Chapter 4

Epistemic reasoning in computer games

4.1 Introduction

Higher-order knowledge, that is, knowledge about (someone else’s) knowledge, is important in everyday social interaction.\textsuperscript{1} That importance is well-recognized in logic and game theory (insofar as these disciplines extend to everyday life), see, e.g., the work by van Ditmarsch [49] and by Brandenburger [31]. In this chapter, we 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\textsuperscript{TM}), but we also show examples from other genres.

We substantiate one example using what we call explicit knowledge programming, which we have proposed in [155] and used in Chapter 3 in this dissertation. It consists in implementing a well-defined restricted epistemic logic within a knowledge module, in order to provide epistemic statements on the programming language level and thus give programs access to the knowledge they can be said to have from a modeler’s point of view.

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 [122] currently do not accommodate this 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

\textsuperscript{1}In this chapter we talk 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. 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)”.

77
Chapter 4. Epistemic reasoning in computer games

the rest of this chapter.

4.1.1 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 [119], that is, the ability to represent and to empathize with the mental states and attitudes of those around us. A recent and popular theory by Baron-Cohen [15] argues that the absence of any theory of mind is what causes such obstacles for people affected with autism spectrum disorders to interact profitably with those around them.

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 the subject’s 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 many multiplayer games actually depends on dealing with, and possibly exploiting, higher-order beliefs or knowledge. Cluedo, Diplomacy and Poker are examples of such knowledge games, as van Ditmarsch [49] has called them; 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. This is known within the logic community, in fact van Ditmarsch [49] has given detailed formalizations of the epistemic mechanisms involved in real-world games of knowledge and suspicion such as Cluedo. However, so far there has not been much focus on putting such formalizations to work to support a computer simulation of a virtual game world, and in actual computer games these mechanisms have been ignored. We therefore propose to take this line of reasoning seriously, and argue 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 in a normal social context that it almost seems not worth mentioning: if I know that you know something, I won’t tell you about it—unless I want you to know that I know. Translated into a computer game context, if Ann knows that Bob knows that the enemy is attacking Bob’s base, then she will not tell him that the enemy is
attacking his base, unless she wants him to know that she knows (and is coming to support him). So even this seemingly trivial example illustrates the importance of higher-order reasoning in social interaction.

Increasingly, networked computers and game consoles have led to a rise in the number and scale of multiplayer games. However, we do not believe that this will diminish the need for social artificial intelligence 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 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. A programmer can then use these high-level notions in order to program (in this context also called to script) behaviors of NPCs, or other relevant parts of the virtual world.

4.1.2 Plan of the chapter

This chapter is built up as follows. In the next Section 4.2, we review our basic approach to realizing these ideas. Then, in Section 4.3, we survey both actual computer games and related research with respect to higher-order reasoning and come to the conclusion that it is largely absent. We then give examples for possible applications of our ideas in Section 4.4, and look at a prototype implementation for one of them more closely in Section 4.5. Finally, Section 4.6 concludes.

4.2 Programming with knowledge

Our approach, proposed in [155], is based around epistemic logic, and since we have used it in Chapter 3 in this dissertation, we recall it here only briefly.

The approach, which we call explicit knowledge programming, involves making epistemic formulas available at the level of a programming language, 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 he can directly use the familiar notions of belief or knowledge in his 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 and meet efficiency requirements, which are particularly strict in the case of real-time applications such as computer games.

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” (see Section 4.6 for some more discussion on this). In addition to these human limitations, in the case of NPCs like those mentioned above, for example, a merchant might have a very restricted set of “interests”, and only be interested in very specific kinds of knowledge. In any case, the specification of a knowledge module ultimately includes a description of the epistemic formulas that it can evaluate.

In [155] and in Chapter 3 in this dissertation, we were dealing with distributed systems, where a knowledge module is instantiated for each process. We showed the implementation to be sound with respect to the formal notion of knowledge ultimately defined on the level of the underlying process calculus, CSP. Even with the simple implementations we obtained, it was desirable to show that it is in some sense “correct”. To this end, we used an epistemic logic formalism based on Kripke semantics to model the occurring situations. The 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 $\sigma$ as they are by agent $a$ in the model after the same sequence of events.

