Cognitive modeling for games and animation explores the provocative but largely uncharted interface between computer graphics and artificial intelligence. That interface is now on the verge of explosive growth as a new breed of highly autonomous, quasi-intelligent graphical characters begins to populate the domains of production animation, game development, and multimedia content creation, as well as distributed multi-user virtual worlds, e-commerce, and other Web-enabled activities.

The modeling of graphical characters is a multifaceted endeavor, progressing from geometric modeling at the bottom of the hierarchy, through intermediate-level physics-based modeling, up to behavioral modeling. My research has sought to pioneer cognitive modeling as the hitherto absent but substantive apex of the character-modeling pyramid (see Figure 1). 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 physical and sensing actions. Cognitive models can also play subsidiary roles in controlling cinematography and lighting for computer games and animation.

Moreover, cognitive modeling addresses a challeng-
Figure 1. Cognitive modeling is on top of the modeling hierarchy but works with the lower levels to create a visually compelling experience. To demonstrate the entire modeling hierarchy in one application, Xiaoyuan Tu and I adapted an underwater world [4, 11]. A geometric model captures the form and appearance of the characters. A biomechanical model captures their anatomical structure, including internal muscle actuators, and simulates the deformation and physical dynamics of their bodies using a physics model. A behavioral-control model implements simpler brain functions and is responsible for motor, perception, and low-level behavior control. The "merperson's" reactive behavior system interprets intentions generated by the cognitive layer, then generates coordinated muscle actions controlling locomotion by deforming the body to generate propulsion-inducing forces against the virtual water.

Duffy, the merman, in his underwater world

But add some obstacles and …

With nowhere to hide, Duffy is destined for a grisly end!

… Duffy is smart enough to escape from "Jaws" by hiding behind the rocks
ing problem closely related to mainstream AI and robotics research. Imagine a virtual prehistoric world inhabited by a Tyrannosaurus rex (T-rex) and a pack of Velociraptors (Raptors). Suppose that, in small numbers and in open territory, the Raptors are no match for the T-rex. But a pack of cunning Raptors conspires to fell their much larger opponent. Through cognitive modeling, the Raptors hatch a strategic plan—an ambush. Based on their domain knowledge, they have inferred that the T-rex’s size, her most important asset in open terrain, would hamper maneuverability within a narrow passage under a stone arch. The leader of the pack plays the decoy, luring the unsuspecting opponent into the narrow opening. Her packmates, assuming positions near both ends of the passage, rush into it on command. Some Raptors jump on the T-rex, chomping on her back while others bite her legs. Thus the pack overcomes the brute through strategic planning, cooperation, and overwhelming numbers.

This ambush scenario is just one of the exciting possible applications of cognitive modeling. Before describing other applications, please note the balloon over the T-rex’s head. Its contents represent the character’s own internal mental model of its virtual world. It is this internal model I expressly refer to as a cognitive model. Foundational work in behavioral modeling has made progress toward self-animating characters that react appropriately—usually, though not always, in the best interests of their own survival—to perceived environmental stimuli. Without cognitive modeling, however, it would be difficult for both the game developer and the game player to instruct these autonomous characters so they are able to satisfy specific goals.

Cognitive models do not function in a vacuum. For example, the undersea world in Figure 1 represents an application in which all layers of the character-modeling pyramid must work cooperatively to create a visually compelling experience. Therefore, much of my research deals with how cognitive modeling can be perspicuously integrated into the modeling hierarchy.

**Herding Behavior**

I decompose cognitive modeling into two related subtasks: domain-knowledge specification and character instruction. This organization is reminiscent of the classic dictum knowledge + instruction = intelligent behavior from the field of AI that (by separating out knowledge from control) seeks to promote design modularity. Domain (knowledge) specification involves administering knowledge to the character about its world and how that world can change. Character instruction involves telling it how to try to behave within the world. These instructions can involve detailed step-by-step directions or, alternatively, provide high-level goals, so the character has to work out for itself how to behave in order to achieve them. I refer to this high-level style of instruction as “goal-directed behavior specification.” The important middle ground between these two extremes—step-by-step and goal-directed instruction—can also be exploited through the notion of “complex actions.”

As a simple concrete example of cognitive modeling, I offer a brief look at a goal-directed specification approach to synthesizing herding behavior. The example is from an application in which the T-rex in Figure 1 automatically formulates plans for driving Raptors out of its volcanic territory through a narrow passageway into a neighboring jungle territory [4, 5].

