# Cognitive Modeling:
## Knowledge, Reasoning and Planning for Intelligent Characters

John Funge
Intel Corporation

Xiaoyuan Tu
Intel Corporation

Demetri Terzopoulos
University of Toronto

## Abstract

Recent work in behavioral animation has taken impressive steps toward autonomous, self-animating characters for use in production animation and interactive games. It remains difficult, however, to *direct* autonomous characters to perform specific tasks. This paper addresses the challenge by introducing *cognitive modeling*. Cognitive models go beyond behavioral models in that they govern what a character knows, how that knowledge is acquired, and how it can be used to plan actions. To help build cognitive models, we develop the cognitive modeling language CML. Using CML, we can imbue a character with domain knowledge, elegantly specified in terms of actions, their preconditions and their effects, and then direct the character's behavior in terms of goals. Our approach allows behaviors to be specified more naturally and intuitively, more succinctly and at a much higher level of abstraction than would otherwise be possible. With cognitively empowered characters, the animator need only specify a behavior outline or "sketch plan" and, through reasoning, the character will automatically work out a detailed sequence of actions satisfying the specification. We exploit interval methods to integrate sensing into our underlying theoretical framework, thus enabling our autonomous characters to generate action plans even in highly complex, dynamic virtual worlds. We demonstrate cognitive modeling applications in advanced character animation and automated cinematography.

**Keywords:** Computer Animation, Character Animation, Intelligent Characters, Behavioral Animation, Cognitive Modeling, Knowledge, Sensing, Action, Reasoning, Planning

**CR Categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—Representation languages, Modal logic, Temporal logic, Predicate logic; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search—Graph and tree search strategies, Heuristic methods; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

[0]www.cs.toronto.edu/$\sim${funge,tu,dt}

Published in the Proceedings of SIGGRAPH 99 (Los Angeles, CA, August, 1999). In *Computer Graphics* Proceedings, Annual Conference Series, 1999, ACM SIGGRAPH, pp. 29–38.

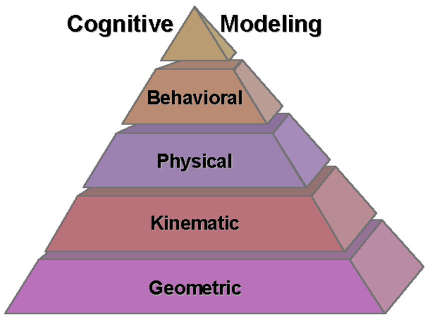


Figure 1: Cognitive modeling is the new apex of the CG modeling hierarchy.

## 1 Making Them Think

Modeling for computer animation addresses the challenge of automating a variety of difficult animation tasks. An early milestone was the combination of geometric models and inverse kinematics to simplify keyframing. Physical models for animating particles, rigid bodies, deformable solids, fluids, and gases have offered the means to generate copious quantities of realistic motion through dynamic simulation. Biomechanical modeling employs simulated physics to automate the lifelike animation of animals with internal muscle actuators. Research in behavioral modeling is making progress towards self-animating characters that react appropriately to perceived environmental stimuli. It has remained difficult, however, to *direct* these autonomous characters so that they satisfy the animator's goals. Hitherto absent in this context has been a substantive apex to the computer graphics modeling pyramid (Fig. 1), which we identify as *cognitive modeling*.

This paper introduces and develops cognitive modeling for computer animation and interactive games. Cognitive models go beyond behavioral models in that they govern what a character knows, how that knowledge is acquired, and how it can be used to plan actions. Cognitive models are applicable to directing the new breed of highly autonomous, quasi-intelligent characters that are beginning to find use in production animation and interactive computer games. Moreover, cognitive models can play subsidiary roles in controlling cinematography and lighting.

We decompose cognitive modeling into two related sub-tasks: *domain knowledge specification* and *character direction*. This is reminiscent of the classic dictum from the field of artificial intelligence (AI) that tries to promote modularity of design by separating out knowledge from control. Domain (knowledge) specification involves administering knowledge to the character about its world and how that world can change. Character direction involves instructing the character to try to behave in a certain way within its world in order to achieve specific goals. Like other advanced modeling tasks, both of these steps can be fraught with difficulty unless animators are given the right tools for the job. To this end, we develop the cognitive modeling language, CML.

### 1.1 CML: knowledge + directives = intelligent behavior

CML rests on a solid foundation grounded in theoretical AI. This high-level language provides an intuitive way to give characters, and also cameras and lights, knowledge about their domain in terms of actions, their preconditions and their effects. We can also endow characters with a certain amount of "common sense" within their domain and we can even leave out tiresome details from the directives we give them. The missing details are automatically filled in at run-time by the character's reasoning engine which decides what must be done to achieve the specified goal.

Traditional AI style planning [1] certainly falls under the broad umbrella of this description, but the distinguishing features of CML are the intuitive way domain knowledge can be specified and how it affords an animator familiar control structures to focus the power of the reasoning engine. This forms an important middle ground between regular logic programming (as represented by Prolog) and traditional imperative programming (as typified by C). Moreover, this middle ground turns out to be crucial for cognitive modeling in animation and computer games. In one-off animation production, reducing development time is, within reason, more important than fast execution. The animator may therefore choose to rely more heavily on the reasoning engine. When run-time efficiency is also important, our approach lends itself to an incremental style of development. We can quickly create a working prototype. If this prototype runs too slowly, it may be refined by including increasingly detailed knowledge to narrow the focus of the reasoning engine.

### 1.2 Related work

Tu and Terzopoulos [25, 24] have taken major strides towards creating realistic, self-animating graphical characters through biomechanical modeling and the principles of behavioral animation introduced in the seminal work of Reynolds [21]. A criticism sometimes leveled at behavioral animation methods is that, robustness and efficiency notwithstanding, the behavior controllers are hard-wired into the code. Blumberg and Galyean [7] begin to address such concerns by introducing mechanisms that give the animator greater control over behavior, and Blumberg's superb thesis considers interesting issues such as behavior learning [6]. While we share similar motivations, our research takes a different route. One of its unique features is the emphasis we place on investigating important higher-level cognitive abilities, such as knowledge representation, reasoning, and planning, which are the domain of AI. The research teams led by Badler [3], Bates [4], Hayes-Roth [13], and the Thalmanns [17] have applied AI techniques to produce inspiring results with animated humans or cartoon characters.

The theoretical basis of our work is new to the graphics community and we consider some novel applications. We employ an AI formalism known as the situation calculus. The version we use is a recent product of the cognitive robotics community [15]. A noteworthy point of departure from existing work in cognitive robotics is that we render the situation calculus amenable to animation within highly dynamic virtual worlds by introducing interval valued fluents [10, 12, 11] to deal with sensing.

Perlin [19] describes fascinating work aimed at providing animators with useful behavior modeling tools. Our work on defining and implementing the cognitive modeling language CML complements these efforts by encapsulating some of the basic concepts and techniques that may soon enough be incorporated into advanced tools for animation. Autonomous camera control for animation is particularly well suited to our cognitive modeling approach because there already exists a large body of widely accepted rules upon which we can draw [2]. This fact has also been exploited by a recent paper on the subject which implement hierarchical finite state machines for camera control [14]. Our approach to camera control employs CML.

### 1.3 Overview

The remainder of the paper is organized as follows. Section 2 covers the theoretical foundations of our research and presents our cognitive modeling language CML. Section 3 presents our work on automated cinematography, the first of three case studies. Here, our primary aim is to show how separating out the control information from the background domain knowledge makes it easier to understand and maintain controllers. Our camera controller is ostensibly reactive, making minimal use of CML's planning capabilities, but it demonstrates that cognitive modeling subsumes conventional behavioral modeling as a limiting case. Section 4 presents two case studies in character animation that highlight the ability of our approach to generate intelligent behavior consistent with goal-directed specification by exploiting domain knowledge and reasoning. In a "prehistoric world" case study, we show how our tools can simplify the development of cognitive characters that autonomously generate knowledge-based, goal-directed behavior. In an "undersea world" case study, we produce an elaborate animation which would overwhelm naive goal-directed specification approaches. We demonstrate how cognitive modeling allows the animator to provide a loose script for the characters to follow; some of the details of the animation are provided by the animator while the rest are filled in automatically by the character. Section 5 presents conclusions and suggestions for future work.

