sense" intelligence concerning their bodies and their movement – a sense that synthetic characters do not, at present, achieve. Without commensurate advances in this area, our virtual creatures risk appearing unintelligent, regardless of the structure and sophistication of their behavior. Before we discuss how to build motor systems for these characters, let's consider the kinds of motor problems we wish our characters to be able to solve.

Walking, Shaking and Moving your Head

The basic competencies of an example-based motor system clearly include such "simplicities" as:

- *Simple gestures.* Characters should be able to perform actions based on animations provided to them by human animators, without unrealistic or highly noticeable discontinuities.
- *Locomotion.* Gross body movement – getting around the world – should be supported.
- *Eye / head / body orientation.* The character should appear to attend to things in the world. Attention is essential to expressivity and the appearance of awareness.

The first of these basic competencies is achieved by the careful replaying of "canned" animations. The remaining two call for something more complex.

The task of moving around at the very least requires *coordination* of animations (for example, turn-left, move forward, move forward, stop) to achieve a particular *goal*. As described above, c4 passes much of this burden off to a separate Navigation System. But the problem of orienting the head towards something demands the creation of a *continuous* output animation space from necessarily discrete source animations.

Each of these competencies has a behavioral component that we can make arbitrarily complex. For example, moving around the virtual world might involve learning spatial maps of the environment, performing collision

[2] If *Illusion of Life* makes clear what one must do to bring a character to life, it is Dennett's intentional stance that explains how Disney's techniques work. The technique described by Thomas and Johnson is essentially a recipe for allowing the viewer to take a rewarding and consistent intentional stance towards a character.

detection and obstacle avoidance. But without a motor system that can move the creature around, this "intelligence" cannot come to pass.

Doing it in Style

But the problem is harder than this: throughout this motion, the character has to keep *in character.* By creating a motor system that keeps close to the source example animations, one will for short periods of time be guaranteed success. But on timescales longer than the length of an animation, this "illusion of life" will break down as the character fails to interact correctly with the environment, fails to attend to things properly and eventually repeats the same animation over and over.

Further, we've already mentioned that the "correct" style of motion may change over time, creating perhaps a slow, or sad, or even a hungry walk in response to the character's internal state. If we cannot achieve these things then we cannot have expressive characters. Therefore we add the following to our wish list:

- *Parameterized motor actions.* The Motor System allows us to produce parameterized motor actions, such as "shake paw high" and "shake paw low." This will work best if we can produce a *continuous space* of shake-paw animations that cover a variety of heights.
- *Support new motor actions.* One of the goals of the behavior system described above is to allow our characters to increase in complexity over time. Although this can often be achieved by reusing already present motor actions, this obviously does not exhaust all the learning possibilities.
- *Create new animations "live."* Provide support for the perceptual models and the output mechanisms of a character to learn new animations during their lives. *Let the Action System understand the body.*

Solve problems

This last point suggests opening up the "contents" of the body to the introspection of the Action System so that we can construct new models of movement. But in many cases we'd like the communication between the Action System and Motor System to occupy *less* "bandwidth," not more. In addition, there is a practical need to shield the authorship process of the Action System from the details of the Motor System content to allow the development of both to take place in parallel.

One goal in this work is to make communication between the Action System and the Motor System take place in terms of *desired pose* descriptions. This could be the Action System saying, "I'd like to walk over there now." It could be in terms of *end-effectors:* "Put my nose near the food. Is it there yet?" Finally, it could be in terms of time: "How long, roughly, before I could get my nose there?" At the very least, this has the very desirable effect of shielding the behavior of a character from changes in the competence and content of the Motor System. So:

- *Communicate in terms of end-effectors.* Can the Action System communicate with the Motor System in terms like "get my mouth to the bone?"
- *Maintain the illusion of (cartoon) physics.* Since this is a virtual world, we are free to do anything we want to alter the position and angles of a character's body. There is a price to this freedom. We typically want characters to fall back to earth when they jump, their joints to resist turning complete rotations, and their feet to stay on the floor rather than under it. These things don't come for free.

