A Layered Brain Architecture for Synthetic Creatures

Damian Isla **Robert Burke** **Marc Downie** **Bruce Blumberg**
naimad@media.mit.edu rob@media.mit.edu marcd@media.mit.edu bruce@media.mit.edu

The Media Laboratory
Massachusetts Institute of Technology
20 Ames St.
Cambridge, MA 02139

Abstract

This paper describes a new layered brain architecture for simulated autonomous and semi-autonomous creatures that inhabit graphical worlds. The main feature of the brain is its division into distinct systems, which communicate through common access to an internal mental blackboard. The brain was designed to encourage experimentation with various systems and architectures. It has so far proven flexible enough to accommodate research advancing in a number of different directions by a small team of researchers.

1.0 Introduction

One approach to designing intelligent systems is to look to existing biological systems for clues and design principles. This paper describes C4, the latest in a series of brain architectures built on this principle by our group. This architecture is used for simulating the behavior of agents, or *creatures*, that inhabit a graphical world. These creatures are able to sense their environment, learn appropriate actions based on expectation of external reward and navigate their environment. C4 sets up a framework to support these and many other mental abilities. The content and structure of this framework are inspired by the abilities of real animals and attempt to deal with some of the constraints that they face.

Some of the goals for the system 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 network rendering. 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, the architecture needed to facilitate the construction and control of synthetic creatures. Intelligence is often seen as the confluent effect of many individually unintelligent components (as in [Minsky 1985]) and the architecture needed not only to support those components individually but also to allow them to coexist and communicate coherently within one brain. Therefore the primary goal was to build a system that facilitated:

Figure 1: Duncan the Highland Terrier

- Reactive behavior: it should be easy to design and implement the kind of reactive behavior that many previous works in the field of autonomous agents support. [Brooks 1991, Tu et al 1993, Blumberg 1995, Perlin 1996, Yoon 2000];
- Learning: creatures should adapt their behavior based on reward and punishment feedback from the world;
- Extensibility: The architecture should be easily extensible in order to support research in various different directions by different researchers.

The result is a highly modular architecture with few essential subsystems and many opportunities for expansion. The canonical brain includes an internal blackboard, Sensory and Perception Systems, Working Memory (a short-term memory model), and Action, Navigation and Motor Systems.

We have implemented two significant projects to date with C4. One project is *sheep|dog* (**Figure 1**), an interactive installation piece in which a user plays the role of a shepherd who must interact through a series of vocal commands with Duncan, a virtual sheepdog, to herd a flock of sheep. This system demonstrated some of the basic reactive, perceptual and spatial abilities of the creatures built under 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 to train real dogs.

The paper proceeds as follows: Section 2.0 will outline the world model being used; section 3.0 will delve into the architecture of the brain itself, and the make-up and function of the various systems that comprise C4; section 4.0 will present the results of building C4 and the installations that use it; section 5.0 will present a discussion of various successful design elements of C4 and sections 6.0 and 7.0 will describe some future and related work.

2.0 World Model

A formal abstraction exists between the agent and the world. The World model's primary function is to maintain the list of creatures and objects and to act as an event blackboard for the posting and distribution of world events. It also coordinates network synchronization and manages rendering.

World events take the form of *DataRecords*, perceptual nuggets that can be processed by a creature's Sensory and Perception Systems. A DataRecord can represent anything from an acoustic pattern (Sheep|Dog allowed the user to "speak" to the dog through a microphone interface) to a visual or symbolic event. Whether a specific form of DataRecord can be interpreted depends entirely on the sensory and perceptual abilities of the sensing creature.

In a single execution of the update loop, events are fed to each of the creatures and objects and each is prompted to update itself (this process can happen in parallel). 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.