## 2 Theoretical Background

The *situation calculus* is an AI formalism for describing changing worlds using sorted first-order logic. Mathematical logic is somewhat of a departure from the repertoire of mathematical tools commonly used in computer graphics. We shall therefore overview in this section the salient points of the situation calculus, whose details are well-documented elsewhere (e.g., [10, 11, 15]). We emphasize that from the user's point of view the underlying theory is *hidden*. In particular, a user is *not* required to type in axioms written in first-order mathematical logic. Instead, we have developed an intuitive high-level interaction language CML whose syntax employs descriptive keywords, but which has a clear and precise mapping to the underlying formalism.

### 2.1 Domain modeling

A *situation* is a "snapshot" of the state of the world. A domain-independent constant $\mathsf{S}_0$ denotes the initial situation. Any property of the world that can change over time is known as a *fluent*. A fluent is a function, or relation, with a situation term (by convention) as its last argument. For example $\mathsf{Broken}(x, s)$ is a fluent that keeps track of whether an object $x$ is broken in a situation $s$.

*Primitive actions* are the fundamental instrument of change in our ontology. The sometimes counter-intuitive term "primitive" serves only to distinguish certain atomic actions from the "complex", compound actions that we will define in Section 2.3. The situation $s'$ resulting from doing action $a$ in situation $s$ is given by the distinguished function *do*, so that $s' = \mathit{do}(a, s)$. The possibility of performing action $a$ in situation $s$ is denoted by a distinguished predicate $\mathit{Poss}\,(a, s)$. Sentences that specify what the state of the world must be before performing some action are known as *precondition axioms*. For example, it is possible to drop an object $x$ in a situation $s$ if and only if a character is holding it, $\mathit{Poss}\,(\mathit{drop}(x), s) \Leftrightarrow \mathsf{Holding}(x, s)$. In CML, this axiom can be expressed more intuitively without the need for logical connectives and the explicit situation argument as follows:[1]

[1]To promote readability, all CML keywords will appear in bold type, actions (complex and primitive) will be italicized, and fluents will be underlined. We will also use various other predicates and functions that are not

**action** *drop*$(x)$ **possible when** $\underline{\text{Holding}}(x)$;

The effects of an action are given by *effect axioms*. They give necessary conditions for a fluent to take on a given value after performing an action. For example, the effect of dropping an object $x$ is that the character is no longer holding the object in the resulting situation, and vice versa for picking up an object. This is stated in CML as follows:

**occurrence** *drop*$(x)$ **results in** $!\underline{\text{Holding}}(x)$; ("!" denotes negation)
**occurrence** *pickup*$(x)$ **results in** $\underline{\text{Holding}}(x)$;

Surprisingly, a naive translation of the above statements into the situation calculus does not give the expected results. In particular, stating what does not change when an action is performed is problematic. This is called the "frame problem" in AI [18]. That is, a character must consider whether dropping a cup, for instance, results in, say, a vase turning into a bird and flying about the room. For mindless animated characters, this can all be taken care of implicitly by the programmer's common sense. We need to give our thinking characters this same common sense. They need to be told that they should assume things stay the same unless they know otherwise. Once characters in virtual worlds start thinking for themselves, they too will have to tackle the frame problem. The frame problem has been a major reason why approaches like ours have not previously been used in computer animation or until recently in robotics. Fortunately, the frame problem can be solved provided characters represent their knowledge with the assumption that effect axioms enumerate all the possible ways that the world can change. This so-called closed world assumption provides the justification for replacing the effect axioms with *successor state axioms* [20].[2]

## 2.2 Sensing

Artificial life in a complex, dynamic virtual world should appear as thrilling and unpredictable to the character as it does to the human observer. Compare the excitement of watching a character run for cover from a falling stack of bricks to one that accurately precomputes brick trajectories and, realizing that it is in no danger, stands around nonchalantly while bricks crash down around it. On a more practical note, the expense of performing multiple speculative high fidelity forward simulations could easily be prohibitive. Usually it makes far more sense for a character to decide what to do using a simplified mental model of its world, sense the outcome, and perform follow up actions if things don't turn out as expected.

We would therefore like characters that can realize when they have some outdated information and can then perform a sensing action to get more current information so that they can replan a new course of action. A simple way to sidestep the issue of when to sense is to have characters replan periodically. One problem with this is that it is wasteful when there is no real need to replan. Even worse a character might not be replanning enough at certain critical times. Consequently, we would like characters that can replan *only when necessary*.

To this end, we must come up with a way for a character to represent its uncertainty about aspects of the world. Previous approaches to the problem in AI used modal logic to represent what a character knows and doesn't know. The so-called epistemic K-fluent allows us, at least in principle, to express an agent's uncertainty about the value of a fluent in its world [22]. Unfortunately, the result is primarily of theoretical interest as there are as yet no clear ideas regarding its practical implementation.[3] Next, we shall instead propose the practicable concept of interval-valued epistemic fluents [10, 12, 11].

fluents. These will not be underlined and will have names to indicate their intended meaning. The convention in CML is that fluents to the right of the **when** keyword refer to the current situation.

[2]For example, the CML statements given above can now be effectively translated into the following successor state axiom that CML uses internally to represent how the character's world can change. The axiom states that, provided the action is possible, then a character is holding an object $x$ if and only if it just picked up the object or it was holding the object before and it did not just drop the object: $\textit{Poss}\,(a,s) \Rightarrow [\text{Holding}(x, \textit{do}(a,s)) \Leftrightarrow a = \textit{pickup}(x) \vee (a \neq \textit{drop}(x) \wedge \text{Holding}(x,s))]$.

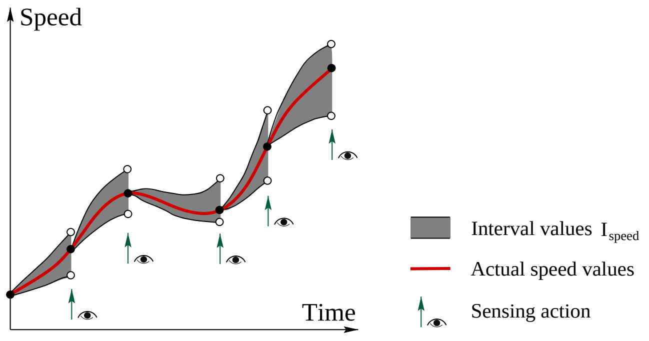


Figure 2: Sensing narrows IVE fluents bounding the actual value.