The carnage in the figure demonstrates why the Raptors have good reason to fear the larger, stronger T-rex should it come close. This is one piece of domain knowledge a game developer would need to give the T-rex about its world and about the Raptors’ reactive behavior (not unlike Craig Reynolds’s “boids” [10]). In total, the game developer needs to tell the T-rex the following four pieces of information:

- The Raptors are temporarily frightened if you approach them;
- Frightened Raptors run away from you;
- Unalarmed Raptors continue as they are; and
- You can’t walk through obstacles (see [4, 5] for the code I used to provide this and other knowledge to the T-rex).

Even in self-contained complex, dynamic virtual worlds, life should appear as thrilling and unpredictable to the character as it does to the human observer.
To get the T-rex to herd the Raptors toward a particular location, the developer has to give it the goal of getting more Raptors heading in the right direction than are currently heading that way. This goal, along with the supplied domain knowledge, enables the T-rex to plan its actions as if it were a smart sheepdog. It autonomously devises collision-free paths to maneuver in and around groups of Raptors in order to frighten them in the desired direction. This strategy enables the T-rex (which can plan up to six moves ahead of its current position) to quickly expel the unruly mob of Raptors from its territory. Longer-duration plans degrade real-time performance and are rarely useful, since the Raptors’ obstacle-avoidance routines mean the second and third assumptions in its domain knowledge are only approximations of their true behavior. A better strategy is “adaptive herding” through periodic re-planning.

Semantics of Cognitive Modeling

“The situation calculus” can be employed to provide a simple, powerful, and particularly elegant semantics for cognitive modeling. This AI formalism, invented in the 1960s by John McCarthy of Stanford University, describes “changing worlds” using sorted first-order logic. From the game developer’s point of view, the underlying theory can be hidden. To this end, I created the Cognitive Modeling Language (CML) to act as a high-level interaction language. CML’s syntax employs descriptive keywords with precise mappings to the underlying formal semantics of the situation calculus. Details of the situation calculus’s modern incarnations are well documented in numerous papers and books (such as [8]).

A situation represents a “snapshot” of the state of the world. Any property of the world that can change over time is known as a “fluent.” Primitive actions are the fundamental instrument of change in the ontology. The sometimes-counterintuitive term “primitive” serves only to distinguish certain atomic actions from “complex” compound actions. The possibility of performing an action in a given situation is specified by precondition axioms. Effect axioms give necessary conditions for a fluent to take on a given value after performing an action.

Unfortunately, effect axioms do not necessarily prescribe what remains unchanged when an action is performed and thus can lead to unexpected results. Enumerating all the “non-effects” could, however, require the addition of an exponential number of frame axioms. This burdensome requirement would be painstaking and error-prone; constantly considering all of them would slow any character’s reaction time. A solution is to assume the effect axioms enumerate all

To this end, I

...
possible ways the world can change. In 1991, Ray Reiter of the University of Toronto showed how this assumption can be incorporated through straightforward syntactic manipulation of the user-supplied effect axioms to automatically generate a set of successor state axioms [9].

**Sketch Plans with Complex Actions**

The actions, effect axioms, and preconditions I've described can be thought of as a tree (see Figure 2). The nodes of the tree represent situations, while effect axioms describe the characteristics of each situation. At the root of the tree is the initial situation; each path through the tree represents a possible sequence of actions. The precondition axioms mean that some sequences of possible actions are not possible. This winnowing of possible actions is represented in the figure by the pruned-away black portion of the tree. If some situations are desired goals, a game developer can use a conventional logic programming approach to automatically search the tree for a sequence of actions to get to the goal. The green nodes in the figure represent goal situations; various search strategies can be used to come up with an appropriate plan.

The problem with plotting long-range plans is that the search space grows exponentially in the depth of the tree. Much of the planning literature has sought to mitigate this problem with more sophisticated search algorithms, such as the well-known A* algorithm, and stochastic planning techniques. A game developer can push the idea of pruning the tree further by using complex actions to prune away arbitrary subsets of the search space. How the character has been programmed to search the remaining space is an important but independent problem for which all previous work on planning is applicable.