Verb Graphs

Motor systems today typically possess a number of these basic competencies: They can play out animations like a walk cycle onto a creature's body on command. They can layer animations, such as a hand wave atop that walk cycle. They can blend animations, for example "turn left" and "walk forward," to produce an intermediately turning walk cycle. One canonical solution to these problems is the Verb / Adverb model of Rose (see [Rose 1999]) and the motor systems of Perlin (see [Perlin 1996]). Rose introduces the *Verb Graph* – a structure in which nodes represent hand-made animations (Verbs) and the edges represent allowed transitions between Verbs. The Verb Graph in this way represents some of the basic physical and continuity constraints of having a body.

C4's default motor system is a Verb/Adverb Motor System. Such motor systems do a good job of playing simple gestures, moving the character around in the world, and providing a continuous variety of motor actions.

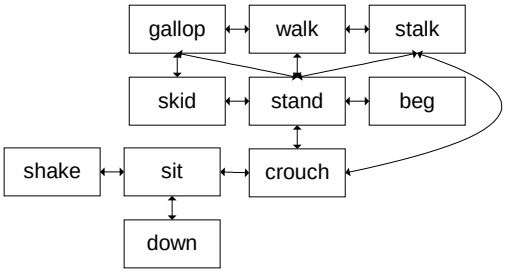

Figure 11: Part of Duncan's motor system verb graph. Nodes represent verbs, and edges represent allowed transitions between verbs. Thus if Duncan was crouching and wanted to gallop, he would transition from crouch to stand, then transition from stand to gallop. Each node may represent any number of hand-made animations arranged in an Adverb space.

However, such a motor system is insufficient to solve all the motor problems posed above. In particular, nothing in such a motor system suggests *how* to blend animations in order to achieve a particular goal; the complete animation seems too large an atom to base either movement or learning on; and it is hard to see where and what kind of "new material" could be placed in a verb-graph structure.

Pose Graphs

To address these problems, we have extended the idea of a verb-graph, increasing its resolution from the level of "animation" to the level of pose. The contents of these *pose-graphs* are still based on source animation material. However, this source material is broken down into poses that can be annotated and associated with pre-computed information. The current implementation supports the formation of multi-resolution graphs with nodes built, perhaps dynamically, from blends of other nodes. Paths through such structures that take the body from pose to pose can be efficiently found, and animations reformed in real-time.

An experimental pose graph motor system (described in detail in [Downie 2001]) has been integrated into c4 to achieve a number of our motor problem goals, such as the ability to generate paths through the graph on demand. It provides a channel of communication to the Action System that can be as abstract or as detailed as the creature designer desires – the kind of information stored in the internal blackboard's MOTOR_DESIRED entry is flexibly interpreted. Actions can communicate in terms of "end effector" positions ("get my nose close to the food") and the motor system can quickly work out how to get to the pose that best solves the problem. The Action System may also be more explicit, calling for specific poses (like "sitting") to be reached, or animations to be played. Finally, the behavior system may inspect the pose transitions from the pose-graph and begin to build models of those sequences in response to reward from the environment. In short, creatures can learn new ways of putting together old animation material. All of this is done using an algorithm that runs at an interactive rate.

Pose-graph motor systems can offer the variability and the plasticity required for compelling, adaptive creatures, while remaining an example-based technique - extending the usefulness of the handcrafted animation into an interactive domain.

Conclusions: Beyond "AI From Day One"

Given the amount of coverage the topic has received recently in Game Developer Magazine, the notion that "AI shouldn't be an afterthought" has become a platitude. We take a step further. Almost every decision affecting the design of an architecture that will support autonomous creatures should be made with the AI in mind.

