

Explicit Knowledge Programming for Computer Games

Andreas Witzel

ILLC, University of Amsterdam
Plantage Muidergracht 24
1018TV Amsterdam, Netherlands
and CWI, Kruislaan 413,
1098SJ Amsterdam, Netherlands
awitzel@science.uva.nl

Jonathan Zvesper

ILLC, University of Amsterdam
Plantage Muidergracht 24
1018TV Amsterdam, Netherlands
and CWI, Kruislaan 413,
1098SJ Amsterdam, Netherlands
jonathan.zvesper@gmail.com

Ethan Kennerly

Interactive Media
School of Cinematic Arts
University of Southern California
University Park, LUC-310B
Los Angeles, CA 90089-2211
kennerly@finegamedesign.com

Abstract

The main aim of this paper is to raise awareness of higher-order knowledge (knowledge about someone else's knowledge) as an issue for computer game AI. We argue that a number of existing game genres, especially those involving social interaction, are natural fields of application for an approach we call *explicit knowledge programming*. We motivate the use of this approach, and describe a simple implementation based upon it. A survey of recent literature and computer games illustrates its novelty.

Introduction

Higher-order knowledge, i.e. knowledge about (someone else's) knowledge, is important in social interaction.¹ That importance is well established in game theory (Brandenburger 2007). We will point out its relevance to computer games that incorporate simulations of social interaction, by which we mean interaction with artificial agents and non-player characters (NPCs). The most obvious examples of social interaction occur in computer role-playing games (RPGs), interactive fiction (IF) and life simulation games (such as *The Sims*TM), but we will also show examples from other genres.

In this paper we do not make any significant technical contribution, but we will suggest one general methodology, which we call *explicit knowledge programming*, for implementing the ideas that we will describe. It involves an approach founded in epistemic logic with case-dependent restrictions of the logical language to remain tractable.

As a side note, we think that higher-order *desires* as well as beliefs are important, and indeed the interplay between the two: for example, in an interaction between *A* and *B*, it can be significant if *A* believes that *B* desires that *A* believe such-and-such. Belief-desire-intention (BDI) architectures (Rao & Georgeff 1995) currently do not accommodate this

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

¹In this paper we talk pretty much interchangeably about *higher-order knowledge* and *higher-order belief*. There are philosophical subtleties at stake here, but they are not relevant for our purposes, for which it is sufficient to stipulate that "knowledge" refers to true belief. They both have in common the "higher-order" aspect and that is what is crucial. We use the adjective *epistemic* to mean "of (or about) beliefs (or knowledge)".

kind of interactive (i.e. truly multi-agent) phenomena. Since beliefs and knowledge are, from a logical point of view, better understood than desires, we suggest to start by focusing on the former, and that is what we do in the rest of this paper.

Motivation

Human beings are, on the whole, social animals: most of us derive much interest, enjoyment and drama from interacting with other people. One important feature recognized by psychologists is the so-called *theory of mind*, that is the ability to represent and to empathize with the mental states and attitudes of those around us. A recent and popular theory argues that the absence of any theory of mind is what causes such obstacles for those people affected with autism spectrum disorders to interact profitably with those around them (Baron-Cohen 1995).

The standard empirical test for a theory of mind, which develops in most people around the age of four, is a test of the subject's ability to represent and "correctly" form higher-order beliefs, that is to model other people as having their own model of the world in their head, which in turn contains a model of your model. This recursive notion can seem surprising, even paradoxical, however the fact is that most people's behavior is informed by some understanding of higher-order beliefs.

Much of the enjoyment of a number of multiplayer games actually depends upon higher-order beliefs or knowledge. Cluedo, Diplomacy and Poker are examples of such *knowledge games* (Ditmarsch 2000); each of them illustrates in a different way that it can be enjoyable to reason about the beliefs of others, particularly about the beliefs that they might have about your beliefs, or others' beliefs. If these games were to be played against artificial opponents, with the aim that they be enjoyable, a very natural approach would be to create an opponent who models the player, including the beliefs the player has about the opponent, etc.

So it can be on the one hand "natural", and on the other hand enjoyable, to reason about higher-order knowledge. We therefore propose that this line of reasoning be taken seriously, and that providing some facility for higher-order reasoning to NPCs would enhance the aim that many games have: being enjoyable simulations of human interaction.