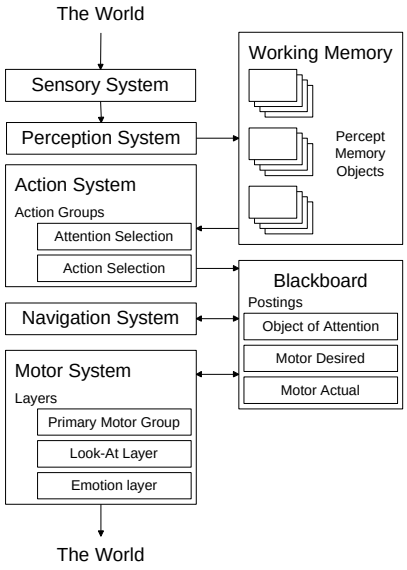

Figure 2: The brain architecture

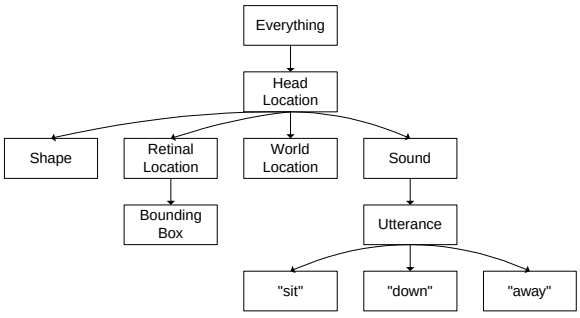

Figure 3: The Percept Tree. The Head Location Percept is derived from the root of the tree because, at present, all events have locality. Retinal location and shape only apply to visual events, just as an event can only be classified as an utterance or a specific utterance if it is an auditory event.

3.0 Brain Overview

Figure 2 shows the layout of a typical creature's brain. 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 (differentiated by string labels).

A detailed description of each system follows.

3.1 Sensory System

Physical intelligent systems (biological or otherwise) by necessity feature a set of well-defined sensors through which information about world-state and world-events pass. We protect the integrity of this world-agent division by making use of a Sensory System abstraction.

The Sensory System is a filter through which all world-events (represented by DataRecords) must pass. Unlike its physical equivalents, where any data that passes through is fair game, in C4 the Sensory System plays an active role in keeping the virtual creature's virtual sensation honest. In a simulated world, there is potentially much more accessible information than the creature, limited by its sensory apparatus, should be able to sense. For example, though Duncan the sheepdog *could* be given the location of the sheep that is behind him, he *shouldn't* be given it. While often the role of the Sensory System is to filter out data that can not be sensed, other times its role is to transform it. For example, it converts visual location information into the local space of the sensing creature (Duncan receives all location information from the world in the coordinate frame of his left eye) or in attenuating the intensity of a sound or acoustic event proportionally to the square of the distance from the source. It is because the same piece of data will be viewed differently by each creature in the world (and because sometimes we want some creatures to be able to cheat and not others) that the individual creature's Sensory System performs the event filtering rather than the world itself.

Perceptual honesty is an important theme for our research, since we feel that more honest perception leads to more believable behavior. More importantly, creatures in the real world are able to make generally good decisions despite noisy, unreliable and occasionally entirely *missing*

data. This is the point of Brooks' insistence on *situatedness* [Brooks 1991]. While we do not claim that our virtual world holds anything close to the complexity of a physical environment, the Sensory System allows us to begin to face some of the problems like noisy sensation and McCallum's *perceptual aliasing* [McCallum 1995].

3.2 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 may "sense" an acoustic event, but it is up to the Perception System to recognize and process the event as something that has meaning to the creature. Thus, it is within the Perception System that "meaning" is assigned to events in the world.

The Perception System takes the form of a *Percept Tree* (**Figure 3**). A *Percept* is an atomic classification and data extraction unit that models some aspect of the sensory inputs passed in by the Sensory System. Given a DataRecord it returns both a match probability (the SheepShapePercept will return the probability that a DataRecord represents the experience of seeing a sheep) and, if the match is above a threshold, a piece of extracted data (such as body-space coordinates of the sheep). The details of how the confidence is computed and what exact data is extracted are left to the individual percept. The percept structure might encapsulate a neural net or it might encapsulate a simple "if … then …else" clause. This freedom of form is one of the keys to making the C4 Perception System extensible, since the system makes no assumptions about what a percept will detect, what type of data it will extract or how it will be implemented.