While this idea of formal grounding remains the same, in this chapter we are dealing with a centralized framework. Since in a computer game there is a central place where the game world is simulated (and where other models such as a physics engine are already present), only one epistemic model needs to be maintained, even in a scenario with several agents. In this sense, this setting is closer in spirit to the typical viewpoint of epistemic logic, which is a central modeler’s viewpoint. Rather than maintaining a separate model inside each agent, our central model is fed with the events occurring in the game world, and queried by the agents’ programs. Note, however, that this design decision is not fundamental, since central models and internal models of individual agents correspond very
4.3. Related work

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

4.3.1 Existing games

The state of the art in commercial computer games is not easy to judge, since computer game companies are not very interested in publishing the details of their artificial intelligence (AI) implementations. So if one does not want to rely on enthusiastic slogans from marketing departments, then the best sources of information about computer game AI are private web pages like the one by Rabin [120], 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 overview of online resources, see the compilation by Reynolds [125].

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™ by Bethesda Softworks [23] to make the game more lifelike. The following quotation is taken from an interview [153] 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 did not mutually know their interests, or they could not 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,\(^2\) 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. Mäkelä [96] has 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.

The clearest statement promoting the use of epistemic reasoning comes from the famous IF writer Emily Short [140]:

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” is just 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 by Eve [58] 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 by Mateas and Stern [99], 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 [98]. 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.

4.3.2 Research

Again, where knowledge is considered, the concern seems to be exclusively domain knowledge, or knowledge about facts in the game world, as in work by Ponsen

\(^2\)Note, 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.”
4.3. Related work

et al. [118] and Spronck et al. [143]. A more general approach of using agent programming languages to script NPCs (e.g., by Leite and Soares [91]) 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 by Cutumisu et al. [45] 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.

An interesting approach, described by da Silva and Vasconcelos [141], 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.

In a similar way, the work by Magnusson and Doherty [95] is a highly promising existing platform for implementing epistemic mechanisms. It consists in a virtual game world that is based on a generic theorem prover for a logic of time and action. Incorporating rules for epistemic reasoning into this framework seems like a very natural extension.

Some literature on higher-order reasoning in multi-agent systems that does not focus on computer games is also very relevant. Dragoni et al. [53] study the specific problem of agent communication, 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 by Yin et al. [160] 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., the recent survey by Bordini et al. [27]). While these often do provide for explicit knowledge operators, they are never higher-order.3 So these platforms allow for the use of conditions of the form “If the agent knows that the cup is on the table, then . . .”, but not “If the agent

3However, see [34] for a recent step towards higher-order modeling in an agent language.
knows that the other agent knows that . . . ”.

One AI framework that does allow, and is explicitly designed for, handling higher-order beliefs is the formalism of Interactive Partially Observable Markov Decision Processes (I-POMDPs) by Gmytrasiewicz and Doshi [68]. It implements hierarchies of nested probabilistic beliefs of agents about the world and about each other, including Bayesian updating of such belief hierarchies. Since we here focus on symbolic frameworks, we do not pursue this approach further.

4.4 Potential applications

In the following we illustrate how our approach could enhance potential or actually existing computer games.

Knowledge games, as described in Section 4.1.1, 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 do 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.

4.4.1 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™ by Eidos Interactive [56]. 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.

\footnote{The framework is similar to (a finite version of) type spaces known from game theory.}
4.4. Potential applications

if B(g, t_present):
  if B(g, not B(t, g_present)):
    g.ambush(t)
  elif B(g, not B(t, B(g, t_present))):
    g.attack_or_alarm_inconspicuously()
  else:
    if not alarm_active:
      g.alarm_quickly()
    else:
      g.attack(t)

Listing 4.1: Pseudo-code for a guard in Thief: The Dark Project. B(x,y) stands for “x believes y”, g stands for guard and t for thief.

While there are more obvious and ubiquitous applications for higher-order reasoning generally in RPGs and life simulation games, in this somewhat unusual social interaction setting it is certainly crucial to keep track of who knows (or believes) what about whose presence, both for the player and for plausible and challenging NPCs. Essentially, the thief player exploits the, possibly false, beliefs of the guard regarding the player’s presence. Due to the simplicity of the guard’s control program in the commercial game, the guard’s beliefs are in practice perceived as either believing that the thief is, or may be, near (having seen him or become suspicious in some other way), or not.