The kinds of interactive situations that depend on higher-order knowledge vary greatly, from trivial to subtle. The

following example is so trivial that it may seem strange: if I've just told you something, then I won't tell you again since I know that you know it, that you know that I know that you know it, etc. (indeed it creates what is called *common knowledge*). An example that follows a similar principle is the following: If Ann knows that Bob knows that the enemy is attacking Bob's base, then she will not tell him, unless she wants him to know that she knows (and may be coming to support him).

Increasingly, networked computers and game consoles have lead to a rise in the number and scale of multiplayer games. However, we do not believe that this will diminish the need for social AI in games, because in a reasonably deep virtual world, there are always "boring" roles, such as quest-givers, merchants, henchmen, and thugs. These are a few of the roles we have in mind when advocating the use of some simple higher-order knowledge reasoning.

We will return to some more interesting examples and potential applications later. For now, we hope it is intuitively clear that a virtual world could be enhanced by keeping track of what its virtual inhabitants (could be said to) believe or know, and to provide a programming interface to access these beliefs and knowledge. Then a programmer can use these high-level notions in order to script behaviors of NPCs, or other relevant parts of the virtual world.

How should this be done? In the next section we propose one approach to realize these ideas.

Programming with Knowledge

Our approach is based around epistemic logic, a research field with its origins in philosophy (Hintikka 1962). We propose to use a formal language for epistemic statements, including higher-order ones. The formulas of such a language might for example be built recursively from *atoms*, which are non-epistemic facts, by using propositional connectives, like \vee (or) and \neg (not), and knowledge operators K_a , one for each agent a being simulated. We will call these *epistemic formulas*. So, for example, if p were an atom, then $K_a K_b p$ could be an epistemic formula with the intended reading " a knows that b knows that p ".

The idea, which we call *explicit knowledge programming*, is to make these statements available at the programming level, for example as conditions in `if` clauses. These statements are evaluated by a *knowledge module* that processes those events in the world that affect the knowledge state of its inhabitants. It may be implemented in the programming language itself, and thus does not necessarily increase its expressivity. However, the modular approach increases succinctness and flexibility, makes it possible to develop and verify the epistemic processing of the program separately, and essentially gives the NPC scripter a "black box", so that she can use directly the familiar notions of belief or knowledge in her program. At the same time, the knowledge module is firmly grounded in a theoretical framework which gives a formal meaning to these notions.

It is important to note that we are *not* proposing to make available *every* formula that can be built from any arbitrary combination of atoms, connectives and knowledge operators. Rather, for any given concrete application, a certain

subclass of formulas needs to be identified as relevant. In this way the implementation can remain tractable. For example, in the case of NPCs like those mentioned in the previous section it is obviously not a question of implementing a "full" human-like knowledge module. But for example a merchant might have a restricted set of "interests", and only be interested in very specific kinds of knowledge.

In order to make our discussion more concrete, we present now a simple and preliminary implementation of explicit knowledge programming. This is discussed in previous work (Witzel & Zvesper 2008), where the implementation is also proved to be sound with respect to a formal notion of knowledge defined on the level of the underlying process calculus. We will briefly review this work in the following.

We wrote a knowledge module that is instantiated for each process. In our particular implementation, the events that were used as inputs to the knowledge module were always synchronous communications between two of the processes concerning the values of some bits. (In general though, an event can be anything which would have epistemic effects.) The idea is to pass to the knowledge module of a process a the events that a "observes".

The queries to which the knowledge module can respond are epistemic formulas. We used a formal language with atoms $p_{x_0}, \neg p_{x_0}, p_{x_1}, \neg p_{x_1}, \dots$ for each of the bits x_0, x_1, \dots , and a knowledge operator K_a, K_b, \dots for each of the processes a, b, \dots ² Then, as an example, the formula $K_c K_b \neg p_{x_2}$ means that process c knows that process b knows that x_2 has the value 0. If the knowledge module of process a were passed this formula it could respond by saying "yes", "no" or "don't know". If the module were to respond "yes", then this should be interpreted as (the agent which is modeled by process) a knowing that c knows that b knows that x_2 is 0.

