The AIIDE-17 Workshop on
Experimental AI in Games
WS-17-19

A Sandbox for Modeling Social AI

Ethan Robison
Northwestern University
ethanrobison@u.northwestern.edu

Abstract

We present a system for quickly prototyping AI code for modeling social interaction among the simulated residents of a skyscraper. The system is built on top of a commercially released indie game but replaces its character AI with a general-purpose logic programming language. These two features together simplify the more tedious parts of creating prototype AI code from scratch, enabling more effort to be focused on the meatier parts of research.

Introduction

One of the obstacles facing game AI researchers is the sheer scope of effort involved in making a game, most of which has little or no publishable research content: UI design, playtesting, music and sound effects, and art and animation are notoriously difficult facets of game design. While generic assets for certain genres, such as First-Person Shooters, or (especially) platformers exist, finding appropriate assets for other genres can be a challenge. For other genres, finding assets-especially aesthetically consistent ones-can be nearly impossible. For social games like *The Sims 3* (Electronic Arts Inc., 2009) or visual novel style games like *Long Live the Queen* (Hanako Games, 2012), researchers are often forced to get by with a good deal of willing suspension of disbelief on the parts of the play testers when it comes to accepting their assets as suitable.

This compounds with a further problem for anyone interested in social modeling in games. Researchers using rules-based inference systems—such as Comme il Faut (Joshua McCoy, 2014) or Versu (Richard Evans, 2014)—as the foundation of their AI modeling may find themselves constructing complex scripting languages from scratch every time they try a new idea. However, for those experimenting with new ideas, the overhead of starting from scratch every time is a greater burden than the advantages provided by the flexibility of making everything themselves.

Figure 1: Screenshot of the AI Sandbox

Architecture

In this paper, we describe a system for rapidly prototyping AI code within the framework of the commercially released game *Project Highrise* (SomaSim LLC, 2016). *Project Highrise* is a simulation game in which players "build and manage a modern-day skyscraper," (SomaSim LLC, 2016). In the game, players develop and arrange a skyscraper as they see fit, and the tenants and visitors to the skyscraper go about their lives-working, shopping, eating, and sleeping in the simulated world. The AI Sandbox leaves most of the systems in *Project Highrise* intact, removing just the AI that controls the NPCs' decision-making process and replacing it with our own code, written in the logic programming language *BotL* (Horswill, 2017)—a simplified logic programming language similar to Prolog (W. F. Clocksin, 2003). For the sake of writing code for the Sandbox, BotL works much the same as Prolog, except for performance variation and slight syntactic differences.

This solves—at least for researchers doing work on social interaction—the problems of assets, UI design, and hairy things like serialization, because *Project Highrise* had those already in place. The system is designed to make programming the entity AI the only work that researchers need to do.

The system is developed in Unity3D (Unity Technologies, 2017) and written in C#.

The BotL-C# Interface

While the BotL interpreter and the code for *Project Highrise* are both written in C#, the code that the interpreter runs is written in plain BotL. This is to allow end users (who will not have access to the *Project Highrise* codebase) to edit the BotL code that controls the NPCs. Although BotL code can indirectly call any C# method in the codebase, this is done via reflection, which is inefficient. To bridge this gap, and to prevent users from having to learn the structure of the *Project Highrise* API, a series of registers exists which serve as neutral ground to which both parts of the code have access.

Whenever the *Project Highrise* engine needs BotL to make a decision—for instance at start-up, whenever an NPC needs to decide on its next action, or to access any of the information stored in the internal logic database—the C# code writes any relevant information to these registers and the BotL code reads this information, runs its calculations, and writes its results back to the registers.

In this way neither side of the codebase needs to know what specifically the other side is doing, just that the end results will end up in a designated location. This helps to decouple the halves from one another and to increase efficiency.

The Exclusion Logic Database

Most of the engine code is deliberately not imperative; since some operations are run dozens of time a frame, side effects could be problematic. However, this makes tracking the internal state of the game challenging. The Sandbox uses an Exclusion Logic database (Evans, Introducing Exclusion Logic as a Deontic Logic, 2010) to hold any information that the user wants to keep track of.

It is worth noting that the BotL/C# interface knows nothing about level of access. More explicitly, *everything* can read and write to every part of the database. This is not as dangerous as it first seems, since end users are only editing their own BotL code and so are able to govern their own editing of the exclusion logic database. The C# side of the code does only very basic editing of the database at start-up, before any user BotL code has been run.

Controlling NPCs

Since the purpose of the Sandbox is to facilitate writing custom NPC control code, it is a little confusing to talk about the code for controlling the NPCs that is built into the system already. Specifically, when we refer to controlling the NPCs, we mean the C# logic that decides what to do when the user-written BotL code instructs it to do a specific task.

The logic consists of three main parts: handling user input, handling per-NPC actions, and making broader decisions about the state of the game world in general.

The Sandbox allows for keyboard and mouse input, both of which are left as empty predicates for the user to fill in as they please.

Decision Making

The replacement for the vanilla AI is two-tiered: first, at a lower level, each NPC is also capable of deciding things on their own. Whenever a NPC finishes their last action, they call the `makeDecision` procedure, which informs them of the next action that they need to take. These are the possible actions that an NPC can take:

Idle: stand around and do nothing for a specified amount of time;

Go somewhere: move from one place to another place, or to the current location of another entity (a NPC, a building, a water cooler, etc.);

Say something: pop up a speech bubble with a plaintext string on it and notify the NPCs around it that it spoke;

Change color/clothing: part or all of the NPC's sprite is changed in color, a good way to reflect changes in internal state without text;

Die: the NPC is removed from the game (although any pertinent information is not automatically cleared from the exclusion logic database).