In Disney's terms, it is the life that gives meaning to the expression. In Dennett's, it is the ability to infer a creature's beliefs and desires that makes the creature appear to have intentions. It is the interactions between the participant, the virtual characters, and their shared environment that make an experience compelling. This leaves us with an intriguing (if daunting) challenge that we have only begun to address with c4's architecture.

The first part of the challenge is *recognizing which architectural features provide the creatures with the ability to have and express beliefs and desires*. The second part of the challenge is determining *how to implement those features in a flexible, extensible way*.

Expressivity is key

Note that the creature must both *have* and *express* beliefs and desires. Adding a feature to a synthetic character (such as an emotional axis) is pointless if the character lacks the means to express that feature. Similarly, adding a feature to a virtual world is pointless if creatures in that world can't interact with the feature. The things that an autonomous creature can't interact with seem broken, or cause the artificial intelligence to seem stupid. We have discussed some of these decisions – perceptual honesty, animations that use verbs and adverbs, and so on – that have helped c4 further a creature's illusion of life.

Simulation and Mental Models

The strict split that c4 enforces between simulation and mental representations demands that each creature make decisions not on the basis of an external object's state, but on the basis of the creature's *view* of that state. We have found the benefits of this representational division to greatly outweigh the costs of additional system memory and bookkeeping.

Without this division, we would not be able to implement the "sensory honesty" layer provided by the Sensory System. In a sense, the representational division *is* the sensory honesty. Because creatures cannot "cheat" by accessing the brains of other creatures, they are required to base their decisions on their own world view. As an added bonus, we can add and remove creatures from worlds without fear of leaving dangling pointers to other creatures.

Most importantly, by divorcing these two representations, many "second-level" cognitive effects become possible, most of which arise from situations in which the creature's mental model fails to match the world model. In other words, creatures make "honest" mistakes. These effects include mistaken identity, surprise, confusion and the ability to be teased. Paradoxically, these "mistakes" can add greatly to a creature's realism.

Variety is the Spice of A-Life

The many kinds of problems that animals (and interactive characters) can solve entail many kinds of solutions. In order to be a viable platform for intelligence in all its forms, an architecture like c4 needs to support and encourage this heterogeneity. Each system in a brain built with c4 uses representations appropriate for the problems it solves. When needed, new systems can be added. Variety, far from leading to a hodge-podge design, contributes to a system's robustness and generality.

Intuitive Behavior Design

Believable behavior involves many details: the creature's gaze and physical positioning must be controlled as appropriate for the active behavior, the creature's physical emotion-layers must reflect an attitude toward the behavior or the behavior's target, and so on. It is important that creature designers have access to mechanisms that make it easy to specify behavior.

One strength of c4's architecture is its ability to handle many of these details automatically. Most behaviors follow the same general model of "pick a target, approach it, and do something," and much infrastructure is provided to support this common case. For example, the OBJECT_OF_ATTENTION posting directs the creature's gaze, the Navigation System provides approach and orientation capabilities, and the various layers of the Motor System control expressive movement. It remains for the designer to fill in a few critical components of the behavior, such as specifying triggers (when should the behavior be active?) and ObjectContexts (what should a behavior act on?).

When a behavior fails to adhere to the common-case model, the appropriate system can be modified as needed – perhaps the Perception System needs a percept to detect a new type of world event, or perhaps the Motor System needs to perform a new kind of physical action. Whatever the case, the system conspires to make *simple things simple and complex things possible*.

Supersumption

Two architectural themes are seen throughout c4. The first, reminiscent of Rodney Brooks' *subsumption*, occurs when high-level systems send control signals to low-level systems in order to change their behavior. (Subsumption, and the core of Brooks' philosophy circa 1991, is explained in detail in [Brooks 1991].) The Action System, for example, outputs tokens that instruct the Motor System to perform specific physical actions. (In this case "high-level" and "low-level" are not intended to reflect "degree of sophistication").

