Towards Articulate Game Engines

Daniel Jeffery Dobson and Kenneth D. Forbus
Northwestern University
Department of Computer Science
1890 Maple Ave.
Evanston, IL 50201
{wolff, forbus}@cs.nwu.edu

Abstract

Conventional simulators lack explanatory power. Traditional simulators provide the dynamic behavior needed for a gaming world, but do not provide conceptual understandings of the world to players or game AIs. By creating game engines that have a qualitative understanding of the system they describe, authors can more easily build richer and smarter game agents. This paper outlines why game engines should include more conceptual understandings, suggests some relevant AI technologies, and speculates on how this might be accomplished.

Inroduction

"We've analyzed their attack sir, and there is a danger."

In "Star Wars,"[1] the Grand Moff Tarkin had logistical support when battling the rebels. His aides found a critical weakness in their own setup and had generated an escape plan. Unfortunately, players never experience such rich interactions with their computer advisors or opponents. Computer advisors tend to be simplistic and sensitive only to simple global features of the situation. They tend to be predictable and often not good enough, relying on hidden information and massive cheating (e..g, accelerated development and extra resources) to give their human players a good game. If the Grand Moff Tarkin were a typical computer player, he would continue to construct Death Star after Death Star, watching the human player knock one off after another without ever revising the plans to remove those darn trenches. (Even George Lucas only tried to make it work twice.)

We describe how some ideas developed in artificial intelligence, notably qualitative reasoning and self-explanatory simulations, might help improve this situation. By creating game engines that have a conceptual understanding of the gaming world, games could include better advisors and opponents, as well as automating more of the simulation construction process. We start by examining the need for a built-in conceptual understanding of the game world, then describe how qualitative representations could provide that understanding. Then we describe a particular technology, *self-explanatory simulators*, illustrating their potential for creating game engines that contain such built-in understanding with very small runtime penalties. Finally, we list some of the difficulties that must be overcome in building articulate game engines.

Why Tarkin is doomed in today's simulations

Consider first the Sim games from Maxis. In games like SimCity[2] or SimEarth[3] , which are combinations of cellular automata (CA) and system dynamics (plus a few other ingredients) are combined to provide a rich simulated world. The CA component provides a sense of space and locality, as well as discrete changes based on both local properties and global properties from the system dynamics aspects of the model. The differential equations of the system dynamics component provide global interactions, integrating the properties of cells into aggregate parameters, such as overall crime levels in a simulated city. These causal relationships then help drive the interaction of the CA component. The very richness of this interaction which makes these simulations so appealing can also lead to dreadful confusion on the part of players: *Why did my attempt to terraform Venus suddenly fail?*

Explaining the simulations

Game designers often explicate parts of their models in the interface, documentation, or software advisors, but these partial models are usually unsatisfying (to both player and designer). Moreover, the advice given is not based on a conceptual understanding of the player's current situation. Typically their advice is more like a warning light on a dashboard than a detailed analysis of the player's position. "You don't have enough fire departments" is much different than "Your fire departments aren't strategically placed."

Of course, keeping the appropriate amount of mystery in a game is important -- experimenting and playing detective is often a critical source of enjoyment -- but the game designer should have a broad range of options as to how much information to provide and in what form, rather than being minimalist because that is the only feasible option.

Computer players

Interactions with computer players, as opponents, advisors, or just passers-by, help draw a player into the game world. An understanding of the events of the simulated world are also important for creating richer computer characters. In games like Civilization II [4] and Age of Empires [5], computer players are governed by simple scripts, build-lists, and carefully-tuned mechanisms. The conceptual description of the gaming world, which governs the designer's crafting and tuning of the world and its bots, is nowhere explicit in the computational engine itself. If players find fundamental weaknesses (say, trenches in Death Stars), there is no conceptual knowledge to fall back on.

Yet it is this conceptual level of knowledge, tied appropriately to the detailed mechanics of the world (e.g., qualitative human perception, in the case of the real world), that is typically used as a guide to categorizing events and to act accordingly. Computer players that understood the causal relationships that govern the game world could strategize, plan, and advise better. Understanding the game world will also help interpret the human player's actions, enabling responses that help make the human player feel more visible, and hence more engaged, in the game world.

