Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

Structure Editors and Autonomous Agents

Alexander Card
North Carolina State University

Abstract

Rational agents are becoming prevalent in many domains, from data analysis to entertainment and games. The increased prevalence of agents has evolved new tools and techniques to work with and design new agents. One such technique is system simulation. Systems simulation is a technique an author can use to imitate tasks, processes, or systems, and in particular, agents. Systems simulation has a variety of uses, ranging from simulating ecological systems to entertainment, such as interactive narratives and digital games. However, many system simulators use specialized programming languages and require prior programming experience. This causes a disconnect between individuals with limited programming experience who wish to use the simulation tools, and the software itself. New users may find the specialized languages daunting, and the initial learning process too intense for the anticipated reward. This research strives to bridge the gap between system simulation tools and users with little to no programming experience. Future work includes a corpus of narrative and autonomous agent creation tools designed for users with little to no programming experience.

Introduction

Rational agents are becoming prevalent in many domains, from data analysis to entertainment and games. The increased prevalence of agents has evolved new tools and techniques to work with and design new agents. One such technique is system simulation. Systems simulation is a technique an author can use to imitate tasks, processes, or systems, and in particular, agents. Systems simulation has a variety of uses, ranging from simulating ecological systems, to entertainment, such as interactive narratives and digital games.

However, many system simulators use specialized programming languages and require prior programming experience. This causes a disconnect between individuals with limited programming experience who wish to use the simulation tools, and the software itself. New users may find the specialized languages daunting, and the initial learning process too intense for the anticipated reward.

On a different path, agents in interactive narratives and digital games are traditionally manually authored or scripted (McNaughton et al. 2003), which may cause the agents to behave unpredictably in certain edge cases the author missed or reducing the state space of possible narratives for the agents. One proponent that is lacking in many interactive narrative agents is a belief system and theory of mind. This leads to considerably fewer narratives of certain genres, such as mystery narratives and detective narratives.

My research interests are two-fold. The current work aims to increase participation in systems simulation by introducing a linear logic based system simulator which forces the structures created in the editor to be well-formed in the syntax of the language (also called correctness by construction), and requires little to no programming experience to create and execute system simulations. Correctness by construction is common in structure editors which guide the user through the syntax, frequently preventing the user from creating a syntactically incorrect structure (Teitelbaum and Reps 1981; Kelleher et al. 2002; Miller et al. 1994)

Data analysis from from the pilot study is underway, and future studies are planned to address additional research questions, both about usability as well as comparisons to existing tools which perform similar functions with specialized languages. Continuing research expands beyond the system simulation editor to introduce a belief system and theory of mind for autonomous agents in interactive narratives, and perform studies on notability and believability. Work on theory of mind models for autonomous agents has been explored via belief implementation in (Eger and Martens 2017) and (Shirvani, Ware, and Farrell 2017), and simulation as a storytelling agent in (Evans and Short 2014).

Current Research

A Structure Editor for Ceptre

Ceptre is a rule specification language (Martens 2015) which uses linear logic to model and transition between states in the system simulation. Each state is a set of ground predicates, and the possible transitions between states are sets of instantiated rules, where the instance of the rule is satisfied by the predicates in the current state. Predicates are structures which consist of zero or more arguments, where each of these arguments is a set of objects. Rules have preconditions and effects, where each precondition or effect is a predicate which has previously been declared. For states, the

ground predicates' arguments are elements from the set in the corresponding predicate's argument list. The states gives the world a set of facts which can be transitioned from one state to another by applying a rule to the current state, a process which continues until terminated by the user or the system reaches quiescence, where no more rules can fire. Quiescence is useful for debugging simulations such as game systems, where an unintended quiescence could mean the game is unable to completed, given an initial state and the current set of rules.

The primary focus of this research is to design a tool which assists new authors of system simulations. The structure editor tool contains both an editor for the rule specification language, as well as a simulation engine to run the constructed simulation. The editor is designed to enforce correctness by construction using both static and dynamic validation on the input supplied by the simulation author, while not limiting the functionality of the language. Menus of options are used to mitigate the complexity of requiring specialized syntax for the author, alongside reducing the amount of debugging which is required of the simulation program. Alongside the enforcement of correct syntax, as structure editor allow users to design programs without needing to read or create textual code, all programming is done through the user interface. The simulation engine allows the user to execute the simulation from within the tool without any additional installations or compilations. Additionally, the engine allows the author to view the system state at each step, manually stepping through the simulation, or executing the simulation to quiescence.

Future Research

Upon finishing the data analysis for the pilot study, additional studies will be performed to answer research questions such as:

- Does prior programming experience significantly impact the ability to model and understand systems created in the Ceptre editor?
- Does prior logic experience significantly impact the ability to model and understand systems created in the Ceptre editor?
- Does the Ceptre editor allow users with little to no programming experience to create and understand models more quickly and effectively than the text-based Ceptre language?

Additionally, bringing in experienced authors of system models, doing feature analysis as well as performing a study to see if the system is more understandable using this tool than using other methods.

After the conclusion of these projects, I aim to increase the participation in interactive narrative design with believable agents. As a potential thesis project, I intend to design and implement a theory of mind model with the purpose of integrating it into an existing game engine with an accessible interface for agent behavior. This model will be built with the intent to continue broadening participation and ease of access into interactive narrative and digital game design, allowing designers to construct autonomous agents based on a theory of mind model instead of a hierarchical task network or a model learned from demonstration. One distinct advantage a theory of mind model would provide over a hierarchical task network or a learned model is the ability to maintain higher order beliefs about the world and other agents in the world. This autonomous agent model combined with the previous projects would combine to form a corpus of open-source tools, accessible via an existing game engine, which could be used to assist authors and creators; decreasing the complexity of the authoring experience while increasing the ease of entry into the field for individuals with little to no programming experience.

To close, I intend for my research to assist new authors and designers, while providing features that experienced authors would expect. I aim to create a corpus of tools which assist with the creation and understanding of the systems used in interactive narratives and digital games.

References

Eger, M., and Martens, C. 2017. Character beliefs in story generation. In *Intelligent Narrative Technologies 10, AIIDE 2017 Workshop WS-17-20*, 184–190.

Evans, R., and Short, E. 2014. Versu—a simulationist storytelling system. *IEEE Transactions on Computational Intelligence and AI in Games* 6:113–130.

Kelleher, C.; Cosgrove, D.; Culyba, D.; Forlines, C.; Pratt, J.; and Pausch, R. 2002. Alice2: programming without syntax errors. *User Interface Software and Technology*.

Martens, C. 2015. Ceptre: A language for modeling generative interactive systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

McNaughton, M.; Redford, J.; Schaeffer, J.; and Szafron, D. 2003. Pattern-based ai scripting using scriptease. In *Proceedings of the 16th Canadian Society for Computational Studies of Intelligence Conference on Advances in Artificial Intelligence*, AI'03, 35–49. Berlin, Heidelberg: Springer-Verlag.

Miller, P.; Pane, J.; Meter, G.; and Vorthmann, S. 1994. Evolution of novice programming environments: The structure editors of carnegie mellon university. *Interactive Learning Environments* 4(2):140–158.

Shirvani, A.; Ware, S. G.; and Farrell, R. 2017. A possible worlds model of belief for state-space narrative planning. *Artificial Intelligence and Interactive Digital Entertainment*.

Teitelbaum, T., and Reps, T. 1981. The cornell program synthesizer: A syntax-directed programming environment. *Commun. ACM* 24(9):563–573.