The right side of Figure 2 is an example of a complex action and its corresponding effect of reducing the search space to the tree's blue region (see [4, 5, 8] for more intuitive examples of complex actions and their definitions). The point I want to make here is that complex actions provide a convenient tool for encoding heuristic knowledge about a problem as a nondeterministic behavior outline. By “nondeterministic,” I mean multiple possibilities can be covered in one instruction, not that the behavior is random. This programming style allows many behaviors to be specified more naturally, simply, and succinctly at a much higher level of abstraction than would be possible otherwise. In general, the search space of a game character's options is still exponential, but pruning with complex actions allows the formulation of potentially longer plans, yielding characters that appear to the game player a lot more intelligent and a lot more fun and entertaining.

Various research teams have applied AI techniques to produce inspiring results with animated humans and cartoon characters [1, 2, 6, 11] (see Table 1). My own use of cognitive modeling is exemplified in three case studies [4, 5]. The first is the dinosaur-herding application discussed earlier. The next, suggested by Eugene Fiume of the University of Toronto, applies cognitive modeling to cinematography. One of my aims was to show how separating out the control information from the background domain knowledge makes it easier to understand and maintain controllers. The resulting camera controller is ostensibly reactive, making minimal use of planning, but it demonstrates that cognitive modeling subsumes conventional behavioral modeling as a limiting case. For example, the “undersea world” case study started out as the brainchild of Demetri Terzopoulos, also of the University of Toronto, from which we created an elaborate character animation to demonstrate how complex actions can be used to create an interactive story by giving characters a loose script, or “sketch plan.” At runtime, the undersea character uses its background knowledge to automatically decide for itself how to fill in the necessary missing details while still following the basic plot.

**Integrating Cognitive Modeling**

For applications like robotics and computer games involving interaction with the real world, it is important for programmers to be able to deal with a character's uncertainty about its world. Even in self-contained virtual worlds, life should appear as

---

**Table 1. Notable online resources related to cognitive modeling.**

<table>
<thead>
<tr>
<th>Web site</th>
<th>URL</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruce Blumberg</td>
<td><a href="http://www.media.mit.edu/~bruce">www.media.mit.edu/~bruce</a></td>
<td>Papers, teaching materials</td>
</tr>
<tr>
<td>Cognitive Robotics Group</td>
<td><a href="http://www.cs.toronto.edu/cogrobo">www.cs.toronto.edu/cogrobo</a></td>
<td>Golog source code, papers, teaching materials</td>
</tr>
<tr>
<td>John Funge</td>
<td><a href="http://www.dgp.toronto.edu/~funge">www.dgp.toronto.edu/~funge</a></td>
<td>Animations, CML source code, papers, tutorials</td>
</tr>
<tr>
<td>Henry Kautz</td>
<td><a href="http://www.research.att.com/~kautz/">www.research.att.com/~kautz/</a></td>
<td>Papers, source code, teaching materials</td>
</tr>
<tr>
<td>John Laird</td>
<td>ai.eecs.umich.edu/people/laird</td>
<td>Papers, teaching materials</td>
</tr>
</tbody>
</table>
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 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. It usually makes far more sense for a character to decide what to do using a simplified cognitive model of its world, sense the outcome, and perform follow-up actions if things don’t turn out as expected.

The upper-left quadrant of Figure 3 depicts the traditional “sense-think-act” cycle advocated in much of the literature and widely used in computer games and animation. During every such cycle, the character senses its world, decides what (if any) action to perform next, then performs it. For noninteractive animation, this cycle works well and is conceptually simple. Unfortunately, for real-time applications, including computer games, the cycle forces characters to make split-second decisions about what may be highly complex situations. Therefore, I propose the alternative taxonomy depicted on the figure’s lower-left quadrant—the tight “sense-react-act” cycle vital for creating lively and reactive characters. It also allows more thoughtful deliberation to be spread over many cycles.

This new architecture involves many challenges. From an implementation perspective, the deliberative behavior should be an independent process that can be suspended, interrupted, even aborted, depending on other real-time constraints and the changing state of the world. Fortunately, this process is relatively straightforward; complications arise from deeper technical issues. In particular, if the character is “thinking” over a period of time, there should be some way to represent its increasing uncertainty about its world. Previous approaches to the problem in AI proposed the use of “possible worlds” to represent what a character knows and doesn’t know. Unfortunately, if the application includes a set of relational fluents whose values may be learned through sensing, the programmer has no choice but to list a potentially exponential number of initial possible worlds. Things get more complicated with functional fluents whose range is the real numbers, since we cannot list the vast number of possible worlds associated with uncertainty about their values.