Even with a simple implementation, it was desirable to prove that it was in some sense "correct". Thus we used a formalism from the literature on epistemic logic, namely Kripke models. The argument for correctness of the implementation then proceeds in two steps, which can roughly be stated as follows:

- Argue that a particular model represents faithfully the intuitive situation which we intended to capture.
- Prove that knowledge formulas are evaluated in the same way by process a after the sequence of events σ as they are by agent a in the model after the same sequence of events.

One criticism that one can make of using a Kripke model formalism as an intermediate step is that it is a formalism that itself can appear unintuitive. However, we know of no more philosophically grounded and mathematically robust formalism with which to work in the context of reasoning about higher-order knowledge. (In order to deal with various phenomena like so-called *explicit belief*, or inconsistent beliefs, many other models have been proposed, but these are all essentially refinements or variations of Kripke mod-

²Note that here we are not using the richer language that could be built using also the connectives \neg and \vee mentioned above.

els – see (Meyer & van der Hoek 1995, Sections 2.4 to 2.13) for a selective survey.)

The Kripke model for our particular implementation resembled an *interpreted system* from (Fagin *et al.* 1995). It was not our aim to implement an entire interpreted system, which in this case is an infinite structure. Even if it is finitely representable, we might only be interested in certain parts of it.

In general, an implementation can be simplified by considering only a subclass of the formulas of the full logical language. As it happened, for the particular implementation we had in mind (a distributed implementation of an algorithm for eliminating dominated strategies in strategic games), it was only necessary to consider formulas from the very simple epistemic language that we have described.

In the case of simulating *human* agents that we are interested in here, the limits to human cognitive faculties should be taken into account. So for example, it presumably would not make sense to allow as queries to the knowledge module epistemic formulas involving complex iterations, like: “Ann believes that Bob believes that Carl doesn’t believe that Ann believes that Derek believes that it’s raining”. The specification of a knowledge module would include a description of the epistemic formulas that it can evaluate.

Relation to Existing Work

We will briefly review the current state of the art with respect to epistemic modeling in computer games, both in existing games and in (academic) research.

Existing Games

The state of the art in commercial computer games is not easy to judge, since computer game companies are not very keen on publishing the details of their AI implementations. So if one doesn’t want to rely on enthusiastic slogans from marketing departments, then the best sources of information about computer game AI are private web pages like (Woodcock 2007) where observations and analyses from playing, interview quotations, and possibly the website creator’s own knowledge and experience as a game AI programmer are carefully collected and presented. For an extensive list of online resources, see (Reynolds 2007).

From these resources it becomes evident that epistemic reasoning is definitely not in the focus of existing computer game AI, and we did not find any mention of higher-order reasoning. For example, the highly acclaimed Radiant AI engine is used in the RPG *The Elder Scrolls: Oblivion*TM (Bethesda Softworks 2006) to make the game more lifelike. The following quotation is taken from an interview (Wikipedia 2006) during the testing phase of the game AI:

One [non-player] character was given [by the testers] a rake and the goal “rake leaves”; another was given a broom and the goal “sweep paths,” and this worked smoothly. Then they swapped the items, so that the raker was given a broom and the sweeper was given the rake. In the end, one of them killed the other so he could get the proper item.

Obviously, the characters didn’t mutually know their interests, or they couldn’t make use of that knowledge. Without seeing the implementation, it is difficult to make suggestions as to how exactly one might build in a knowledge module, and, as mentioned in the beginning, a whole architecture incorporating beliefs and desires is formally involved; however, the Radiant AI engine does seem to have some kind of goal-oriented behavior rules,³ and it is very possible that epistemic statements would find a natural place in them.

To us it seems natural that one would use a logic-based approach in order to effectuate epistemic reasoning. Yet references in these directions are scarce. In (Mäkelä 2001), it is suggested to use logic for NPC scripting; however, higher-order epistemic reasoning is not considered, and that article is purely programmatic and apparently the ideas have not been pursued further by the authors.

The clearest statement promoting the use of epistemic reasoning comes from the famous IF writer Emily Short (Short 2007):

Abstract Knowledge. One of the artificial abilities we might like to give our NPCs, aside from the ability to wander around a map intelligently and carry out complex goals, is the ability to understand what they are told: to keep track of what items of knowledge they have so far, use them to change their plans and goals, and even draw logical inferences from what they’ve learned.

It is not clear whether this refers to higher-order knowledge, or whether “abstract” just is meant to imply that the implementation should be generic and encapsulated in something like a knowledge module; in any case, the currently existing IF implementation of such ideas (Eve 2007) is restricted to pure facts and does not include any reference to the possibility of higher-order knowledge.