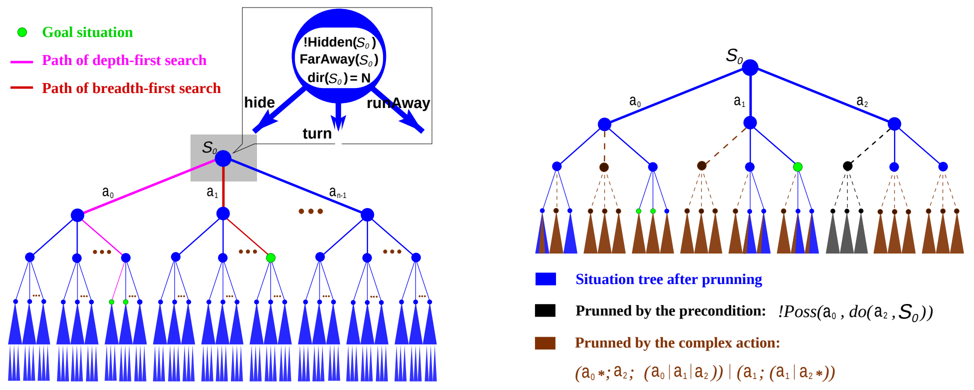
### 2.2.1 Interval-valued epistemic (IVE ) fluents

Interval arithmetic is relatively well-known to the graphics community [23, 26]. It can be used to express uncertainty about a quantity in a way that circumvents the problem of using a finite representation for an uncountable number of possibilities. It is, therefore, natural to ask whether we can also use intervals to replace the troublesome epistemic K-fluent. The answer, as we show in [10, 12, 11], is affirmative. In particular, for each sensory fluent $f$, we introduce an interval-valued epistemic (IVE ) fluent $\mathcal{I}_f$. The IVE fluent $\mathcal{I}_f$ is used to represent an agent's uncertainty about the value of $f$. Sensing now corresponds to making intervals narrower.[4]

Let us introduce the notion of *exogenous* actions (or events) that are generated by the environment and not the character. For example, we can introduce an action *setSpeed* that is generated by the underlying virtual world simulator and simply sets the value of a fluent $\underline{\text{speed}}$ that tracks an object's speed. We can introduce an IVE fluent $\underline{\mathcal{I}_{\text{speed}}}$ that takes on values in $\mathcal{I}_{\mathbb{R}^{\star+}}$ (the extended positive real numbers), which denotes the set of pairs $\langle u, v\rangle$ such that $u, v \in \mathbb{R}^{\star+}$ and $u \leqslant v$). Intuitively, we can now use the interval $\mathcal{I}_{\text{speed}}(\mathrm{S}_0) = \langle 10, 50\rangle$ to state that the object's speed is initially known to be between 10 and 50 m/sec. Now, as long as we have a bound on how fast the speed can change, we can always write down logically true statements about the world. Moreover, we can always bound the rate of change. That is, in the worst case we can choose our rate of change as infinite so that, except after sensing, the character is completely ignorant of the object's speed in the current situation: $\mathcal{I}_{\text{speed}}(s) = \langle 0, \infty\rangle$. Figure 2 depicts the usual case when we do have a reasonable bound. The solid line is the actual speed $\underline{\text{speed}}$ and the shaded region is the interval guaranteed to bound the object's speed. Notice that the character's uncertainty about the object's speed increases over time (i.e., the intervals grow wider) until a sensing action causes the interval to once again collapse to its actual value (assuming noise-free sensing). Whenever

[3]Potential implementations of the epistemic K-fluent are plagued by combinatorial explosion. In general, if we have $n$ relational fluents whose values may be learned through sensing, then we must list potentially $2^n$ initial possible worlds. Things get even worse with functional fluents whose range is the real numbers $\mathbb{R}$, since we cannot list out the uncountably many possible worlds associated with uncertainty about their value.

[4]IVE fluents represent *uncertainty intervals* about time-dependent variables. They do not represent and are unrelated to *time intervals* of the sort that have been used in the underlying semantics of various temporal logics (for example see [16]).



Goal situation
Path of depth-first search
Path of breadth-first search
!Hidden($S_0$)
FarAway($S_0$)
dir($S_0$) = N
hide
turn
runAway
$S_0$
$a_0$
$a_1$
$a_{n-1}$
(a)
$S_0$
$a_0$
$a_1$
$a_2$
Situation tree after prunning
Prunned by the precondition: $!Poss(a_0, do(a_2, S_0))$
Prunned by the complex action:
$(a_0*; a_2; (a_0|a_1|a_2)) | (a_1; (a_1|a_2*))$
(b)

Figure 3: The situation tree (a). Pruning the tree (b).

the interval is less than a certain width, we say that the character "knows" the property in question. We can then write precondition axioms based not only upon the state of the world, but also on the state of the character's knowledge of its world. For example, we can state that a character cannot calculate its travel time unless it knows its speed. So, if a character wishes to know when it will arrive somewhere, but does not know its speed (i.e., $\mathcal{I}_{\text{speed}}(s)$ is too wide), then it can infer that it must perform a sensing action. In [10, 12, 11] we prove several theorems that allow us to justify formally our IVE fluent as a replacement for the troublesome $\mathcal{K}$-fluent.

## 2.3 Complex actions

The actions, effect axioms and preconditions we have described so far can be thought of as a tree (Fig. 3(a)). The nodes of the tree represent situations. Effect axioms describe the characteristics of each situation. At the root of the tree is the initial situation and each path through the tree represents a possible sequence of actions. The precondition axioms mean that some sequences of actions are not possible. This is represented in the picture by the black portion of the tree. If some situations are desired "goals" then we can use a conventional logic programming approach to automatically search the tree for a sequence of actions that takes us to the goal. The green nodes in the figure represent goal situations and we can use various search strategies to come up with an appropriate sequence of actions to perform. The red path shows the sequence of actions that result from a breadth-first search of the tree, and the magenta path from depth-first search.

The problem with the exhaustive search approach is that the search space is exponential in the length of the plan. Much of the planning literature has sought to address this problem with more sophisticated search algorithms, such as the well known $A^\star$ algorithm, or stochastic planning techniques. We shall introduce a different approach. In particular, we shall be looking at how to speed up planning by "pruning" the search space. How we choose to search the remaining space is an important but independent problem for which all the previous work on planning is equally applicable.

It is interesting to note that conventional imperative style programming can be regarded as a way to prune the tree down to a single path. That is, there is no "searching" and the programmer bears the sole responsibility for coming up with a program that generates the desired sequence of actions. However, by defining what we refer to as "complex actions" we can prune part of the search tree. Figure 3(b) represents the complex action $(a_0\star \,\S\, a_2 \,\S\, (a_0|a_1|a_2))|(a_1 \,\S\, (a_1|a_2)\star)$ and its corresponding effect of reducing the search space to the blue region of the tree. In what follows we shall see more examples of complex actions and their definitions. For now, it is important to understand that the purpose of complex actions is to give us a convenient tool for encoding any heuristic knowledge we have about the problem. In general, the search space will still be exponential, but reducing the search space can make the difference between a character that can tractably plan only 5 steps ahead and one that can plan 15 steps ahead. That is, we can get characters that appear a lot more intelligent!

The theory underlying *complex actions* is described in [15]. Complex actions consist of a set of recursively defined operators. Any primitive action is also a complex action. Other complex actions are composed using various control structures. As a familiar artifice to aid memorization, the control structure syntax of CML is designed to resemble C. Fig. 4 gives the complete list of operators for specifying complex actions. Together, these operators define the instruction language we use to issue direction to characters.

Although the syntax of CML is similar to a conventional programming language, CML is a strict superset in terms of functionality. The user can give characters instructions based on behavior outlines, or "sketch plans". In particular, a behavior outline can be nondeterministic. By this we mean that we can cover multiple possibilities in one instruction, not that the behavior is random. As we shall explain, this added freedom allows many behaviors to be specified more naturally, more simply, more succinctly and at a much higher level of abstraction than would otherwise be possible. Using its background knowledge, the character can decide for itself how to fill in the necessary missing details.

As a first serious example of a powerful complex action, the one to the left below,[5] with its corresponding CML code on the right, defines a depth-bounded (to $n$ steps) depth-first planner:

```
**proc** *planner*$(n)$
    goal? |
    $[(n > 0)? \,\S$
        $(\pi\ a)($primitiveAction$(a)? \,\S\, a) \,\S$
        *planner*$(n-1)]$
**end**
```

```
**proc** *planner*$(n)$ {
    **choose test**($\underline{\text{goal}}$);
    **or** {
        **test**$(n > 0)$;
        **pick**$(a)$ {
            primitiveAction$(a)$;
            **do**$(a)$; }}
    *planner*$(n-1)$; }
```

We have written a Java application, complete with documentation, that is publicly available to further assist the interested reader in mastering this novel language [8].