Percepts are organized hierarchically in terms of their specificity. For example, 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 SheepShapePercept). The children of a percept will receive only the data that was extracted by its parent to process. This hierarchical structure is primarily an efficiency mechanism (no point in testing whether an event is the spoken word "sit" if it has already been determined that the event was not an acoustic one) and is very similar to previous hierarchy-of-sensors approaches.

Many percepts are plastic, 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, in concert with the Action System, can modify the topology of the tree itself, dynamically growing a hierarchy of children in a process called *innovation*. As will be described in section 3.5, the process of innovation is reward-driven, with only percepts that are believed to be correlated with increasing the reliability of an action in producing a desirable outcome being prompted to innovate.

Both the confidence and the extracted data of every percept are cached in a *PerceptMemory* object. When a DataRecord enters the Perception System, a new PerceptMemory is created, and as the DataRecord is pushed through the Percept Tree, each percept that registers a positive match adds its respective data to the new PerceptMemory. Thus, given a sensory stimulus, the PerceptMemory represents all the creature can *know* about that stimulus.

3.3 Working Memory

Like other agent-control architectures (e.g. [Rosenbloom et al. 1993]), C4 contains a *Working Memory* structure whose function mirrors that of the pyschological conception of Working Memory – an object-based memory that contains information about the immediate task or context. The ultimate goal of Working Memory is to provide a sensory history of objects in the world. It is on the basis of these histories that action-decisions will be made, internal credit for reward assigned, motor-movements modulated, etc. In C4, Working Memory is a repository for persistent PerceptMemory objects. Taken together, they constitute the creature's "view" of the world.

The PerceptMemory is itself a useful structure. By caching together the various perceptual impressions made by a world event ("the thing that was in front of me was also blue," blueness and relative location being separate Percepts) they solve (or perhaps avoid) the infamous *perceptual binding problem* ([Treisman 1998]). They also allow us to submit complex queries to WorkingMemory: "which is the sheep that is nearest me?"

PerceptMemory objects become even more useful when they incorporate a time dimension with the data they contain. On any one timestep, the PerceptMemory objects that come out of the Perception System will by necessity only contain information gathered in that timestep. However, as events often extend through time, it is possible to *match* PerceptMemory objects from previous timesteps. Thus a recent visual event may represent only the latest sighting of an object that we have been tracking for some time. 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 (e.g. distinguishing between two sheep). Location matching is also used to combine perception information of different modalities, for example a visual event and an auditory one that originate from the same location. Assuming that 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 percept data is not observed, the confidence in the existing data is decayed. The rate of decay is in part determined by the percept. Roughly, the rate of decay should be proportional to the observed variability of the data.

3.4 Prediction and Surprise

The "view" that Working Memory provides of the world can be informed by more than just direct perception. Much like our own perception, where unobserved data is subconsciously "filled in" by low level predictions and assumptions, creatures implemented under C4 can be designed to act upon events that are likely to occur, or qualities that objects are likely to have. We believe that the ability to recognize temporal patterns and act on them is an essential component of common sense intelligence.

Sometimes prediction is used not to predict the future but simply to maintain a coherent view of the present. The stream of sensory data coming from an object can easily be interrupted – for example, because the object is out of the creature's visual field, or because it is occluded by another object. In these cases, prediction can allow a creature to maintain a reasonable estimate of where the object is, even though it is not being observed.

The actual mechanisms of prediction can take many forms. Since PerceptMemory objects contain histories of percept data, it is possible, if the data are vector or scalar, to use function approximation techniques to extrapolate values. In more complex cases, periodic behaviors or percept-activity correlations (the bell always rings before the food appears – classical conditioning, essentially) can be recognized and exploited. These prediction-mechanisms could conceivably extend to common sense knowledge about the world – if an object is suspended with no visible support, it might be predicted to fall.

The occasional deviation of predictions 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 which has just provided a surprising stimulus is an excellent candidate for the creature's object of attention. [Kline 1999] includes an excellent discussion of expectation and surprise in synthetic characters. C4 does not currently make use of surprise.