Another technique for control is when the decisions of high-level systems, which have a good general idea of *what* to do, are overridden by specialized low-level systems that have specific knowledge of *how* to do it. This is a technique we call *supersumption*.

The Navigation System provides an example. Though the Action System sends tokens to the Motor System, other signals are sent to the Navigation System, such as requests to approach and orient toward a target. Based on these signals, the Navigation System may decide to override the output of the Action System. For example, suppose the Action System requests a "BEG," but the dog is facing away from the shepherd. The Navigation System will replace the "BEG" with a "TURN" for as long as it takes for the dog to orient itself toward the shepherd. During the time Motor System receives its "TURN" command, it does not know whether its instructions came from the Navigation or Action System, and the Action System is decoupled from the details of the contents of the Motor System.

Supersumption is also seen within the Action System, where an AttentionGroup may pick an object of attention, only to have its decision overridden by the later-executed Primary ActionGroup. Since the Primary ActionGroup controls overall physical behaviors, the ActionTuple it activates might be better equipped to decide what the focus of attention should be.

Learning the right things at the right levels

The levels of abstraction provided by the Perception System, Action System, Navigation System and Motor System provide the potential for learning different things at different levels in the brain. Fundamentally, dividing learning problems between systems makes learning easier.

The scheme used to back-propagate value in the Action System would be far less efficient if that system had to precede every "eat" with an "approach." The creature would end up with the idea that since "approach" often leads to "eat," the act of "approaching" must inherently be a good thing. By allowing some basic spatial ability to be included implicitly in the "eat" behavior, the creature recognizes the process of navigation and orientation toward food as part of the overall experience of eating.

Likewise, if a more sophisticated Navigation System were designed to *learn* basic spatial abilities, that navigation learning could take place in parallel with the Action System's high-level behavioral learning, internally assigning its own reward or punishment for its navigational successes. If the creature very accurately navigates toward a lion, its adaptive Navigation System should register a positive result, even though the creature may realize very quickly that, as a general survival strategy, approaching lions is a bad idea.

Lessons from *sheep|dog* and *Clicker*: Doing it right makes it easier!

sheep|dog was created to demonstrate some of the basic abilities of creatures implemented with c4, and also serve as a stress test for the system's engineering. It features two creatures with full brains (Duncan and the shepherd), six sheep, and obstacles that react to the presence of the creatures. The project also features distributed rendering, with two clones running subsets of c4 rendering the same world from different views.

The learning algorithms being developed by our group were put to use in *Clicker*, in which a user can train Duncan using "clicker training," an actual dog-training technique in which behaviors are "marked" (by a salient click sound) and then reinforced with food reward (See [Yoon 2000] for more on training synthetic characters). In this simulation, Duncan can be trained to associate vocal commands with behaviors, and demonstrates a number of the phenomena that one sees in real dog training, such as Thorndike's Law of Effect, shaping, resistance to extinction, and so on. ([Shettleworth 1998] provides an excellent summary of these phenomena.)

Duncan features the most complex Perception, Action and Navigation systems of any of the creatures. In contrast to the autonomous raccoon of the *Swamped* project, which consisted of over 80 pages of a VRML-like data structure, Duncan is constructed in about 10 pages of easily understood Java code. The use of the right representations at the right levels made it relatively easy for a creature designer to implement complex behaviors like Figure 8's "circle clockwise around a sheep when the shepherd says, 'bye!'" Duncan's Sensory System filters external input and represents it in his coordinate frame, providing the rest of his brain with a notion of "clockwise." The results of Duncan's Perception System, structured as PerceptMemory objects in his Working Memory, allow him to track a specific sheep. Duncan's Action System recognizes that in the current context (he heard "bye"), he should select a circling action and a target sheep. His Navigation System takes care of how he should get there. Finally, his Motor System blends layers of verbs and adverbs to produce expressive animations.