[5]Adopted from R. Reiter's forthcoming book "Knowledge in Action".

(Primitive Action)
If $\alpha$ is a primitive action then, provided the precondition axiom states it is possible, do the action.
[same syntax in CML i.e. $<$ACTION$>$; except we must use an explicit **do** when the action is a variable.]

(Sequence)
$\alpha \fatsemi \beta$ means do action $\alpha$, followed by action $\beta$.
[$<$ACTION$>$ ; $<$ACTION$>$ ; (note the semi-colon is used as a statement terminator to mimic C)]

(Test)
$p?$ succeeds if $p$ is true, otherwise it fails.
[**test**($<$EXPRESSION$>$)]

(Nondeterministic choice of actions)
$\alpha \mid \beta$ means do action $\alpha$ or action $\beta$.
[**choose** $<$ACTION$>$ **or** $<$ACTION$>$]

(Conditionals)
**if** $p$ $\alpha$ **else** $\beta$ **fi**, is just shorthand for $p? \fatsemi \alpha \mid (\neg p)? \fatsemi \beta$.
[**if** ($<$EXPRESSION$>$) $<$ACTION$>$ **else** $<$ACTION$>$]

(Non-deterministic iteration)
$\alpha\ \star$, means do $\alpha$ zero or more times.
[**star** $<$ACTION$>$]

(Iteration)
**while** $p$ **do** $\alpha$ **od** is just shorthand for $p?\ \alpha\ \star$.
[**while** ($<$EXPRESSION$>$) $<$ACTION$>$]

(Nondeterministic choice of arguments)
$(\boldsymbol{\pi}\ x)\ \alpha$ means pick some argument $x$ and perform the action $\alpha(x)$.
[**pick**($<$EXPRESSION$>$) $<$ACTION$>$]

(Procedures)
**proc** $P(x_1, \ldots, x_n)$ $\alpha$ **end** declares a procedure that can be called as $P(x_1, \ldots, x_n)$.
[**void** $P$($<$ARGLIST$>$) $<$ACTION$>$]

Figure 4: Complex action operators. Following each definition, the equivalent CML syntax is given in square brackets. The mathematical definitions for these operators are given in [15]. It is straightforward to modify the complex action definitions to include a check for any exogenous actions and, if necessary, include them in the sequence of resulting actions (see [10, 11] for more details).

## 3 Automated Cinematography

At first it might seem strange to advocate building a cognitive model for a camera. We soon realize, however, that it is natural to capture in a cognitive model the knowledge of the director and cameraperson who control the camera. In effect, we want to treat all the elements of a scene, be they lights, cameras, or characters as "actors". CML is ideally suited to realizing this approach.

To appreciate what follows, the reader may benefit from a rudimentary knowledge of cinematography. The exposition on principles of cinematography given in Section 2.3 of [14] is an excellent starting point. In [14], the authors discuss one particular formula for filming two characters conversing. The idea is to flip between "external" shots of each character, focusing on the character doing the talking (Fig. 5). To break the monotony, the shots are interspersed with reaction shots of the other character. In [14], the formula is encoded as a finite state machine. We will show how elegantly we can capture the formula using the instruction facilities of CML. First, however, we need to specify the domain. For conciseness, we restrict ourselves to explaining only the principal aspects of the specification (see [10, 9, 11] for the details).

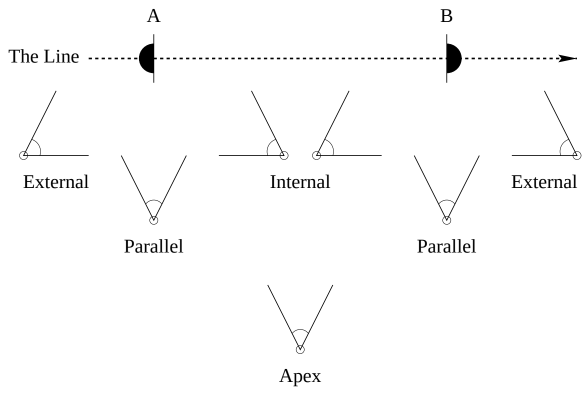


Figure 5: Common camera placements relative to characters $A$, $B$.

### 3.1 Camera domain

Assuming that the motion of all the objects in the scene has been computed, our task is to decide the vantage point from which each frame is to be rendered. The fluent <u>frame</u> keeps track of the current frame number, and a *tick* action causes it to be incremented. The precomputed scene is represented as a lookup function scene which completely specifies the position, orientation, and shape of each object in each frame.

The most common camera placements used in cinematography will be modeled in our formalization as primitive actions. These actions are referred to in [14] as "camera modules". This is a good example where the term "primitive" is misleading. As described in [5], low-level camera placement is a complex and challenging task in its own right. Here we shall make some simplifications to clarify our exposition. More realistic equations are easily substituted, but the principles remain the same. For now, we specify the camera with two fluents <u>lookFrom</u>, and <u>lookAt</u>. Let us assume that up remains constant and also make the simplifying assumption that the viewing frustrum is fixed. Despite our simplifications, we still have a great deal of flexibility in our specifications. We will now give examples of effect axioms for some of the primitive actions in our ontology.

The *fixed* action is used to specify explicitly a particular camera configuration. We can, for example, use it to provide an overview shot of the scene:

**occurrence** *fixed*$(e,c)$ **results in** <u>lookFrom</u> $= e$ && <u>lookAt</u> $= c$;

A more complicated action is *external*. It takes two arguments, character $A$, and character $B$ and places the camera so that $A$ is seen over the shoulder of $B$. One effect of this action, therefore, is that the camera is looking at character $A$:

**occurrence** *external*$(A,B)$ **results in** <u>lookAt</u> $= p$ **when**
  scene($A$(upperbody,centroid)) $= p$;

The other effect is that the camera is located above character $B$'s shoulder. This could be done with an effect axiom such as:

**occurrence** *external*$(A,B)$ **results in**
  <u>lookFrom</u> $= p + k_2 *$ up $+ k_3 *$ normalize$(p - c)$ **when**
  scene($B$(shoulder,centroid)) $= p$ && scene($A$(upperbody,centroid)) $= c$;

where $k_2$ and $k_3$ are some suitable constants. There are many other possible camera placement actions. Some of them are listed in [14], and others may be found in [2].

The remaining fluents are concerned with more esoteric aspects of the scene, but some of their effect axioms are mundane and so we shall only explain them in English. For example, the fluent <u>Talking</u> $(A,B)$ (meaning $A$ is talking to $B$) becomes true after a *startTalk*

$(A,B)$ action, and false after a *stopTalking* $(A,B)$ action. Since we are currently only concerning ourselves with camera placement, it is the responsibility of the application that generates the scene descriptions to produce the start and stop talking actions (i.e., the start and stop talking actions are represented as exogenous actions within the underlying formal semantics).

A more interesting fluent is silenceCount, which keeps count of how long it has been since a character spoke:

```
occurrence tick results in silenceCount = $n - 1$
    when silenceCount = $n$ && !Talking($A$,$B$);
occurrence stopTalk($A$,$B$) results in silenceCount = $k_a$;
occurrence setCount results in silenceCount = $k_a$;
```

Note that $k_a$ is a constant ($k_a = 10$ in [14]), such that the counter will be negative after $k_a$ ticks of no-one speaking. A similar fluent filmCount keeps track of how long the camera has been pointing at the same character:

```
occurrence setCount || external($A$,$B$) results in filmCount = $k_b$
    when Talking($A$,$B$);
occurrence setCount || external($A$,$B$) results in filmCount = $k_c$
    when !Talking($A$,$B$);
occurrence tick results in filmCount = $n - 1$ when filmCount = $n$;
```

where $k_b$ and $k_c$ are constants ($k_b = 30$ and $k_c = 15$ in [14]) that state how long we can continue the same shot before the counter becomes negative. Note that the constants for the case of looking at a non-speaking character are lower. We will keep track of which constant we are using with the fluent tooLong.