We conjecture that the entertainment value of a typical Thief scenario would be enhanced by a guard that acts not only depending on his own beliefs about the facts in the world, but also depending on what he believes the thief believes, including what he believes the thief believes he believes.

Imagine an NPC that embodies a guard, hostile to the thief. Among his behavior rules could be the following: If the guard believes that the thief is present, but the guard also believes that the thief does not believe that the guard is present, then the guard tries to ambush the thief. Under other conditions, the guard may instead act inconspicuously and attack by surprise, or, if all else fails, attack the thief openly. The pseudo-code in Listing 4.1 specifies several such rules, depending on higher-order beliefs. How such beliefs can come about may vary: for example, 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 epistemic 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 turn out to be very efficient.

In Section 4.5 we return to this scenario in some more detail and describe a
simple example implementation of a corresponding knowledge module.

4.4.2 Adding credence to Assassin’s Creed

Assassin’s Creed™ by Ubisoft [146] presents another crisp case for higher-order
knowledge or beliefs. Altair (the player 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.

For higher-order reasoning, suppose that a guard partitions the crowd into
bystanders and suspects by observing who is armed. The guard observes speech
and body language of each bystander (technically voice-over callouts and animation
states) to infer beliefs of the bystander. As soon as the guard observes a bystander
who is responding to a murderer, the guard assumes the bystander’s target is the
killer. Observing the body language of the suspect (possibly Altair), the guard
may be able to infer whether the suspect believes that the guard is onto him.

Listing 4.2 shows pseudo-code of this higher-order analysis. Admittedly, such
a proposal would require further design, but the example illustrates reasoning yet
somewhat gullible guards, which a crafty assassin may delight in deceiving.

```plaintext
if B(g, murder_happened):
    suspects = { x | B(g, is_armed(x)) }
    bystanders = { x | B(g, not is_armed(x)) }
    for b in bystanders:
        if B(g, B(b, murder_happened)):
            for s in suspects:
                if B(g, B(b, is_killer(s))):
                    if B(g, B(s, B(g, is_killer(s)))):
                        g.shout(is_killer(s))
                        g.attack(s)
                    else:
                        g.whisper(is_killer(s))
                        g.ambush(s)
                    break
Listing 4.2: Pseudo-code for Assassin’s Creed. Again, B(x,y) stands for “x
believes y” and g stands for guard.
```
4.5 Implementation study for Thief

In this section, we substantiate our discussion from Section 4.4.1 by describing an actual implementation of the knowledge module used in the pseudo-code in Listing 4.1.

4.5.1 Knowledge module

We consider various ways in which relevant beliefs can come about: by causing or hearing noise, seeing the other one from behind, or facing each other.

We want to emphasize again that the NPC scripter need not worry about the details, he simply uses the familiar concept of belief in order to express the rules on a high level. It is the job of the knowledge module to take care of maintaining and updating the agents’ knowledge, taking into account whichever events occur in the virtual world. As an agent’s control script is executed, the knowledge module is queried and determines the truth value of any given formula.

We assume that the scene starts with thief and guard present and no events having occurred, and that agents cannot enter or leave. Put differently, the guard will not leave the scenario, and if the player does, the scenario ends. Note that this assumption is not inherent in our approach and serves mostly to avoid unnecessary complications. In the context of a computer game it is not too unnatural: game worlds often are simulated per scene, and a scene starts and ends whenever the player enters or leaves.

In our scenario, we consider the following events:

- \( b_t, b_g \): The thief, respectively the guard, sees the other one from behind.
- \( n_t, n_g \): The thief, respectively the guard, makes some noise. We limit this to “relevant” events in the sense that, e.g., \( n_t \) can only occur if the thief is present.
- \( f \): Thief and guard see each other face to face.

The epistemic effects of these events intuitively are as follows:

- \( b_t \): The thief believes that the guard is present; the guard never assumes this event to occur, so this event does not affect the guard’s beliefs.
- \( n_t \): The guard believes a thief is present; the thief believes that, if a guard is present, that guard believes that the thief is present.
- \( f \): Thief and guard commonly believe that both are present.

\[ ^5 \text{In our case beliefs rather than knowledge, but we stick with the name “knowledge module”}. \]
The effects of $b_g$ and $n_g$ are analogous, and all these effects are commonly believed. For now, we assume that events are not forgotten. Note that a consequence of all this is that after both $n_g$ and $n_t$ have occurred, guard and thief commonly believe that both are present.