Therefore, I propound the practicable alternative of using intervals and interval arithmetic to represent and reason about uncertainty [5]. Specifically, I introduce the notion of “interval-valued epistemic” (IVE) fluents to represent a character’s uncertainty about the true value of the variables within its world. The intuition behind this approach is in the top-right quadrant of Figure 3 whereby sensing corresponds to narrowing uncertainty intervals that grow over time until sensing “collapses” them back to the their true value. Panel (d) is my two-layer concrete instantiation of the new architecture.

2IVE fluents represent uncertainty intervals about time-dependent variables. They do not represent and are unrelated to time intervals of the sort used in the underlying semantics of various temporal logics.
inputs at a set frame rate, then re-planning, is that it is wasteful if previously sensed information is still usable. Worse, a character might not be re-planning often enough at critical times. A game programmer would therefore like to be able to create characters that sense asynchronously and re-plan only when necessary. The width of an IVE fluent measures the degree of uncertainty, possibly indicating unacceptably outdated information. The first concrete instantiation of this architecture was realized in [4, 5]. The bottom-right quadrant of Figure 3 shows the system consisted of just two levels. The low-level reactive-behavior system Xiaoyuan Tu of the University of Toronto and I used was (with minor modifications) the artificial life simulator she developed (under the supervision of Demetri Terzopoulos) [12].

Learning
Sensing allows a character to acquire for itself knowledge about the current state of its world. Since one of the major bottlenecks in cognitive modeling is defining and refining a character's domain knowledge, it would be extremely useful for a game developer, as well as a game player, if the character could automatically acquire knowledge about its world's dynamics. Such knowledge acquisition is studied in the field of machine learning. A character should also be able to endeavor to learn not only about how its world behaves but about how other characters, including human avatars, behave in it. The character could even seek some measure of self improvement by learning about its own behavior. For example, I envisage a hierarchy of reasoning models (lower-left quadrant of Figure 3) a character might use to ponder its world at increasing degrees of sophistication. This hierarchy could include even a post-game-analysis level to develop better strategies for the next time the game is played.

Ideally, some mechanism should exist through which the knowledge obtained via deliberation at a higher level can be compiled down into one of the underlying representations. This process should eventually propagate all the way down to the lowest reactive level where knowledge can be represented as simple, fast-executing rules. Moreover, with the advent of the Internet, there is no reason why all reasoning modes in a particular game have to be on the same machine. A game console might communicate with online processing centers that automatically generate new behavior rules as needed.

In contrast, programming a character to learn simple things about its world is relatively straightforward. For example, it is straightforward to program characters to autonomously map out all the obstacles by exploring their world in a preprocessing step. To help them acquire higher fidelity, knowledge researchers have turned to increasingly sophisticated machine-learning techniques. One notable approach is based on the Soar AI architecture, a general cognitive architecture for developing systems exhibiting intelligent behavior [8].

3Historically, Soar stood for State, Operator, And Result, because all problem-solving in Soar is regarded as a search through a problem space in which an operator is applied to a state to get a result, though it is no longer regarded as an acronym and is no longer written in upper case.
This approach enables a character to learn the knowledge it needs by first watching an expert complete the given task. By way of analogy with motion capture, this process is referred to as “behavior capture.” It was initially designed for developing intelligent air-combat agents for military applications. More recently, it has been applied to a number of computer games, including Quake II, producing deathmatch “Soarbots,” some prompted by voice commands (see Figure 4).

The most important topic of behavior-learning involves an approach inspired by ethology, rather than the more traditional AI outlook behind most other cognitive modeling work [3]. For example, Bruce Blumberg of the MIT Media Lab and his team are building a virtual dog to determine how closely its behavior can be made to resemble that of a real dog. In particular, the team wants it to be able to learn the kinds of things real dogs are capable of learning. Moreover, they want to be able to train it using a standard animal-training technique called “clicker training.”

The Future of Cognitive Modeling

Cognitive researchers have only begun to embrace a vision of the untapped synergy between AI and computer graphics. In the next two to five years, the potential for communication among cognitively enabled characters should provide fertile ground for research into developing characters capable of sophisticated cooperative group behaviors. Naturally, one of the key factors fueling interest in such advanced modeling techniques as cognitive and physics-based modeling is the rapid pace of hardware development. I am especially excited about the emergence of powerful new game consoles (see Figure 5) promising to invigorate each layer of the modeling hierarchy to yield characters with unprecedented levels of interactivity and physical realism.

REFERENCES


JOHN FUNGE (funge@dgp.toronto.edu) is a research scientist in Sony Computer Entertainment Americ in Foster City, Calif.