For convenience, we now introduce two defined fluents that express when a shot has become boring because it has gone on too long, and when a shot has not gone on long enough. We need the notion of a minimum time for each shot to avoid annoying flitter between shots:[6]

```
defined Boring := filmCount $< 0$;
defined TooFast := tooLong - $k_s \leqslant$ filmCount;   (where $k_s$ is a constant)
```

Finally, we introduce a fluent Filming to keep track of the character at whom the camera is pointing.

Until now we have not mentioned any preconditions for our actions. Unless stated otherwise, we assume that actions are always possible. In contrast, the precondition axiom for the *external* camera action states that we only want to point the camera at character $A$ if we are already filming $A$ and it has not yet gotten boring, or if we are not filming $A$, and $A$ is talking, and we have stayed with the current shot long enough:

```
action external($A$,$B$) possible when (!Boring && Filming($A$)) ||
    (Talking($A$,$B$) && !Filming($A$) && !TooFast);
```

We are now in a position to define the controller that will move our "cognitive camera" to shoot the character doing the talking, with occasional respites to focus on the other character's reactions:

```
setCount;
while ($0 <$ silenceCount) {
    pick($A$,$B$) external($A$,$B$);
    tick; }
```

This specification makes heavy use of the ability to nondeterministically choose arguments. The reader can contrast our specification with the encoding given in [14] to achieve the same result.

[6]A *defined* fluent is defined in terms of other fluents, and therefore, its value changes implicitly as the fluents on which it depends change. The user must be careful to avoid any circular definitions when using defined fluents. A defined fluent is indicated with the keyword "**defined**" and symbol ":=".

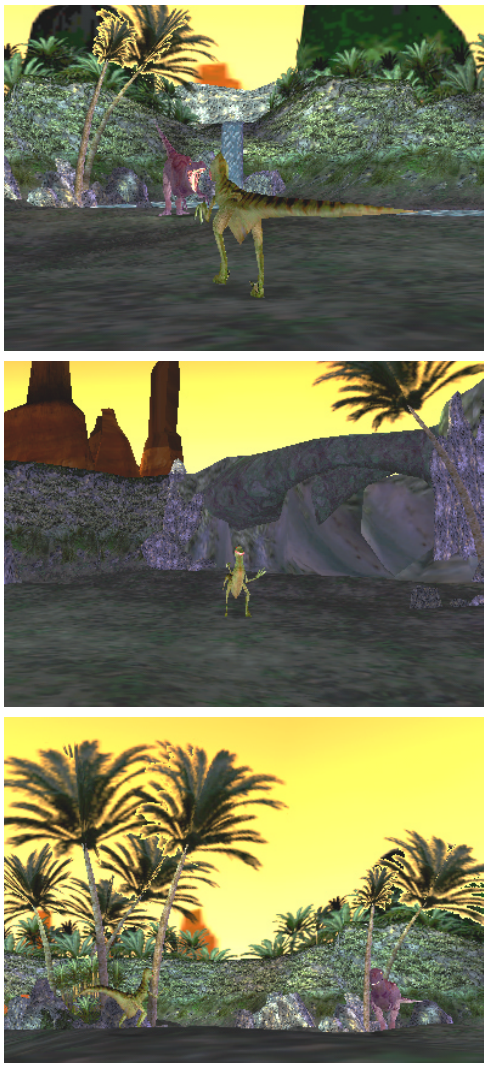


Figure 6: The "Cinemasaurus" autonomous camera animation. (top) External shot of the T-Rex. (center) Internal shot of the Raptor. (bottom) Apex shot of the actors.

## 4 Character Animation

We now turn our attention to the main application of our work, character animation. Our first example is a prehistoric world and the second is an undersea world. The two worlds are differentiated by the complexity of their underlying models, the undersea world model being significantly more complex.

### 4.1 Prehistoric world

The prehistoric world, comprising a volcanic territory and a jungle territory, is inhabited by a Tyrannosaurus Rex (T-Rex) and Velocirators (Raptors). It is implemented as a game engine API which runs in real-time any modern PC. The dinosaur characters are animated by keyframed footprints and inverse kinematics to position the legs onto the ground. To add some physical realism, the body is modeled as a point mass that moves dynamically in response to the leg movements.

We interfaced the game engine to a reasoning engine implemented in C++.[7] The performance of the cognitive modeling aug-

[7]We first tried compiling our CML specifications into Prolog using our

mented prehistoric world remains real-time on average, but we see occasional pauses when the reasoning engine takes longer than usual to plan a suitable behavior. We will present two cognitive modeling animations. The first one demonstrates our approach to camera control and the second demonstrates plan-based territorial behavior.

The action in the camera control demonstration consists of a T-Rex and a Raptor "conversing" by roaring at each other. The camera always films the dinosaur that is roaring unless it roars for too long, in which case it will get a reaction shot from the other dinosaur. The T-Rex has an additional behavior—if it gets bored listening to a yapping Raptor, it will attack! The camera will automatically track moving creatures. Sample frames from the resulting animation "Cinemasaurus", which was filmed automatically in the jungle territory by our cognitive camera, are shown in figure 6. The cognitive camera uses essentially the same CML code as the example in Section 3, although some of the camera angles are programmed a bit differently.

In the territorial T-Rex animation our challenge is to administer enough knowledge to the T-Rex about its world, especially about the reactive behavior of the Raptors (which behave not unlike Reynold's "boids" [21]), so that the T-Rex knows enough to automatically formulate plans for expelling Raptors out of its volcanic territory and into the neighboring jungle territory. To this end, the T-Rex must herd Raptors through a narrow passage that connects the two territories. The passage is marked by a stone arch at the northwest corner of the volcanic territory.

The Raptors have good reason to fear the larger, stronger and highly vicious T-Rex should it come close. The following code shows how we use CML to instruct the T-Rex that the Raptors will become frightened when it approaches them:

```
occurrence move(dir) results in Frightened(Raptor($i$))
    when position(T-Rex) = $p$ && position(Raptor($i$)) = $q$ &&
        $|q -$ adjacent(p, direction)$| \leqslant \Delta$;
```

The code we used in the demonstration was slightly more complicated in that we also instructed the T-Rex that even less proximal Raptors would also become frightened if it roared. A second CML expression tells the T-Rex that frightened Raptors will run away from it:

```
defined heading(Raptor($i$)) = direction
    where (Frightened(Raptor($i$)) &&
        direction = opposite(directionT-Rex)) ||
        (!Frightened(Raptor($i$)) && direction = directionOld)
    when relativeDirectionOfT-Rex (Raptor($i$)) = directionT-Rex &&
        heading(Raptor($i$)) = directionOld;
```

Here, relativeDirectionOfT-Rex is a fluent that is easily defined in terms of the relative positions of the T-Rex and Raptor $i$.

With a third CML expression, we instruct the T-Rex to plan paths that avoid obstacles:[8]

```
action move(direction) possible
    when position(T-Rex) = $p$ && Free(adjacent($p$, direction));
```

Given enough patience, skill and ingenuity, it is conceivable that one could successfully program herding behavior using sets of stimulus-response rules. Using CML, we can do the same thing

online Java applet [8] and then linking the compiled Prolog code with the API using Quintus Prolog's ability to link with Visual C++. Although convenient, this approach adversely affected the real-time performance, so we abandoned it in favor of a complete C++ implementation of the reasoning engine.