An interesting example of a game devoted to small-scale social interaction is *Façade*, a dialog-based graphical version of interactive fiction with a detailed plot that revolves around an evening in a small group of friends. *Façade* is grounded in academic research, and its authors use intricate techniques to interactively generate the plot, including a behavior language (Mateas & Stern 2004). That language allows to specify the behavior of the NPCs in a very flexible and general way, but does not include facilities for explicitly dealing with knowledge states. We do not suggest that this particular game would necessarily be *improved* if our approach of explicit knowledge programming were adopted, but we do think that it would make a natural addition to this language, and one that should make the programmer’s job more straightforward.

Research

Again, where knowledge is considered, the concern seems to be exclusively *domain knowledge*, or knowledge about facts in the game world, as in (Ponsen *et al.* 2007;

³Note, however, that the retail version of the game may be less sophisticated; as one referee pointed out, “the version of Radiant AI that ended up in the game certainly does not have the capabilities that Bethesda aimed for.”

Spronck *et al.* 2006). A more general approach of using agent programming languages to script NPCs (e.g. (Leite & Soares 2006)) inherits the epistemic reasoning facilities of such languages – which tend to focus on facts. The closest in spirit to higher-order modeling are attempts to detect the general attitude of the human player (for example, aggressive or cautious) and to adjust the game AI accordingly. But we could find no references to explicit higher-order epistemic modeling.

The ScriptEase system (Cutumisu *et al.* 2007) is an academic approach to NPC scripting, which was motivated by the insight that the scripting process needs to be simplified. It provides a graphical interface for generating behaviors of characters in a commercial RPG. However, knowledge statements to steer the behavior are not considered.

A very interesting approach, described in (da Silva & Vasconcelos 2006), uses deontic logic to specify NPC behavior in a rule-based fashion. While epistemic issues are not considered there, a fusion of these two aspects could provide a highly suitable system for scripting believable social agents.

Some literature on higher-order reasoning in multi-agent systems that does *not* focus on computer games is also very relevant. In (Dragoni, Giorgini, & Serafini 2002), the specific problem of agent *communication* is studied, in which agents weigh costs against expected benefit of communication. The authors point out the importance of using higher-order reasoning, in the form of beliefs about beliefs, when agents make such assessments. Their particular interest is in formal representation of belief “*abduction*”. We do not consider abductive reasoning here, but we recognize that it is also important in our settings.

We also note that higher-order reasoning is discussed in (Yin *et al.* 2000) in the context of a Petri Net method for designing “intelligent team training systems”. The authors suggest that using Petri Nets can help to overcome tractability issues in epistemic reasoning. However, they note that communication, an important ingredient in the kind of social interaction we wish to simulate, “is more complicated than Petri Nets can represent”. We do not consider the Petri Net formalism further, but if progress is made in this area it could be of relevance.

Several rich platforms for multi-agent programming, including BDI architectures, have been proposed (see e.g. (Bordini *et al.* 2006) for a recent overview). While these often do provide for explicit knowledge operators, they are never higher-order. So these platforms allow for the use of conditions of the form “If the agent knows such-and-such then ...”, but not “If agent *A* knows that agent *B* knows that ...”.

Potential Applications

We will try to illustrate how our approach could enhance potential or actual computer games.

Knowledge games, as mentioned above, are an obvious application area for explicit knowledge programming, if they are to be implemented in a computer game version with intelligent computer-controlled opponents. To what extent a knowledge module here should implement the complete

theoretical epistemic model, depends on performance considerations as well as a good balance to make the opponents challenging yet not super-human. We will not focus on such games here.

As mentioned before, RPGs and life simulation games naturally try to simulate realistic social interaction. Therefore, any real-life social interaction situation involving knowledge can be viewed as potential application, if one wants to script the required behavior rules into a virtual world.

For example, in real life the following behaviors might be observed in situations where Ann would get an advantage from lying to Bob:

- if she knows that he doesn’t know the truth then she might indeed lie;
- if she knows that he does know the truth then she usually won’t lie;
- if she doesn’t know whether he knows the truth then her decision may depend on other circumstances.

