

Game-Mechanics Reasoning for Automated Design Support

Mark J. Nelson

School of Interactive Computing*
Georgia Institute of Technology
mnelson@cc.gatech.edu

Videogame design fundamentally involves engineering interactive rule systems: a game designer combines a set of *game mechanics* such that, when they interact with each other and with the player's actions, they produce the desired gameplay. Game designers typically prototype these rule systems to understand how they operate. Prototypes range from paper versions, in which a stripped-down form of the game's rule system is simulated manually, to playable, implemented versions, which can be played by the designer and others to get feedback on gameplay ideas or discover problems. Although these types of prototypes will remain part of the design process, my thesis proposes that a number of the design questions such prototypes try to answer can be answered automatically.

Prototypes aim to answer both subjective and objective questions. The ultimate design questions are mainly subjective: is the game interesting, fun, challenging, balanced, etc.? However, much prototyping gets at these issues indirectly by asking objective questions that help the designer understand how their rule system operates. For example, is there a way to win with a combination of items? Can the player ever get to a bit of story without having gotten its set-up?

The objective kinds of questions are amenable to automated reasoning, since they have answers that depend solely on the game's formal rule system. By answering them automatically, we can speed up the design loop by allowing designers to quickly understand how their rule system is operating, getting much more factual understanding of the system that they can use in their subjective design thinking. This increases what design researchers describe as the *backtalk* of a design situation [Schön, 1983]. Such a system has some similarities to the goals of especially early CAD systems (before a greater focus on visual modeling in later systems), which aimed at a design loop where automated reasoning could give engineers quick feedback about the physical properties of their designs [Reintjes, 1991, pp. 95–96].

The proposed system involves four main research questions. First, videogame mechanics must be formalized in a manner that supports automated reasoning. Second, we must understand what kinds of automated reasoning videogame designers would want from such a system. Third, some subset of those desiderata need to be implemented in a tractable automated-reasoning system. Finally, the backend reasoning

must be integrated with an interface and way of specifying mechanics that allows designers to interact with it usefully.

Current progress

To position game-mechanics reasoning within game design, I proposed a four-knowledge-area domain analysis of games [Nelson and Mateas, 2007]. In this view, games consist of *abstract mechanics*, which are the high-level rules of the game (time limits, qualitative spatial relationships, goals, etc.); *concrete representations* and their associated mechanics, which are the low-level rules and audiovisual representations of the game (color changes; position in 3d space; collision detection; etc.); *thematic mappings*, which are the game's real-world references (war, politics, bars, etc.); and *input mappings*, which specify how the player's inputs interact with the abstract and concrete mechanics. Of these, automated reasoning about game mechanics can largely ignore thematic mappings (though they are important for game designers), but must consider the other three knowledge areas and their relationships.

Game mechanics then need to be formalized in a way that permits automated reasoning. I've adopted the event calculus, a logical representation for reasoning about states, events, and time, which contains a number of design features and idioms aimed at commonsense reasoning [Mueller, 2006]. Game state maps on to event-calculus *fluents* (time-varying predicates); both player input and events triggered by game state map onto *events*, and the game mechanics themselves are formulas specifying the relationships between fluents and events [Nelson and Mateas, 2008].

A game represented in the event calculus can be both played and queried. Playing is done via temporal projection, with input events generating event-calculus assertions, and certain fluents mapped to graphical representations, *e.g.* drawing $At(sprite, x, y)$ on a 2d canvas. Queries can take various forms, including planning to find whether particular sequences of events are possible, retrieval of game entities with particular properties, and so on.

To get an idea of what designers would actually want to do with the representation, I conducted a preliminary study with three small teams of independent game designers [Nelson and Mateas, 2009]. Since the idea at this stage was to collect functional requirements rather than prototype an interface, I served as the system's interface, mapping the designers'

*Much of this research was done while visiting the Expressive Intelligence Studio at the University of California, Santa Cruz.

work-in-progress prototypes to formal representations, translating their English queries to logic queries (where possible), and translating query results to interpretable form. Designers generally found it easiest to interpret the system's possibilities in terms of enhanced simulation (rather than distinctions between *e.g.* planning and simulation): they frequently both wanted a distribution of outcomes under some assumption (*e.g.* of a particular player policy), and to be able to pick out specific traces where an event took place. This requires some approach to handling nondeterminism. In addition, a number of questions could be mapped onto logical abduction or induction (depending on the approach), *e.g.* coming up with a simple player model that can consistently beat a game. There were also threshold type questions, such as how large or small a particular value could be made before dynamics changed.