A technology for articulate game engines

Designing and implementing the physics (or economics) of a game engine requires selecting appropriate modeling assumptions, trading off constraints of physical verisimilitude, playability, CPU and memory budgets, and development time. In simulation construction, qualitative knowledge provides an organizing framework for the physical objects, relationships, processes, and their mathematical models. After deciding what phenomena are relevant, the engine designer needs to figure out the appropriate approximations and mathematical models, and then implement them in efficient code. Unfortunately, the conceptual understanding of the simulated world remains in the simulation designer's head; it is *unarticulated.*

If we can embed into the game engine itself the kind of conceptual understanding that players (human and computer) need to operate in it, we will have an important tool for creating the kinds of explanation tools, post-mortem debriefings, and bots that we would like. We propose that *qualitative knowledge,* expressed for example in the form of self-explanatory simulators, can provide a basis for *articulate game engines* that would support these new developments.

Qualitative knowledge is the kind of conceptual, everyday knowledge that we use to categorize, explain, and reason about the physical events and processes that happen around us. Consider a ball thrown up in the air. Viewed as a trajectory of real-valued coordinates, it would take an infinite amount of information to describe. Viewed in floating point, the number of states is smaller but still huge, proportional to the time step used in the simulation. But for human conceptual descriptions, the most natural description is something like "goes up, comes down." Qualitative representations impose symbolic descriptions on continuous parameters, making them amenable to conceptual reasoning.

What kind of conceptual reasoning do you need to do? If what is thrown is a piece of Waterford crystal rather than just a ball, one might decide to maneuver to get underneath it before it falls. A feeling for the quantitative is necessary to understand where you need to physically be and how soon you need to get there, but it is qualitative knowledge that tells you that you have this dilemma in the first place. Moreover, qualitative knowledge provides concise summaries that make it easier to spot complex behavior patterns. It is easier to extract a pattern like "decaying oscillation" from a sequence of `UP/PAUSE/DOWN/COLLIDE` events where the energy at each subsequent `PAUSE` is lower than it is from the raw numerical data. (In fact, in engineering qualitative reasoning techniques are being used to create useful, human-like descriptions of very complex behaviors, including chaotic systems.)

Self-Explanatory Simulators

Our group has developed a technology that can help change this. *Self-explanatory simulators* combine the ability to do efficient numerical simulation with the conceptual clarity of qualitative representations. The basic idea is to make the process of simulation authoring more automatic---automating generation of both the code for grinding the numbers and articulating the conceptual knowledge. That is, a self-explanatory simulator compiler (here, SIMGEN Mk3) is given as input a conceptual understanding of the domain to be simulated and a description of the structure of the particular system within that domain to be simulated. SIMGEN figures out what phenomena are relevant and what modeling assumptions are appropriate, selects the appropriate mathematical models, and writes the simulation code. (It can take advice from a human partner, depending on the application.) SIMGEN also creates, as part of the simulator, a compact version of the conceptual understanding it used to create the rest of the simulator. This information is used to provide explanations about the physical entities and relationships in the simulation. This explanation system enables self-explanatory simulators to generate a conceptual description of simulated behavior as an inexpensive byproduct of the simulation itself.

The runtime costs of self-explanatory simulators are slightly higher than traditional simulators: The explanation system takes up a small, fixed amount of memory, and the conceptual explanation of any particular simulated behavior grows according to the qualitative complexity of that behavior. The qualitative complexity grows at a rate that is measured in aggregate timesteps, rather than whipping around rapidly at every timestep. However, the