[8]In fact, the T-Rex autonomously maps out all the obstacles by exploring its world in a preprocessing step. When it encounters an obstacle, the T-Rex remembers the location of the obstacle in a mental map of its world.

with relative ease through much higher-level, goal-directed specification. Suppose we want to get some Raptors heading in a particular direction. Then, we simply give the T-Rex the goal of getting more Raptors heading in the right direction than are initially heading that way. Here is how this goal is specified in CML:

```
defined goal := NumRaptorsInRightDirection = $n$ && $n \geqslant n_0 + k$
    when initially $n_0$ = NumRaptorsInRightDirection;
```

This goal along with our previous instructions enable the T-Rex to plan its actions like a smart sheepdog. It autonomously plans collision-free paths to maneuver in and around groups of Raptors in order to frighten them in the desired direction.

The T-Rex plans up to 6 moves ahead of its current position. Longer duration plans degrade real-time performance. They are also rarely useful in a highly kinetic world about which the T-Rex has only partial knowledge. A better strategy is adaptive herding through periodic re-planning. To speed things up we also defined undesirable situations using the fluent Undesirable. These are the antithesis of goals in that they represent situations that, although not illegal, are undesirable. For example, if the Raptors are too far away there is no point in roaring as it will have no effect. Therefore a situation in which the T-Rex roars without anticipating any Raptors changing direction is useless, hence undesirable:

```
defined Undesirable after roar := NumRaptorsInRightDirection = $n$
    when NumRaptorsInRightDirection = $n_0$ && $n_0 \geqslant n$;
```

The T-Rex need not consider this or its subsequent situations when searching for appropriate behavior.

The pack of reactive Raptors prefer to stay away from the passage under the arch, but the smarter, cognitively empowered T-Rex succeeds in expelling this unruly mob from its territory.[9] Some frames from the corresponding animation are shown in figure 7.

## 4.2 Undersea world

Our undersea world is entirely physics-based. It is inhabited by mermen, fabled creatures of the sea with the head and upper body of a man and the tail of a fish. Its other inhabitants are predator sharks. An artificial life simulator implements the virtual creatures as fully functional (pre-cognitive) autonomous agents. The modeling is similar to that in [25, 24]. It provides a *graphical display model* that captures the form and appearance of our characters, a *biomechanical model* that captures their anatomical structure, including internal muscle actuators, and simulates the deformation and physical dynamics of the character's body, and a *behavioral control model* that implements the character's brain and is responsible for motor, perception and low-level behavior control. A merman's reactive behavior system interprets his intentions and generates coordinated muscle actions. These effect locomotion by deforming the body to generate propulsion-inducing forces against the virtual water. The sharks are animated likewise.

Our goal is to equip the mermen with a cognitive model that enables them to reason about their world based on acquired knowledge, thus enabling them to interpret high-level direction from the animator. Fig. 8 depicts the relationship between the user, the reasoning system and the reactive system.

The simulated dynamics makes it hard for a merman to reason precisely about his world because, as is the case in the real world, it is possible to predict only approximately the ultimate effect of one's actions. However, the reactive behavior model helps by mediating between the reasoning engine and the physics-based environment. Thus at the higher level we need only consider actions

[9]Note that all a "reactive T-Rex" (i.e. a cognitive T-Rex allowed to plan only a single move ahead) can do is aimlessly chase the agile Raptors around. Only by sheer luck can it eventually chase a few Raptors through the narrow passage under the arch and out of its territory.

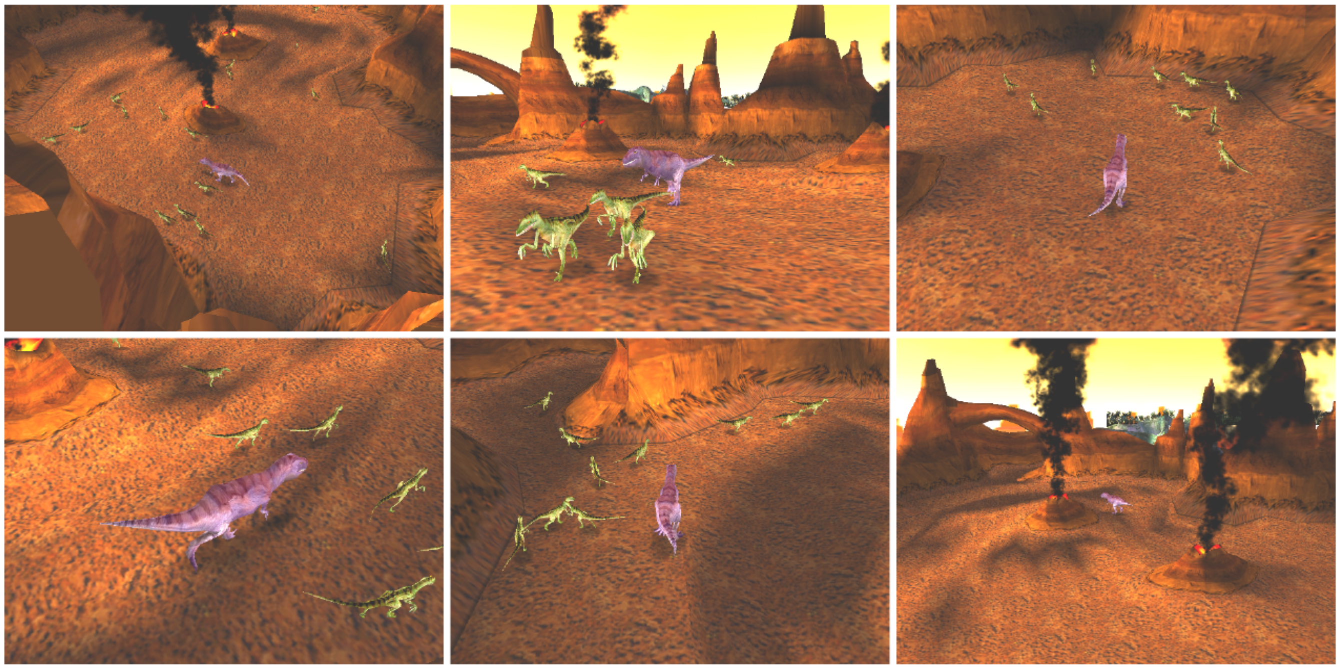


Figure 7: The “Territorial T-Rex” animation. A cognitively empowered T-Rex herds Raptors like a smart sheepdog.

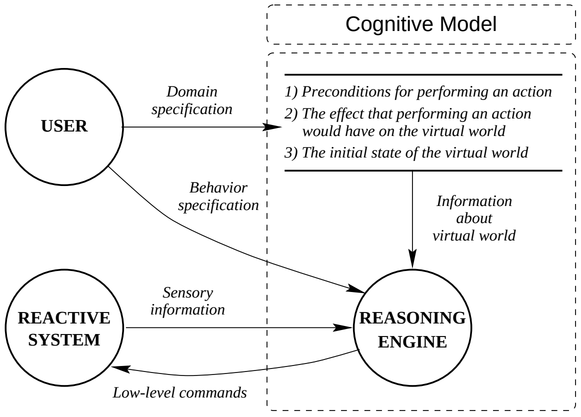

Cognitive Model
USER
Domain specification
1) Preconditions for performing an action
2) The effect that performing an action would have on the virtual world
3) The initial state of the virtual world
Behavior specification
Information about virtual world
REACTIVE SYSTEM
Sensory information
REASONING ENGINE
Low-level commands

Figure 8: Interaction between cognitive model, user and low-level reactive behavior system.

such as “swim forward” and “turn left”. The reactive system translates these commands down to the necessary detailed muscle actions. It also includes sensorimotor control loops that enable the agent to approximately satisfy commands, such as “go to a given position”. The reactive system furthermore acts as a fail-safe should the reasoning system temporarily fall through. In the event that the character cannot decide upon an intelligent course of action in a reasonable amount of time, the reactive layer continually tries to prevent the character from doing anything stupid, such as bashing into obstacles. Typical default reactive behaviors are “turn right”, “avoid collision” and “swim for your life”.