For designers to be able to specify mechanics and reasoning queries, and for the queries to be automatically answered by one of the many event-calculus reasoning backends, I designed a translation layer that encapsulates common event-calculus idioms as language-level constructs, and compiles the resulting reasoning problems to answer-set programming, Prolog, SAT, and possibly other reasoners as appropriate. The main language-design goal was to make it so common operations can be handled with less tedium and likelihood of errors. Examples of common things that are tedious or error-prone, but for which correct representations have been worked out, are representing integer-valued counters via fluents, having multiple events simultaneously change the same counter, and having events cause indirect effects [Miller and Shanahan, 1999; Shanahan, 1999]. In addition, some optimizations can be performed when language-level constructs know that a particular set of fluents really represents an integer, or that a particular formula is an effect axiom.

Proposed plan

Work left to be done will focus on technical development, interface development, and evaluation.

Automated reasoning in general can easily become intractable: A few dozen lines of event calculus can ask for the optimal opening move in chess. On the other hand, designers currently do some of their prototyping by simulating game rules manually, so orders of magnitude speedup on existing practice is possible. Many queries in our preliminary work have the flavor of large-but-easy, such as simple planning in a large grid world. Some of our ongoing work aims to speed up these kinds of queries, for example by using Prolog rather than grounding-based reasoners, which in large-but-easy problems take an inordinately long time to produce the grounded problem. Nonetheless a major issue will be how to screen out a large proportion of intractable queries, and time out on the rest in a way that makes the interface still usable.

The event-calculus translation layer is a step towards a system that can be used by people other than AI researchers, but there are a number of other interface questions. One potential target of a system for reasoning about game mechanics is non-programming designers, who are used to thinking about procedural rule systems but not used to implementing them. A number of game-industry personnel have told us that

they would use a tool that allowed their designers to, rather than simply manually simulating their designs and writing up the results in a prose design document for engineers to implement, also automatically determine some properties of their design, or even regression test against known design problems. There are a number of potentially difficult design questions in this phase regarding the role of GUI versus programming-like representations, built-in versus designer-specified knowledge and representations, and so on, tying into more general literature on domain-specific design environments [Fischer, 1994].

Evaluation will be in part done by a contextual-design process [Wixon *et al.*, 1990], in which increasingly capable prototypes of the tool are deployed to game designers to use in their actual work. Because of the number of design decisions and the lack of much existing literature on how game designers work, this approach aims to integrate the evaluation as a pervasive component of system design rather than something to be done after the system's completed, ensuring that the system's goals and capabilities stay relevant to its intended audience. This process will also hopefully raise interesting AI questions, as it identifies new features designers would like.

References

- [Fischer, 1994] Gerhard Fischer. Domain-oriented design environments. *Automated Software Engineering*, 1(2):177–203, 1994.
- [Miller and Shanahan, 1999] Rob Miller and Murray Shanahan. The event calculus in classical logic: Alternative axiomatizations. *Electronic Transactions on Artificial Intelligence*, 3(A):77–105, 1999.
- [Mueller, 2006] Erik T. Mueller. *Commonsense Reasoning*. Morgan Kaufmann, 2006.
- [Nelson and Mateas, 2007] Mark J. Nelson and Michael Mateas. Towards automated game design. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, pages 626–637. 2007.
- [Nelson and Mateas, 2008] Mark J. Nelson and Michael Mateas. Recombinable game mechanics for automated design support. In *Proc. Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.
- [Nelson and Mateas, 2009] Mark J. Nelson and Michael Mateas. A requirements analysis for videogame design support tools. In *Proc. Foundations of Digital Games (FDG)*, 2009.
- [Reintjes, 1991] J. F. Reintjes. *Numerical Control: Making a New Technology*. Oxford University Press, 1991.
- [Schön, 1983] Donald A. Schön. *The Reflective Practitioner*. Basic Books, 1983.
- [Shanahan, 1999] Murray Shanahan. The ramification problem in the event calculus. In *Proc. Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 140–146, 1999.
- [Wixon *et al.*, 1990] Dennis Wixon, Karen Holtzblatt, and Stephen Knox. Contextual design: An emergent view of system design. In *Proc. Human Factors in Computing Systems (CHI)*, pages 329–336, 1990.