Catching the Thief

To be a bit more concrete and give an example set in an actually existing computer game, we consider Thief: The Dark Project™ (Eidos Interactive 1998). This game involves an interesting special case of “social interaction”: the player is a thief, and as such, the best tactic most of the time involves remaining undetected and avoiding confrontation.

While there are more obvious and ubiquitous applications for higher-order reasoning generally in RPGs and life simulation games, even in this unusual social interaction setting it is thus crucial to keep track of who has seen or heard whom and who knows it, both for the player and for plausible and challenging NPCs. This has obviously been taken care of in the commercial game, but we can nicely illustrate our approach in this setting.

Imagine an NPC *G* that embodies a guard, hostile to the thief *T*. Among his behavior rules could be something similar to the pseudo-code in Listing 1. How the knowledge or beliefs used there can come about may vary: by causing or hearing noise, seeing the other one from behind, or facing each other. When scripting these behaviors, the programmer need not worry about this – it is the job of the knowledge module to take care of maintaining and updating the knowledge states, taking into account whichever events occur in the virtual world.

Once it is established exactly what kinds of knowledge formulas need to be used, and what kinds of events can take place in the virtual world, one can go about designing and implementing the knowledge module for the specific application in question. It depends on the class of formulas and on the events how straightforward this implementation will be; but it may well, as in our exemplary implementation of explicit knowledge programming, turn out to be very efficient.

Adding Credence to Assassin’s Creed

Assassin’s Creed™ (Ubisoft 2007) presents another crisp case for higher-order beliefs. In each city, Altair (the player

Listing 1: Pseudo-code for Thief: The Dark Project

```
if guard.believes(thief in castle):
    if guard.believes(not thief.believes(guard in castle)):
        guard.ambush(thief)
    elif guard.believes(not thief.believes(guard.believes(thief in castle))):
        if not guard.believes(alarmed):
            guard.alarm(inconspicuously)
        else:
            guard.attack(thief)
    else:
        if not guard.believes(alarmed):
            guard.alarm(quickly)
        else:
            guard.attack(thief)
```

character) frequently has the optional objective to Save a Citizen. To do this, Altair kills the guards that are accosting the citizen. Vengeful guards nearby comment on the dead allies, which notifies Altair that they know a murder has occurred. Because Altair is present, the guards infallibly assume that Altair is the killer.

To demonstrate higher-order beliefs, suppose that a guard partitions the crowd into bystanders and suspects, by observing who is armed. The guard observes the speech and body language of each bystander (technically a voice over callout and animation state) to infer the belief of the bystander. As soon as the guard observes a bystander who is responding to a murderer, that guard claims the bystander's target is the killer. The guard then observes the body language (technically the high or low status) of Altair (or another prime suspect) to infer if the suspect believes that the guard is onto him.

Listing 2 shows pseudocode of this higher-order analysis, written for clarity. Admittedly, such a proposal would require further design, yet the example illustrates gullible

guards, which a crafty assassin may delight in deceiving.

Summary

We have described a modular approach to adding (higher-order) knowledge operators to scripting languages for NPCs in various kinds of games. We have argued and given examples to show that this systematic approach would help script plausible or entertaining simulations that involve social interaction, where that last term has a broad interpretation. We have surveyed existing work, in industry and in academic research, and found that higher-order knowledge has not so far been discussed or implemented in the context of computer games.

In games, the goal of epistemics may not necessarily be verisimilitude, because the ultimate objective is entertainment. For analogy, car racing games such as Burnout Paradise™ (Electronic Arts Inc. 2008) obviously employ rigid body dynamics for traction and collision in a way that veers from verisimilitude and toward excitement. Game physics engines have evolved from algorithms originally de-

Listing 2: Pseudo-code for Assassin's Creed

```
if guard.believes(murder_happened):
    suspects = guard.filter_by_belief(armed, crowd)
    bystanders = guard.filter_by_belief(not armed, crowd)
    for bystander in bystanders:
        if guard.believes(bystander.believes(murder_happened)):
            for suspect in suspects:
                if guard.believes(bystander.believes(suspect is killer)):
                    guard.add_belief(suspect is killer)
                    break
            if guard.believes(suspect is killer):
                if guard.believes(suspect.believes(\
                    guard.believes(suspect is killer))):
                    guard.shout(suspect is killer)
                    guard.attack(suspect)
            else:
                guard.whisper(suspect is killer)
                guard.ambush(suspect)
```