The shepherd is a “semi-autonomous” character who responds to the voice input of the participant. An example of heterogeneity in the architecture is found in the shepherd’s Perception System, where a special percept that contains the group’s acoustic pattern recognition research allows the shepherd to classify utterances as one of six possible commands. The classifier can be trained through a “one-shot learning” interface so that new users can achieve a high recognition rate after a very short (about 15 seconds) training routine. One of the tasks performed by the shepherd’s Action System is posting the utterances he hears to the world as DataRecords so that other creatures perceive those utterances as coming from the shepherd. Using a layer in his motor system, the shepherd uses hand gestures to indicate his interpretation of the participant’s utterance. Because Duncan does not always respond to a command immediately, this provides the user with immediate feedback that the system heard and interpreted their utterance correctly. A final example of flexibility provided by the architecture was our ability to experiment with three control mechanisms for the shepherd’s navigation, each of which would allow the user some control over his position on the field, automatically helping the shepherd avoid obstacles and taking back control when the shepherd is in danger of being rammed by a sheep.

Finally, the sheep use a series of specialized ActionTuples that cause them to avoid Duncan while exhibiting flocking behavior using Reynolds’ BOIDS algorithm [Reynolds 1987]. BOIDS, which uses local area effects to align the sheep with their neighbors, was easily implemented under c4. Again, the abstractions provided by the Sensory, Perception, Action and Motor Systems made it easy to implement this unique behavior.

Future work

c4 continues to be a work in progress. Work over the next year will continue to emphasize behavioral adaptation and motor learning. We are also exploring how to model development (physical and mental) and social behavior. The authors are integrating models of hippocampal spatial learning, as well as time/rate learning inspired by [Gallistel 2000]. All of this research will be integrated into our next project, AlphaWolf, in which c4 will serve as the behavioral engine for a simulated wolf pack.

Related Work

Our work borrows heavily from the impressive work that has come before. Our ideas about super- and subsumption follow from Brooks’ work (e.g., [Brooks 1991]) as does our emphasis on building the whole creature. Our emphasis on the value of taking an ethologically inspired approach follows from [Reynolds 1987, Tu 1994, Blumberg 1995, Yoon 2000] whose behavior systems were inspired by the behavioral models of ethologists such as Tinbergen and Lorenz. Our representation of action sits between that typically used in reactive systems and that used in planning systems. As such it borrows from the work of Maes and Firby [Maes 1990, Firby 1992] and in its emphasis on choosing a few good representations, Minsky [Minsky 1985]. Our approach to learning borrows ideas from traditional reinforcement learning [Ballard 1997 for a review], animal psychology [Gallistel 2000], and dog training [Gary Wilkes, personal communication]. While our system does not do “cognitive modeling” as proposed by Funge [Funge 1999], the system described could easily be integrated into Funge’s architecture. Our motor system design borrows heavily from the ideas of Rose [Rose 1999] and Perlin [Perlin 1996].

Acknowledgements

We acknowledge and thank all the members of the Synthetic Characters group for their contributions to this research: Michael Patrick Johnson, Bill Tomlinson, Scott Eaton, Ben Resner, Matt Berlin, Jesse Grey and Dan Stiehl.

References

[Ballard 1997] Ballard, D., An Introduction to Natural Computation, MIT Press, Cambridge 1997.

[Brooks 1991] Brooks, R., Intelligence Without Reason, "Computers and Thought" IJCAI 1991

[Blumberg et al. 1995] Blumberg, B.M., Gaylean, T., Multi-level Direction of Autonomous Creatures for Real-Time Virtual Environments, *SIGGRAPH 95 Conference Proceedings*, 1995

[Dennett 1987] Dennett, D. 1987. The Intentional Stance. Cambridge, Mass.: MIT Press.

[Downie 2001] Downie. M. Behavior, Animation, Music. The Movement and Music of Synthetic Characters. Unpublished S.M. Thesis, *MIT Media Lab*, 2001.