Even so, short of performing precise multiple forward simulations, it is impossible for his reasoning system to predict the exact position that a merman will end up after he executes a plan of action. A typical solution would be to re-initialize the reasoning engine every time it is called, but this makes it difficult to pursue long term goals as we are tossing out all the character’s knowledge instead of just the outdated knowledge. The solution is for the characters to represent positions using the IVE fluents that we described in Section 2.2.1. After sensing, the positions of all the visible objects are known. The merman can then use this knowledge to replan his course of action, possibly according to some long-term strategy. Regular fluents are used to model the merman’s internal state, such as his goal position, fear level, etc.

### 4.2.1 Undersea animations

The undersea animations revolve around pursuit and evasion behaviors. The hungry sharks try to catch and eat the mermen and the mermen try to use their superior reasoning abilities to avoid this grisly fate. For the most part, the sharks are instructed to chase mermen they see. If they cannot see any mermen, they go to where they last saw one. If all else fails, they start to forage systematically. Figure 9 shows selected frames from two animations.

The first animation verifies that because the shark is a larger and faster swimmer, it has little trouble catching merman prey in open water. In the second animation, we introduce some large rocks in the underwater scene and things get a lot more interesting. Now, when a merman is in trouble, cognitive modeling enables him to come up with short term plans to take advantage of the rocks and frequently evade capture. He can hide behind the rocks and hug them closely so that a shark has difficulty seeing or reaching him.

We were able to use the control structures of CML to encode a great deal of heuristic knowledge. For example, consider the problem of trying to come up with a plan to hide from the predator. A traditional planning approach will be able to perform a search of various paths according to criteria such as whether the path routes through hidden positions, or leads far from a predator, etc. Unfortunately, this kind of planning is prohibitively expensive. By contrast, the control structures of CML allow us to encode heuristic knowledge to help overcome this limitation. For example, we can specify a procedure that encodes the following heuristic: If the current position is good enough then stay where you are, else search the area around you (the expensive planning part); otherwise, check out the obstacles (hidden positions are more likely near obstacles); finally,

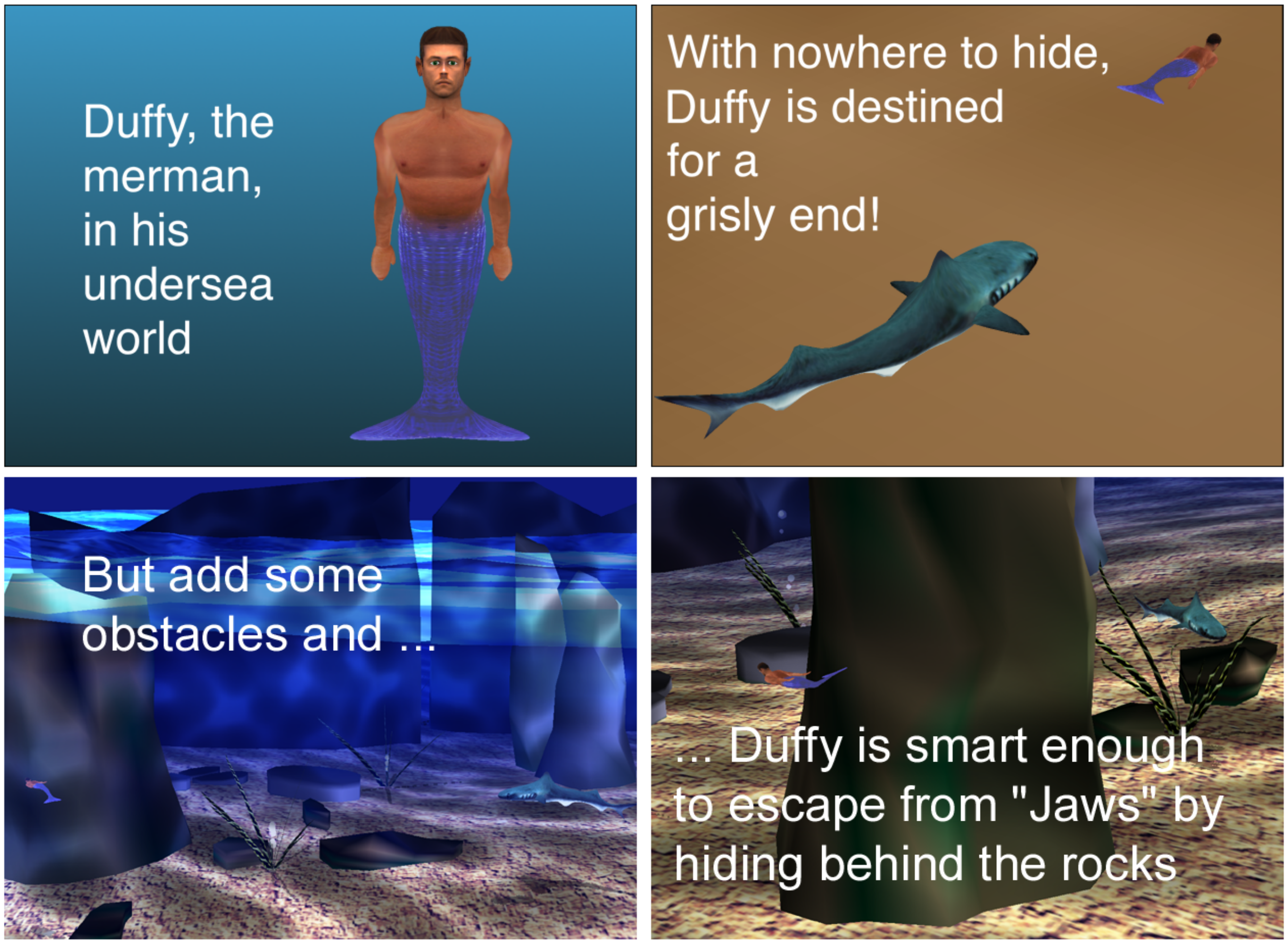

Duffy, the merman, in his undersea world
With nowhere to hide, Duffy is destined for a grisly end!
But add some obstacles and ...
... Duffy is smart enough to escape from "Jaws" by hiding behind the rocks

Figure 9: The undersea animations. Duffy the merman cleverly evades a predator shark.

if all else fails and danger looms, panic and flee in a random direction. With a suitable precondition for *pickGoal*, which prevents the merman selecting a goal until it meets certain minimum criteria, the following CML procedure implements the above heuristic for character $i$:

```
proc evade(i) {
    choose testCurrPosn(i);
    or search(i);
    or testObstacles(i);
    or panic(i); }
```

In turn, the above procedure can be part of a larger program that directs a merman to hide from sharks while, say, trying to visit the other rocks in the scene whenever it is safe to do so. Of course, planning is not always a necessary, appropriate or possible way to generate every aspect of an animation. This is especially so if an animator has something highly specific in mind. In this regard, it is important to remember that CML can also support detailed behavioral programming because it offers a full range of control structures that are customary in regular programming languages.

We used CML's control structures to make the animation "The Great Escape". This was done by simply instructing the merman to avoid being eaten, and whenever it appears reasonably safe to do so, to make a break for a large rock in the scene. The particular rock to which we want to get the merman to go proffers a narrow crack through which the merman, but not the larger-bodied shark, can pass. We wanted an exciting animation in which the merman eventually gets to that special rock with the shark in hot pursuit. The merman's *evade* procedure should then swing into action, hopefully enabling him to evade capture by finding and slipping through the crack. Although we do not specify exactly how or when, we have a mechanism to heavily stack the deck toward getting the desired animation. As it turns out, we got what we wanted on our first attempt (after debugging). However, if the animation that we desired remained elusive, we can use CML to further constrain what happens all the way down to scripting an entire sequence if necessary.