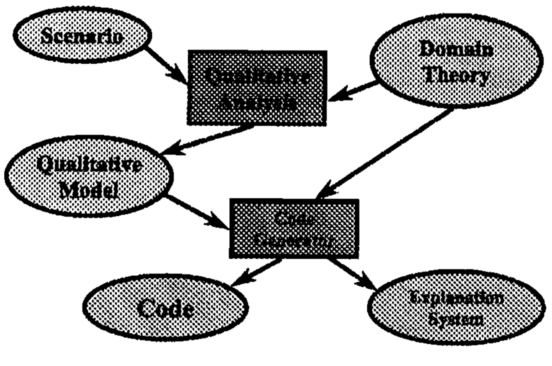

Figure1. SIMGEN's inputs and outputs. A domain theory is combined with scenario information to form a qualitative model. This is compiled to both running numerical simulation code and a compact explanation system that contains information about relationships in the system.

complex reasoning occurs at compile time, so the time cost is basically that of a traditional numerical simulator.

SIMGEN's simulations can be run on small platforms -- we have successfully run them on HP200's (an 8mhz DOS palmtop), for instance, and as Java applets. The compiler produces Common Lisp, C++, or Java runtimes, which can be used with various runtime shells (with more on the way). Examples of simulators built with SIMGEN Mk3 for educational purposes include a spring-block oscillator (with static and dynamic friction, as well as a MEMS spring), an evaporation laboratory (so experiments that would take hours occur in seconds), and a model of the atmosphere to explore the effects of greenhouse gases. We are creating a suite of simulations as part of a Principles of Operations manual for a "virtual Deep Space One." It is part of a joint project with NASA Ames and University of West Florida. We are also creating self-explanatory simulators for use with two middle-school science curricula for the Chicago Public School system.

Currently, the arena in which self-explanatory simulators excel are systems of ordinary differential equations without simultaneities (e.g., the same sorts of systems that system dynamics works for), although the structure of the system of equations can change dynamically during simulation. The largest cost in developing a self-explanatory simulator is the creation of the *domain theory*, the conceptual understanding that fuels the simulator. Domain theories cover a broad range of physical situations and phenomena, so that while creating a domain theory can be a large up-front cost, it can be amortized across many simulators. Moreover, the increased representational capacity has a number of advantages. For example, we have built a development environment that enables developers to add new virtual entities to existing simulations, adjust their explanations, and automatically tune the simulation for desired behaviors.

One might consider grafting this technology onto existing simulations like Civilization or SimCity. Although this is possible, it's not actually a particularly efficient use of the technology. It would be a long and tedious task to back-fit a SIMGEN domain theory to the hand-hacked behavior of, say, SimCity without having SimCity designers on hand to explicate their qualitative design decisions. It would be possible, too, to maintain SimCity's existing numerical simulation code and add hooks to signal numerical changes to a SIMGEN model reflecting a qualitative design, but a really horrible task fraught with trial-and-error. SIMGEN's strength is using qualitative knowledge to design the backbone of the simulation and compile it into running code.

How articulate game engines might help

Concepts from self-explanatory simulators have an obvious application to helping players understand the gaming world. Consider for example SimEarth. The back of the manual provides an interesting attempt to reconstruct the causal structure of the simulation, based on design notes and experimentation. But if players were able to directly trace back through the causal structure of events, to find out the *why* of them, they would be able to tie the appropriate parts of the simulator's causal accounts to the specifics of their actions. Articulate models also provide the basis for indexing into interesting stories and other informational resources: *"Yes, you've just scoured the Earth of all higher life forms. Kind of what may have happened back when..."* Detecting and providing relevant advice to dynamical patterns, e.g., mounting instability or a population death spiral, would also be greatly simplified.

The physics of the gaming world is only part of the story, of course. Qualitative models can also be used for economic and resource models, thus providing a stronger basis for reasoning about strategic decisions. The qualitative descriptions of events and patterns of physical and economic behavior could be used to drive reactive strategies and signal opportunities for reflective planning by bots. In most games, a few simple flags hand-woven into the engine are all that are available to drive advisories and bots. In Civilization, for instance, the advisors provide amusing summaries of your global state, but are more like dashboard warning lights than an advisor who can point out where you went wrong in the last war and how to avoid doing it again.