To model this situation formally, we use a modal logic model with history-based semantics [116, 60]. We introduce two propositions, $p_t$ and $p_g$, with the reading that the thief, respectively the guard, is present. A valuation is a function that assigns either true or false to each of these propositions, and can be denoted as a set $V$ containing those propositions that are to be assigned true.

The set of events, as above, is $E = \{b_t, b_g, n_t, n_g, f\}$. A history $H$ is a sequence of events. For a history $H$, by $Ag(H)$ we denote the set of agents involved in the events in $H$, i.e., $Ag(H) \subseteq \{t, g\}$, where naturally both $t$ and $g$ are involved in any of $\{b_t, b_g, f\}$, and only $t$ (respectively $g$) is involved in $n_t$ (respectively $n_g$). We also write $H - e$ for any $e \in E$ to denote the history obtained by removing all occurrences of $e$ from $H$. The empty history is denoted by $\epsilon$.

A pair $(V, H)$ is a state (or possible world) if and only if $Ag(H) \subseteq V$. So the described initial situation is represented by the state $(\{p_t, p_g\}, \epsilon)$.

We now define accessibility relations $\rightarrow_t$ and $\rightarrow_g$ between states in our history-based model. Intuitively, $(V, H) \rightarrow_t (V', H')$ means that in state $(V, H)$, $t$ considers it possible that the state is $(V', H')$. According to the intuitions described above, we define these relations as follows:

\[(V, H) \rightarrow_t (V', H') \iff \begin{cases} t \not\in V' \text{ and } H' - n_g = \epsilon & \text{if } t \not\in V \\ t \in V' \text{ and } H' = H - b_g & \text{if } t \in V. \end{cases} \]

Note that in the first of these two cases, $n_g$ is the only event which can occur in $H'$, so the condition boils down to saying that $H'$ may be any sequence of $n_g$. The second case captures the intuition that $t$ does not assume the possibility that he is being seen from behind. The relation $\rightarrow_g$ is defined analogously.

We consider the doxastic language consisting of formulas of the following form:

\[\varphi ::= p_t \mid p_g \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \lor \varphi \mid B_t \varphi \mid B_g \varphi.\]

The semantics is standard relational modal semantics, see [24]. Note that here we do not impose further restrictions on the doxastic language, since our scenario is simple enough for the knowledge module to evaluate all formulas efficiently.

If we merge all doxastically equivalent states, that is, all states that make the same formulas true, our model can be depicted as in Figure 4.1.\footnote{This is the so-called bisimulation contraction of our original model. Note that in our case, if $(V, H)$ and $(V', H')$ are doxastically equivalent, so are $(V, He)$ and $(V', H'\epsilon)$ for any event $\epsilon$.}

Note that, in contrast to our starting point from history-based semantics, this representation is static in the sense that it does not grow (or shrink) over time, as the world evolves and events occur. It is straightforwardly represented in any
4.5. Implementation study for Thief

Figure 4.1: Representation of our doxastic model. The numbered nodes represent states and are annotated with one of the equivalent possibilities of what events have (optionally) taken place. Boxed states have $V = \{p_t, p_g\}$, a missing left corner means $p_g \notin V$, and a missing right corner means $p_t \notin V$. State 0 represents the initial situation. Solid arrows with black labels show event transitions, dashed arrows and edges (the latter corresponding to bidirectional arrows) with gray labels show accessibility relations. The following are omitted for clarity: reflexive transitions for each optional event at each state; $f$-transitions from each boxed state to 8; and reflexive accessibilities, except for $g$ in states 2 and 6 and for $t$ in states 1 and 7.

programming language (our implementation uses Python), and the knowledge module simply needs to keep track of which one is the current state.

Finally, note that while agents can have false beliefs, there is no need for elaborate belief revision mechanisms in our simple scenario, since there is no way for the agents to notice their false beliefs.

4.5.2 Expected impact on gameplay

In the original game, a common tactic for the thief is to shoot an arrow against a wall, in order to cause a noise which will attract the guard (who has a behavior rule along the lines of “if I hear a noise, I go there and look around”). It is questionable how innocent such a noise is, and an actually reasoning guard would
probably rather conclude that it is high time to ring the alarm instead of sniffing
around in the bushes. While this was likely a conscious design decision to give
the player an easy and obvious way to outwit that guard, we think that our
belief-based approach provides a more flexible solution.