3.5 Action System

Given a set of PerceptMemory objects that detail the perceived state of the world, the creature must decide what action(s) it is appropriate to perform. Many types of decision-making processes are possible here, however, this section will discuss the action system that we have implemented in C4.

3.5.1 ActionTuples

Any Action-representation must address a number of fundamental questions, namely:

- What do I do?
- When do I do it?
- What do I do it *to*?
- How long do I do it for?
- What is it worth?

Our representation of action, the *ActionTuple*, directly addresses these five questions through its five subcomponents:

Primitive Action(s) (what do I do?): A piece of code that actually executes the action in question. An ActionTuple's set of primitive actions typically modify the contents of the blackboard. These postings in turn 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.

TriggerContext (when do I do it?): A piece of code that returns a scalar value representing the relevance of an ActionTuple given the current state of Working Memory. Triggers are typically references to percepts in the Percept Tree (a trigger that points to the "Sheep Shape" percept will return a high relevance given any PerceptMemory that has a high "Sheep Shape" confidence). However, the TriggerContext is general enough that more complex trigger-conditions can be hand-crafted. As we will see, Percept-based triggers are useful because they can be automatically generated through the learning process.

ObjectContext (what do I do it to?): A piece of code that chooses a target for the action. Again, it is often defined in terms of percepts ("perform the action on something that is sheep-shaped AND blue"). When an ActionTuple is active, the ObjectContext posts the PerceptMemory chosen into the OBJECT_OF_ATTENTION posting of the internal blackboard, thereby making it available to the rest of the system. The ObjectContext is an optional component, since not all actions are necessarily targeted.

DoUntilContext (how long do I do it for?): A piece of code that returns a scalar representing the continuing relevance of an ActionTuple while it is active. This could take the form of a timer, which drops to zero after a specific period of time, or some code that looks for more complicated ending conditions in Working Memory.

Intrinsic Value (what is it worth?): ActionTuples are ascribed an *intrinsic value*, which is an indicator of how generally "good" the ActionTuple is. This is similar to the Q-value in Q-learning (see [Ballard 1997]). This value can be used to bias the Action-Selection process and can be modified through learning (see below).

3.5.2 Action-Selection

The intrinsic value and the relevance (as determined by the Trigger or the DoUntil contexts) can be combined into a single *evaluated value* corresponding to the amount of reward expected to result from performing an action. ActionTuples can then compete for expression on the basis of this evaluated value.

ActionTuples are grouped into *ActionGroups* that are responsible for deciding at each moment which single ActionTuple will execute. Each ActionGroup can have a unique Action-Selection scheme, but the most common scheme is described below.

When ActionTuples are added to an ActionGroup they may be placed on either the *Startle* or the *Default* tuple list. The *Startle* list contains high priority ActionTuples that compete deterministically, i.e. the one with the highest non-zero evaluated-value wins. If no ActionTuple on the Startle list is relevant, the ActionTuples on the Default list are allowed to compete probabilistically for expression on the basis of their *e-values*.

If an ActionTuple is active, it is generally allowed to stay active until its DoUntil condition is met. When it is met, the selection process takes place again. There are two cases in which an active ActionTuple can be interrupted: if a StartleTuple becomes relevant (or more relevant than the current one, if the current one is also a startle) or if the world changes significantly. This change is measured in terms of the evaluated values of inactive tuples: if an inactive ActionTuple's evaluated value has more than doubled recently, then another probabilistic tuple selection takes place between that tuple and the current one.

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 (e.g. things that are large, things that are moving fast, etc.). This decision will often be overridden by later-executed actions.
- *Primary ActionGroup*: The ActionGroup whose actions determine large-scale body motion.

3.5.3 Learning in the Action System

Learning is an important focus of our research, particularly the kind of learning that is observed in animals. The Action System implements three types of learning that together allow creatures to be trained in a manner similar to that used to train real dogs ([Wilkes, personal communication]).