[Drescher 1991] Drescher, G.L. 1991 Made-Up Minds:A Constructivist Approach to Artificial Intelligence, Cambridge, Mass.: MIT Press.

[Eberly 2001] Eberly, David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. Academic Press, 2001.

[Firby 1992] Building Symbolic Primitives with Continuous Control Routines. in: *Proceedings of the First International Conference on AI Planning Systems,* College Park MD, June 1992, pp 62-69.

[Funge et al. 1999] Funge, J., Tu, X., Terzopolous, D., Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Characters, SIGGRAPH 99 Conference Proceedings, 1999

[Gallistel et al. 2000] Gallistel, C. R., & Gibbon, J. Time, rate and conditioning. Psychological Review, 107, pp289-344.

[Johnson 1999] Johnson, M.P, A. Wilson, B. Blumberg, C. Kline, and A. Bobick. *Sympathetic Interfaces: Using a Plush Toy to Direct Synthetic Characters.* In Proceedings of CHI 99.

[Kline 1999] Kline, C., Observation-based Expectation Generation and Response for Behavior-based Artificial Creatures, Unpublished S.M. Thesis, *MIT Media Lab*, 1999

[Laird 2000] Laird, John E. It Knows What You're Going to Do: Adding Anticipation to a Quakebot in: *AAAI 2000 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment,* March 2000.

[Maes 1990] Maes, P. "Situated Agents can have Goals", Robotics and Autonomous Systems, Vol 6.

[Minsky 1985] Minsky, M., Society of Mind, Simon & Schuster, New York 1985

[Perlin et al. 1996] Perlin, K., Goldberg, A., Improv: A System for Scripting Interactive Actors in Virtual Worlds, SIGGRAPH 1996

[Reisberg 1997] Reisberg, Daniel. Cognition: Exploring the Science of the Mind. W.W. Norton & Company, 1997.

[Reynolds 1987] Reynolds, C. W., Flocks, Herds, and Schools: A Distributed Behavioral Model, *SIGGRAPH 87 Conference Proceedings*, 1987

[Rose et al. 1999] Rose, C.F., Cohen, M., Bodenheimer, B., Verbs and Adverbs: Multidimensional Motion Interpolation - *IEEE Computer Graphics And Applications*, Volume 18, Number 5, 1999

[Rosenbloom et al. 1993] Rosenbloom, Paul S., Laird, John E., and Newell, Allen. The Soar Papers: Research on Artificial Intelligence. MIT Press, 1993.

[Russell 1980] A circumplex model of affect. *Journal of Personality and Social Psychology,* 29:1161-1178.

[Shettleworth 1998] Shettleworth, Sara J. 1998. Cognition, Evolution and Behavior. New York: Oxford University Press.

[Sutton 1998] Sutton, R. and Barto, A. Reinforcement Learning: An Introduction. MIT Press: 1998.

[Thomas 1981] Thomas, F. and O. Johnson. 1981. *The Illusion of Life: Disney Animation.* New York: Hyperion.

[Thorndike 1911] Thorndike, E. Animal Intelligence. Hafner, Darien, CT, 1911.

[Tu et al. 1993] Tu, X., Terzopoulos, D., Artificial Fishes: Physics, Locomotion, Perception, Behavior, SIGGRAPH 1993

[Watt et al. 1992] Watt. A. and M. Watt. Advanced Animation and Rendering Techniques: Theory and Practice. ACM Press, 1992

[Wilkes 1995] Wilkes, G. 1995. Click and Treat Training Kit Version 1.2. Mesa, AZ.: www.clickandtreat.com.

[Yoon et al. 2000] Yoon, S., Blumberg, B.M., Schneider, G.E., Motivation-Driven Learning for Interactive Synthetic Characters, *Autonomous Agents 2000 Conference Proceedings*, 2000

Publications of the Synthetic Characters group can be found online at http://characters.www.media.mit.edu/groups/characters/