Instead of rigidly connecting events to resulting behaviors, beliefs in our
approach act as an abstraction layer. The behaviors are defined depending on the
beliefs, and these beliefs are modeled independently.

This approach removes from the scripter the burden of having to decide
exactly which events cause what, and facilitates adjustments and more complex
dependencies. For example, in our scenario a more clearly innocent noise such
as breaking a twig will induce the guard to believe that there is a thief (who
accidentally caused that noise), which will then trigger the behavior rule that says
to ambush him.

We expect that our modest belief engine for the scenario we have described
courages the original game’s tactics of misleading a guard. It also encourages
new tactics: The thief may play stupid and let himself be seen from behind by
the guard, who then again thinks he can do something sneakier than ringing the
alarm.

We also expect that our scenario encourages trepidation. A guard who has
actually noticed the thief without being noticed himself will act inconspicuously
while preparing a counter-attack. The blackjack is the weapon for subduing, which
has a short reach. A guard that pretends to not have noticed, will swing around
and stab with his sword first before the player’s blackjack is in range.

Overall, we expect that the player enjoyment and interest will increase.
Whether or not these expectations prove true remains to be seen in an experimental
evaluation, which we are planning to conduct in the future.

4.6 Conclusions

We have described a modular approach to adding (higher-order) knowledge opera-
tors 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. Finally, we have
presented a simple implementation for one of the described example scenarios.

4.6.1 Explicit knowledge programming

The knowledge module we presented implements a centralized version of the
explicit knowledge programming approach we proposed in [155] and used in
Chapter 3. The fact that it is centralized makes it simpler from the outset than a truly distributed setting: In a distributed setting, an agent would have to deduce locally whether a given formula follows from his observations, that is, whether it holds in all models which are compatible with the local observation and might thus reflect the actual situation; in our centralized setting, the game engine has the model of the actual situation available and only needs to check whether it satisfies the given formula. This together with the simplicity of our scenario removes the necessity of finding restrictions of the doxastic language or other optimizations: about any conceivable model and implementation would be trivially tractable.

However, in all its simplicity our scenario still shows the advantages of the modular approach introducing knowledge or beliefs as an abstraction layer and separating the epistemic model from the agent’s control program. The knowledge module can be refined independently to cope with new events or extensions, such as probabilistic beliefs, with no need to change the behavior scripts. One extension that is easy to incorporate into our knowledge module is decaying events, or unstable states: If the guard makes some noise like whistling, he might have forgotten about that when he 10 minutes later sees the thief from behind, and therefore not conclude that the thief believes the guard is present. This could be modeled by a spontaneous transition from state 5 to 2 after some time.

While one could implement some ad-hoc tracking of events in the agent control program, this is error-prone and difficult to maintain, as already in our small test scenario the exact dependencies on events are getting confusing: What does “if $n_t$ or $b_g$ have occurred, but neither $n_g$ nor $f$ have, then…” mean? This quickly gets even more confusing in larger scenarios—our approach certainly scales better conceptually.

A critical issue with respect to scaling, however, is the possible explosion of the number of states in the model. Mechanisms to generate the model only locally as needed, and strategies for expanding, shrinking, or swapping out unused parts of the model may be necessary. Techniques from the area of model checking [41] will be useful for compactly representing and efficiently processing models.

4.6.2 Alternatives and extensions

The advantages of the representation we have used here is that it is straightforwardly stored using very simple data structures, and that it is static and pre-computed, so there is no expense at run-time for maintaining the model.