These actions are tailored to games centered around social interaction. The real expressiveness of the system stems from the user written BotL code under the hood. In other words, though it seems like the number of actions that an NPC can take is small, the context of those actions provides the richness.

The Director

In addition to per NPC actions, every frame, "director" decisions are made. These are contained in the procedure `directorDecision`, which imperatively handles everything too tricky (or too global) for the NPCs to do themselves. This could be weather, actions involving multiple NPCs, changing the time of day, or creating/removing in-game entities.

It is important to note that "director" is a bit of a misnomer: the director is not a drama manager; rather, director decisions pertain to anything in the game world that might involve multiple NPCs or fall outside of the purview of any one individual. Nor is the director making decisions *for* NPCs. Individual NPCs make their own decisions whenever they finish an action or something happens around them (such as another NPC saying something).

Implementation Status

The AI Sandbox is a work in progress, being a combination of a recently released indie game and a recently created logic programming language. In coming months, the underlying architecture of the system will be re-worked, since the current structure was put together quickly over a short period.

What we have managed thus far is a heavily simplified version of the needs-based AI used in *The Sims 3* (Evans, Modeling Individual Personalities in The Sims 3, 2010) done in approximately 120 lines of code. NPCs walk around, order burgers and coffee, "go to sleep," and, if they cannot make it to food, water, or bed; they die. While not quite as fun as its commercial inspiration, it serves as a morbid proof of concept.

Additionally, we have an experimental game in which NPCs are color-coded according to what "type" of person they are. If too many of their neighbors are a different color than they are, the NPCs move to a different part of the apartment complex. This is a quick rendition of the *Parable of the Polygons* (Vi Hart, 2017) and is about 150 lines long. As an illustration of BotL's pithiness, here is the snippet of BotL code that determines whether a given NPC wants to move:

```
wants_to_move(Peep)<--
  get-different(Peep, DList),
  DList.Count > 3;
get-different(Peep, Different) <--
  listof(X:(nearby(Peep,X),
  different(X, Peep)), Different);
```

Both systems run smoothly with over 50 NPCs on screen, although it quickly becomes difficult visually to tell which NPC is which.

Future Work

In the future, we would like to make more miniature versions of relevant game AI systems, such as CiF (Joshua McCoy, 2014), Prom Week (Josh McCoy, 2013), or Versu (Richard Evans, 2014); or items in the spirit of Schank and Riesbeck's Five Programs Plus Miniatures (R. C. Schank, 1981).

Limitations

The advantages to using a commercial game center mostly around already having assets available, a strong structure upon which to build experimental code, and freedom to work on research instead of on foundation. There are disadvantages, however, to using someone else's product as the starting point for custom research.

For *Project Highrise*, this means being limited to the sprites and sound effects that come with the system. But it also means being limited to the sorts of scenarios that a skyscraper management simulator is likely to include. This is not to say that the barrier to entry is not lowered, just that it is important to keep in mind that this system is designed for writing prototype AI models, not fully-fledged custom playable experiences.

Conclusion

Part of the advantage of systems like this is that they enable the quick creation of prototypes for specific genres of games for AI research. By facilitating the construction of inference systems, and by providing already complete art and UI, the AI Sandbox takes care of some of the heavy lifting that researchers need to do when they test new AI ideas.

The system will probably never work for most AI categories outside of social interaction; and is certainly ill-suited to other AI methods such as behavior trees, deep learning, etc. However, for researchers seeking to model social interaction systems using symbolic reasoning systems, the AI Sandbox should serve as a solid starting point for more robust future efforts.

Acknowledgements

The author was given permission to use the *Project Highrise* codebase for research purposes by its creator, Robert Zubek. Those interested in the architecture of *Project* Highrise should see Rob's paper on the topic (Zubek, 2017).

BotL was created by Ian Horswill, who graciously adjusted its features to suit the Sandbox's needs.

References

Electronic Arts Inc. (2009). The Sims 3.

Evans, R. (2010). Introducing Exclusion Logic as a Deontic Logic. *Deontic Logic in Computer Science*, 179-195.

Evans, R. (2010). Modeling Individual Personalities in The Sims 3. *GDC.* San Francisco, CA.

Hanako Games. (2012, June). Long Live the Queen.

Horswill, I. (2017). Retrieved from https://github.com/ianhorswill/BotL

Josh McCoy, M. T.-F. (2013). Prom Week. *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, (pp. 207-208).

Joshua McCoy, M. T.-F. (2014). Social Story Worlds With Comme il Faut. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 97-112.

R. C. Schank, C. K. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures (Artificial Intelligence Series) 1st Edition.* Routledge.

Richard Evans, E. S. (2014, June). Versu—A Simulationist Storytelling System. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES*, 113-130.

SomaSim LLC. (2016). Project Highrise. Chicago, Illinois, United States.

SomaSim LLC. (2016). *Project Highrise | SomaSim Games*. Retrieved from SomaSim Games: http://www.somasim.com/highrise/

Unity Technologies. (2017). Retrieved from Unity 3D: https://unity3d.com/kr/

Vi Hart, N. C. (2017). *Parable of the Ploygons*. Retrieved from http://ncase.me/polygons/

W. F. Clocksin, C. S. (2003). In C. S. W. F. Clocksin, *Programming in Prolog, 5th ed.* Springer-Verlag.

Zubek, R. (2017). 1000 NPCs at 60 FPS. In S. Rabin, *Game AI Pro 3: Collected Wisdom of Game AI Professionals* (pp. 403-410). A K Peters/CRC Press.