If most of the qualitative descriptions of the world are computed automatically as a consequence of the normal process of simulation, these descriptions can be shared by computer players, thus freeing CPU and memory budget for more strategic thinking. Explicit symbolic descriptions of the goals, actions, strategies, and tactics of the gaming world can be used to extract the maximum benefit from the rich information that would be available in articulate game engines. Imagine a strategy war game where, based on its understanding of your economy, your enemy kept targeting your weakest supply links. Or a competitor in an economic game that figured out it could corner the market on energy, or stop you from cornering a market, or accuse you of illegal acts based on its understanding of your behavior and the laws of the game.

Although we think that self-explanatory simulators could be an important technology for game engines, the potential of qualitative reasoning for building better bots goes beyond them. Consider for example the potential use of *qualitative spatial descriptions.* Qualitative spatial reasoning carves space and shape up into regions that are uniform with respect to some relevant criteria. These conceptually meaningful divisions provide an alternate perspective to grid-based or numerical spatial models. In games like Age of Empires, for instance, a bot could make more global deductions, such as *"I'm in imminent danger of collapse because my enemy controls all the stone in the world,"* or even *"I should try to build a bridge to the stone on the island."* Bots could actually understand concepts like "front", "line of attack", and properties of terrain, concepts which often bedevil designers of wargame AIs.

Technical barriers to articulate game engines

There are several advances that must be made in self-explanatory simulators to make them a practical technology for game builders:

- *Working efficiently with grid and cellular automata models.* SIMGEN Mk3 does not provide for array-based models, which will require some careful engineering but not deep conceptual difficulties.
- *Creating libraries of domain theories and environments for developing new domain theories.* Domain theories describe the laws of some class of phenomena, including both qualitative models and mathematical models. The qualitative models provide high-level descriptions of phenomena that help determine when specific mathematical models are relevant. A large library of off-the-shelf domain theories could drastically cut game development time, especially if there were tools that enabled these domain theories to be customized easily.
- *Creating libraries of high-level behavioral and causal patterns.* Recognizing emergent behavior patterns in the simulation, such as a death spiral in a population or hyperinflation in an economy, requires additional reasoning beyond what the explanation systems of self-explanatory simulators provide (although their representations provide a good starting point). Creating a standard library of these symbolic descriptions, and procedures to recognize them based on qualitative and numerical descriptions of dynamical behavior, would simplify bot development.

There are other issues that must be addressed before the full potential of articulate game engines can be realized:

- *Creating libraries of representations for goals, actions, strategies, and tactics.* Developing symbolic representations for these concepts, tied to the conceptual knowledge of behavior and dynamics, that can be used across multiple games will amortize the cost of bot development. Examples of this might be the idea of forking an opponent as in tic-tac-toe or the concept of a *front* above.
- *Dynamic reconfiguration of simulators.* Creating simulated worlds where players can create new artifacts that can be understood within that world is a difficult challenge. The standard approach of viewing such situations as distributed simulations provides only part of the answer: Unless there is a shared conceptual understanding, generativity will have to be strictly limited or the bots of the world will be unable to cope. It is possible that this shared conceptual understanding can actually make the distributed simulation problem easier (i.e., two simulations can describe what they affect and can realize that they can operate independently of each other, or that they only depend on each other in certain ways).

Conclusion

We believe that qualitative reasoning has much to offer game designers and implementers. Articulate game engines, by generating a qualitative, conceptual understanding of their behavior in parallel with purely numerical descriptions, could support the creation of smarter, savvier computer-controller characters, whether they be advisors, allies, or opponents. Progress in other areas of artificial intelligence (e.g., efforts to formalize many common-sense concepts of space, time, causality, and domains of interest to the military, policy-makers, and gamers), combined with Moore's law, provides a tantalizing opportunity for creating far richer gaming experiences. And, finally, the Grand Moff Tarkin might be smart enough to escape the Death Star and really strike back.

References:

[1] *Star Wars*, Twentieth Century Fox Corporation, 1977.
[2] *SimCity*, Maxis, 1989.
[3] *SimEarth*, Maxis, 1996.
[4] *Civilization II*, MicroProse Inc., 1996.
[5] *Ages of Empires*, Microsoft Corporation, 1997.