A very interesting alternative is building on Dynamic Epistemic Logic (DEL, see [52] for a recent textbook). Instead of one model representing all possible ways in which the world may evolve, one would start with a model representing only the initial situation. Events are represented as so-called update models, which are applied to the current model as the events occur, transforming it to reflect the resulting situation. See Chapter 7 for some discussion on this approach.
An interesting suggestion that would also simplify reasoning is to consider so-called interaction axioms between the belief modalities of the players. For example, Lomuscio and Ryan [93] show that (2WD) is valid on two-player hypercubes, i.e., models that are based on the Cartesian product of an interpreted system’s state space:

\[ \Diamond_1 \Box_2 p \implies \Box_2 \Diamond_1 p. \] (2WD)

This can be read as “if agent 1 considers it possible that agent 2 knows \( p \), then agent 2 knows that agent 1 considers \( p \) possible.”. In a different context, van Ditmarsch and Labuschagne [51] consider interaction axioms that characterize different theories of mind, including “autistic” and “deranged”. It is an interesting question whether interaction axioms might exist that capture agents that are fun to interact with.

Another point worth noting is that we do not distinguish between knowledge and true belief. There are many philosophical discussions concerning differences between these two notions, and formally they are generally taken to be different as well. Beliefs should really be revisable, for example, which is something we have not considered here. A possible future direction of research would be to use models and languages that take both belief and knowledge into account, for example along the lines of Shoham and Leyton-Brown [139, Chapter 13]. Some actions would then generate knowledge, while some would (only) generate belief. This corresponds to the distinction van Benthem [19] draws between “hard” and “soft” information. For example, if you see something then you might be said to know it, but if somebody you do not entirely trust tells you something then you might only believe it.

Finally, it may be desirable to get rid of manually designed behavior rules altogether, and let the agents act purely based on their beliefs and some abstractly defined goals. A very long-term vision is then to have an artificial guard that by himself adopts behaviors similar to the ones we described, or to the tactics we conjectured for the human thief in Section 4.5.2.

### 4.6.3 Cognitive considerations

Rather than working out “bottom-up” what class of epistemic statements are needed for a certain setting or behavior, one may consider a more generic “top-down” approach. Given that we want to simulate human social interaction, the general question is: What class of formulas can humans be said to evaluate in everyday life, consciously or not?

Some results from experimental game theory about levels of strategic thinking, e.g., by Camerer [36], can be interpreted as being relevant to this question. However, these experiments do not focus on everyday social interaction. For example, the Beauty Contest game mentioned there might invoke conscious and explicit reasoning about the other agents, while we believe that in real-life social
interaction, through years of experience, the requisite higher-order reasoning processes may also occur on a more intuitive and reflexive level.

Furthermore, the experimental designs are in general not specifically concerned with knowledge, so that at best the results can give us hints about the nesting depth of knowledge operators. Clearly other criteria might define the class of knowledge formulas that are of relevance in social interactions.

From the real-life example in the beginning of Section 4.4 it is clear that formulas like

\[
\begin{align*}
K_a &\neg K_b p \\
K_a &K_b p \\
\neg (K_a K_b p &\lor K_a \neg K_b p)
\end{align*}
\]

matter. However, that does not necessarily hold for all formulas with knowledge nesting depth 2. Also, human reasoning capabilities may not be monotonic with respect to this complexity measure. For example, for a special concept like common knowledge, which in theory involves infinite depth, we may want to assume that humans are able to cope with it, while this does not hold for “intermediate” depths like 10000.

The main issue thus remains: How can we define this class of relevant formulas?

Results from experiments by Verbrugge and Mol [147] suggest that subjects use first-order theory of mind (beliefs about others’ beliefs), but not “all kinds of reasoning possible and useful in this context”. This supports the claim that the depth of knowledge operators is not the only relevant criterion. It is further reported that some subjects use second-order theory of mind, which corresponds to third-order epistemic formulas.

While it is important to look at such questions in actual experiments, thought experiments or observations from real life can also make their contribution. Work by Parikh [112] is an excellent source for enlightening examples examining what kinds of reasoning processes are going on in real life. They can be convincing enough to remove the need for abstract and reproducible lab conditions, for the benefit of being set in more natural environments, where human reasoning capabilities possibly profit from experience and training in specific social situations.

4.6.4 Final words

As mentioned earlier, in computer games the goal of incorporating epistemic reasoning may not necessarily be verisimilitude, because their ultimate objective is enjoyment and entertainment. For analogy, car racing games such as Burnout Paradise™ by Electronic Arts Inc. [57] 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 developed for simulating physical bodies into frameworks for animating virtual toys. We hope
that an epistemics engine comprised of algorithms originally developed to simulate beliefs about the beliefs of other agents may evolve into a toolkit for cleverly (mis)informing the minds of virtual playmates.