Credit Assignment: When a new ActionTuple with a high intrinsic value is activated (for example, the startle tuple "eat", which can only be run in the presence of food) we wish to give credit to the action that led to that tuple being activated. This back-propagation of value scheme is very similar to that seen in temporal difference learning [Sutton 1998]. However, unlike classical temporal difference learning, it is not always the tuple that actually ran that is given credit – it can be a tuple with a similar primitive action list with a more specific trigger that was relevant at the time. This choice is based on a combination of reliability and novelty metrics (the length of the paper precludes going into too much detail). Whichever tuple is ultimately given credit, some percentage of the intrinsic value of the newly-activated tuple is added to the intrinsic value of the credited tuple.

State-space discovery: When Duncan the sheepdog is rewarded for sitting in response to the utterance "sit", there are two conclusions he can make: first, that sitting in response to "sit" is a good idea; second, the specific acoustic pattern that was reacted to must be a good example of the utterance "sit." Since the "sit" classifier (in the form of a percept in the Perception System) is actually represented by an example-based statistical model, we can use reward information to update that model. When Duncan receives a large amount of reward for responding to an utterance, that utterance is added back into the statistical model that recognized it in the first place. Thus at the same time as Duncan learns the value of actions, he refines his conception of the appropriate *context* for those actions.

Innovation: Most ActionTuple triggers are direct references to percepts in the percept tree. When they are, the triggers keep statistics about the reliability of the percept's *children* in predicting reward. The hope is that one of the children of the current trigger is actually a better predictor. For example, sitting in response to "Any Utterance" (assume "Any Utterance" is a percept with several children) may be quite a good thing, but sometimes fails entirely to procure reward. By examining the reliability statistics, however, we may discover that sitting in response to a "sit utterance" (assume a "sit utterance" percept is a child of "Any utterance") is far better and more reliable. If this is the case, the ActionTuple in question will create a copy of itself, replacing its trigger with a new trigger that references the "sit utterance" percept. This new ActionTuple will be added as a "child" to the old one, and will supercede its parent when both are relevant in an action selection round. This approach was inspired, in part, by [Drescher 1991] who proposed a very similar scheme for exploring what are essentially state-action pairs.

Innovation can also be prompted in the percept tree. Some forms of statistical models allow hierarchical classifications to be grown (for example, hierarchical clustering algorithms). If an ActionTuple that references such a classifier decides to innovate, it prompts the classifier also to innovate. How this mechanism works ultimately depends on the kind of model the percept contains (clustering algorithms can isolate subclusters and spawn new percepts to represent them). Thus reward feedback can also drive the growth of the percept tree.

Both state-space discovery and innovation illustrate 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*.

3.6 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, must avoid physical obstacles on the way. The Navigation System allows such spatial competencies to be included implicitly in the Action System's high-level behaviors.

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" 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.

3.7 Motor System

Ultimately, the primary output of a virtual creature is motion, whether it be motion for the purposes of locomotion, gesticulation or expressivity. This lowest level – the level of joint-angle control of the transform hierarchy that represents the body of the creature – is controlled by the Motor System. This system takes its inputs from MOTOR_DESIRED and MOTOR_ADVERB entries of the internal blackboard.

The design of the Motor System was inspired by the work of Rose et al [Rose 1999]. The system is organized as a *Verb Graph* in which the 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. Multiple labeled examples of the same Verb may be provided that span an *Adverb space*. If multiple examples are provided, the Motor System does multi-target interpolation at runtime based on blend coefficients provided externally by the MOTOR_ADVERB entry of the blackboard. For example, examples of left-, straight- and right-walks are used to create a continuous space of directional walks.

Layering as discussed in [Perlin et al. 1996] is also supported by the Motor System. Layering allows multiple non-conflicting animations to be played concurrently (for example, walking and waving hand). Duncan the sheepdog has a number of layers corresponding to body-pose, head-layer (for look-ats), tail-layer etc.

Space limitations preclude a full discussion of recent research in motor control, which has centered around an extension of the Verb Graph system called a *Pose-Graph*. Pose-Graphs allow source animation files to be decomposed into atomic animation-chunks which can then be connected into a directed graph through a "pose-space". Creatures can then use this graph to explore new animations, potentially adding these new animations to its existing list of motor skills. This system can also be used to demonstrate simple and complex *shaping*, a training technique through which motor skills are perfected through successive approximations, and *luring*, in which a trainer can lure a creature into a certain pose and then reward that pose. For a complete discussion of the Pose-Graph motor system and its integration into learning, see [Downie 2001].