As an extension to behavioral animation, our approach enables us to linearly scale our cognitive modeling efforts for a single character in order to create multiple similarly-behaved characters. Each character will behave autonomously according to its own unique perspective of its world. In a third animation, we demonstrate that numerous cognitive characters may also cooperate with one another to try to survive in shark infested waters. We have specified that some mermen are brave and others are timid. When the timid ones are in danger of becoming shark food, they cry for help (telepathically for now) and the brave ones come to their rescue provided it isn't too dangerous for them. Once a brave rescuer has managed to attract a shark's attention away from a targeted victim, the hero tries to escape.

## 5 Conclusion

We have introduced the idea of cognitive modeling as a substantive new apex to the computer graphics modeling pyramid. Unlike behavioral models, which are reactive, cognitive models are deliberative. They enable an autonomous character to exploit acquired knowledge to formulate appropriate plans of action. To assist the animator or game developer in implementing cognitive models, we have created the cognitive modeling language CML. This powerful language gives us an intuitive way to afford a character knowledge about its world in terms of actions, their preconditions and their effects. When we provide a high-level description of the desired goal of the character's behavior, CML offers a general, automatic mechanism for the character to search for suitable action sequences. At the other extreme, CML can also serve like a conventional programming language, allowing us to express precisely how we want the character to act. We can employ a combination of the two extremes and the whole gamut in between to build different parts of a cognitive model. This combination of convenience and automation makes our cognitive modeling approach in general, and CML in particular, a potentially powerful tool for animators and game developers.

### 5.1 Future work

Cognitive modeling opens up numerous opportunities for future work. For example, we could incorporate a mechanism to learn reactive rules that mimic the behavior observed from the reasoning engine. Other important issues arise in the user interface. As it stands CML is a good choice as the underlying representation that a developer might want to use to build a cognitive model. An animator or other end users, however, would probably prefer a graphical user interface front-end. In order to make such an interface easy to use, we might limit possible interactions to supplying parameters to predefined cognitive models, or perhaps we could use a visual programming metaphor to specify the complex actions.

Cognitive modeling is a potentially vast topic whose riches we have only just begun to explore. In our prehistoric world, for instance, we concentrated on endowing the T-Rex with CML-based cognition. There is no reason why we could not similarly endow the Raptors as well. This would allow the animation of much more complex dinosaur behavior.[10] A lone Raptor is no match for the T-Rex, but imagine the following scenario in which a pack of cunning Raptors conspire to fell their large opponent. Through cognitive modeling, the Raptors hatch a strategic plan—the ambush! Based on their domain knowledge, the Raptors have inferred that the T-Rex's size, his big asset in open terrain, would hamper his maneuverability within the narrow passage under the arch. The leader of the pack plays the decoy, luring their unsuspecting opponent into the narrow passage. Her pack mates, who have assumed their positions near both ends of the passage, rush into it on command. Some Raptors jump on the T-Rex and chomp down on his back while others bite into his legs. Thus the pack overcomes the brute through strategic planning, cooperation, and sheer number. Coordinating multiple Raptors in this way would significantly increase the branching factor in the situation trees of the cognitive models. A solution would be to control them as intelligent subgroups. We could also exploit complex actions to provide a loose script that would specify some key intermediate goals, such as the decoy stratagem.

## Acknowledgments

We would like to thank Eugene Fiume for many helpful comments and for originally suggesting the application of CML to cinematography, Steve Rotenberg and Andy Styles at Angel Studios (Carlsbad, CA) and Steve Hunt for developing the low-level "Demosaurus Rex" API, Steven Shapiro and Hector Levesque for technical assistance on the situation calculus, and Jeffrey Tupper for technical assistance on interval arithmetic. JF thanks Ramesh Subramonian for his benevolent support during the final stages of this work.

[10] See Robert T. Bakker's captivating novel *Raptor Red* (Bantam, 1996).

## References

[1] J. Allen, J. Hendler, and A. Tate, editors. *Readings in Planning*. Morgan Kaufmann, 1990.

[2] D. Arijon. *Grammar of the Film Language*. Communication Arts Books, Hastings House Publishers, New York, 1976.

[3] N. I. Badler, C. Phillips, and D. Zeltzer. *Simulating Humans*. Oxford University Press, 1993.

[4] J. Bates. *The Role of Emotion in Believable Agents*. *Communications of the ACM*, 37(7), 1994, 122–125.

[5] J. Blinn. Where am I? What am I looking at? *IEEE Computer Graphics and Applications*, **8**(4), 1988, 76–81.

[6] B. Blumberg. *Old Tricks, New Dogs: Ethology and Interactive Creatures*. PhD thesis, MIT Media Lab, MIT, Cambridge, MA, 1996.

[7] B. M. Blumberg and T. A. Galyean. Multi-level direction of autonomous creatures for real-time environments. *Proceedings of SIGGRAPH 95*, Aug. 1995, 47–54.

[8] J. Funge. CML compiler. `www.cs.toronto.edu/~funge/cml`, 1997.

[9] J. Funge. Lifelike characters behind the camera. *Lifelike Computer Characters '98 Snowbird, UT*, Oct. 1998.

[10] J. Funge. *Making Them Behave: Cognitive Models for Computer Animation*. PhD Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1998. Reprinted in SIGGRAPH 98 Course Notes #10, Orlando, Florida.

[11] J. Funge. *AI for Games and Animation: A Cognitive Modeling Approach*. A.K. Peters, 1999.

[12] J. Funge. Representing knowledge within the situation calculus using interval-valued epistemic fluents. *Journal of Reliable Computing*, 5(1), 1999.

[13] B. Hayes-Roth, R. v. Gent, and D. Huber. Acting in character. In R. Trappl and P. Petta, editors, *Creating Personalities for Synthetic Actors*. Lecture Notes in CS No. 1195. Springer-Verlag: Berlin, 1997.

[14] L. He, M. F. Cohen, and D. Salesin. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. *Proceedings of SIGGRAPH 96*, Aug. 1996, 217–224.

[15] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31, 1997, 59–84.

[16] C. Pinhanez, K. Mase, and A. Bobick. Interval scripts: A design paradigm for story-based interactive systems *Proceedings of CHI'97*, Mar. 1997.

[17] N. Magnenat-Thalmann and D. Thalmann. *Synthetic Actors in Computer-Generated Films*. Springer-Verlag: Berlin, 1990.

[18] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[19] K. Perlin and A. Goldberg. IMPROV: A system for scripting interactive actors in virtual worlds. *Proceedings of SIGGRAPH 96*, Aug. 1996, 205–216.

[20] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*. Academic Press, 1991.

[21] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Proceedings of SIGGRAPH 87*, Jul. 1987, 25–34.

[22] R. Scherl and H. Levesque. The frame problem and knowledge-producing actions. *Proceedings of AAAI-93*, AAAI Press, Menlo Park, CA, 1993.

[23] J. Snyder. Interval analysis for computer graphics. *Proceedings of SIGGRAPH 92*, Jul. 1992, 121–130.

[24] X. Tu. *Artificial animals for computer animation: Biomechanics, locomotion, perception and behavior.* ACM Distinguished Dissertation Series, Springer-Verlag, Berlin, 1999.

[25] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. *Proceedings of SIGGRAPH 94*, Jul. 1994, 24–29.

[26] J. Tupper. Graphing Equations with Generalized Interval Arithmetic. MSc Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1996. See also GRAFEQ from Pedagoguery Software `www.peda.com`.