4.0 Results

Sheep|dog was created to demonstrate some of the basic abilities of the creatures implemented under C4. The project showed creatures acting and reacting to each other and the world. It also employed some of the group's acoustic pattern recognition research to allow Duncan to classify user utterances as one of six possible commands. This classification could be trained through a "one-shot learning" interface so that a new user could achieve a high recognition rate after a very short (about 15 seconds) training routine.

The project also served as a stress test for the system's engineering. It featured two creatures with full brains (Duncan and the shepherd) and six sheep (flocking according to Reynolds' *BOIDS* algorithm [Reynolds 1987]) and a number of world-obstacles, all running scaled-down versions of C4. The project also featured distributed rendering, with two clones running subsets of the system 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. 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 (i.e. Thorndike's Law of Effect, shaping, resistance to extinction etc.). Given an initial repertoire of a dozen basic behaviors (e.g. "sit", "shake", "lie-down", "beg", "jump", "go-out") together with basic navigational and behavioral competencies we have been able to train him to respond to both symbolic gestures (i.e. game-pad button presses), and more significantly to arbitrary acoustic patterns (indeed one user trained Duncan to respond to commands in Gha, the language of Ghana). A dozen such tricks can be trained in real-time within the space of 15 minutes. We have also demonstrated simple shaping with both motor systems and complex shaping and luring with the Pose-Graph based motor system

5.0 Discussion

5.1 Heterogeneous Design

Heterogeneity of decision-policy and representation is seen at multiple levels in C4. The Motor System uses the Verb

Graph structure to plan movements. The Action System makes decisions using ActionTuples. Within the Action System itself, different decision-making policies are employed – Startle ActionTuples are treated differently from Default ActionTuples. This variety, we feel, contributes to the system's robustness and generality. The many kinds of problems that animals can solve entail many kinds of solutions. In order to be a viable platform for intelligence in all its multitude of forms, C4 needed to support and encourage this heterogeneity.

In acting as the go-between for these disparate systems and representations, the creature's internal blackboard plays an important role. This generic communication device allows the easy reordering or omission of entire systems as well as the overriding of the output of one system by another in a way that would be very difficult if input and output pairs were directly coupled.

5.2 Simulation- vs. Mental-Representations

C4 enforces a strict split between simulation representations and mental representations. The former constitute the "ground truth" of the Virtual World, and are used, for example, in generating the graphics output. The latter are the PerceptMemory objects. Forcing a creature to act strictly on the contents of its own memory demands that it make decisions not on the basis of the world's state, but on the basis of its *view* of that state.

By divorcing these two representations, many "second-level" effects become possible, most of them arising from situations in which the two representations fail to match, i.e. 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. They also provide some insight into how real creatures commit and recover from these kinds of errors.

5.3 Supersumption

Two architectural themes are seen throughout C4. The first, reminiscent of Brooks' *subsumption*, is when high-level systems send control signals to low-level systems in order to change their behavior (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. We call this *supersumption*.

This technique is seen in the Action System, when an object of attention chosen by the Attention Group is overridden by an object of attention chosen by the Primary Action Group (since it controls overall action, it probably has a more appropriate choice). It is seen again in the Navigation System, where motor commands sent by the Action System are overridden with more immediately appropriate motor commands. In this case, the Motor System does not know where its instructions come from and the Action System need not concern itself with the details of how its instructions are implemented.

5.4 Easy Behavior Design

Throughout C4, there is much machinery to make it easy for creature designers to specify behavior. 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, etc. One strength of C4 is the system's ability to handle many of these details automatically through mechanisms like the Attention System (which also directs eye gaze), the Navigation System (which hides behavior implementation details from the Action System and the designer) and layers and adverbs in the Motor System. It remains for the designer to fill in a few critical components of the behavior, such as specifying triggers and ObjectContexts. The extensibility of the system also makes it clear where and how a creature's brain needs to be expanded when new abilities are added. (Perhaps the Perception System needs a percept to detect a new type of world event, or perhaps the Motor System needs a new motor skill.) Whatever the case, the system conspires to make *simple things simple and complex things possible*.

5.5 Building the entire system

No actual brain center ever functions in isolation, but instead feeds and is fed by a myriad of other centers. Intelligent behavior is the combination of all their effects. C4, through its extensibility and easy reconfigurability allows us to explore some of the basic interactions between the various components of the brain. This is why building the *entire* creature is critical.

6.0 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. Indeed, C4 is being used as the behavioral engine for a simulated wolf pack. We are also integrating a model of hippocampal spatial learning using landmark and path-integration navigation that will soon give the creatures a truer sense of space. Inspired by the work of Gallistel [Gallistel 2000], we are continuing to formalize our ideas about time, rate and conditioning.

7.0 Related Work

Our work borrows heavily from the impressive work that has come before. Our ideas about super and subsumption between layers follow from Brooks' work (e.g., [Brooks 1991]) as does our emphasis on building the whole creature. However, in contrast to Brooks' work, our layers communicate through Working Memory which seems to us

to be a more general approach and one which reflects our belief that it is difficult to impose a strict layering in practice. 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 Tindbergen and Lorenz. We go beyond this work in our representation of action that admits "time and rate" as first class objects, and in our integration of learning. In addition, our approach to perception, from the Percept Tree to PerceptMemory Objects, is a good deal more sophisticated and powerful than that proposed in these earlier systems. We also allow a "behavior designer" to work at a higher level of abstraction and one which is more familiar to most people. The tangible benefit of this approach is that it is far easier to develop creatures using our system than in our previous work. For example, the creatures described in Yoon 2000 typically required 30-40 pages of source code to specify their behavior systems. By contrast, Duncan is specified in less than 10 pages of source. Finally, our action selection mechanism balances the pragmatic need for both deterministic and probabilistic choice of action. 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, from Minsky [Minsky 1985]. However, by focusing on the 5 big questions (when, to whom, what, for how long, and how much is it worth), the ActionTuple goes beyond these previous representations in providing a surprisingly powerful and intuitive representation for action. Our system also is novel in showing how learning may be integrated into this representation. 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]. Our approach to innovation is directly inspired by Drescher's seminal work [Drescher 1991]. While only touched on briefly in this paper, our system is novel in its ability to perform state-space discovery (i.e. learning new percepts), behavioral adaptation (i.e. learning new ActionTuples), and motor learning (i.e. learning new motor actions) in an integrated framework. Through the use of heuristics such as temporal proximity, simple statistics such as reliability and novelty, and by using the consequences of actions to help discriminate between good and bad examples from which to build models of relevant state, our system provides an interesting example of how learning may be successfully and powerfully integrated into a larger behavioral framework. 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]. Our contribution, particularly with respect to Rose, is to demonstrate the usefulness of his approach.

Acknowledgements

Many thanks to everyone in the Synthetic Characters group who worked hard on C4 and Sheep|Dog: Scott Eaton, Yuri Ivanov, Ben Resner, Bill Tomlinson, Matt Berlin, Jesse Gray and Geoff Beatty.

References

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

[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.

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

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

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

[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, pp 289-344.

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

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

[McCallum 1995], McCallum, A. K. "Reinforcement Learning with Selective Perception and Hidden State", Ph.D. Thesis, CS Department, University of Rochester, 1995.

[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.

[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*, Vol 18, Number 5, 1999.

[Rosenbloom et al. 1993] Rosenbloom, P.S., Laird, J.E. & Newell, A. (1993) *The Soar Papers: Readings on Integrated Intelligence.*Cambridge, MA: MIT Press.

[Treisman 1998] Treisman, A. (1998). The binding problem. In L. Squire & S. Kosslyn (Eds.), *Findings and Current Opinion in Cognitive Neuroscience*. Cambridge: MIT Press, pp 31-38, OR, Current Opinion in Neurobiology, 1996, 6, pp 171-178.

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

[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.