Versu—A Simulationist Storytelling System

Richard Evans and Emily Short

***Abstract*—Versu is a text-based simulationist interactive drama. Because it uses autonomous agents, the drama is highly replayable: you can play the same story from multiple perspectives, or assign different characters to the various roles. The architecture relies on the notion of a *social practice* to achieve coordination between the independent autonomous agents. A social practice describes a recurring social situation, and is a successor to the Schankian script. Social practices are implemented as reactive joint plans, providing affordances to the agents who participate in them. The practices never control the agents directly; they merely provide *suggestions*. It is always the individual agent who decides what to do, using utility-based reactive action selection.**

***Index Terms*—Exclusion logic, interactive drama, multiagent simulation, script, social practice.**

I. Introduction

VERSU is a text-based simulationist interactive drama. It is available now for iPad (and soon for other devices). The player starts by choosing an episode. At the time of writing, the Versu platform contained episodes from three genres: we have various episodes from a Jane-Austen-esque Regency England, some episodes from a modern office comedy, and episodes from a lighthearted fantasy world. Once the player has chosen an episode, and selected which character she wants to play, the game starts.

The player is presented with text and static images describing the current situation (see Fig. 1). There are two buttons: "act" and "more." If she presses "more," the nonplayer characters (NPCs) will make decisions autonomously. She can keep pressing "more" if she just wants to sit back and watch the situation unfold. At any time, she can interrupt the autonomous action and interject by pressing the "act" button. This brings up a large array of choices. She chooses an action, the NPCs respond autonomously, and play continues.

Versu is a platform for interactive drama (focusing especially on lighthearted comedy of manners) and supports multiple styles and genres of fiction. In this game-play example, from our Regency England setting, the naive young debutante Lucy and the rakish poet Brown are both dining with the Quinn family.

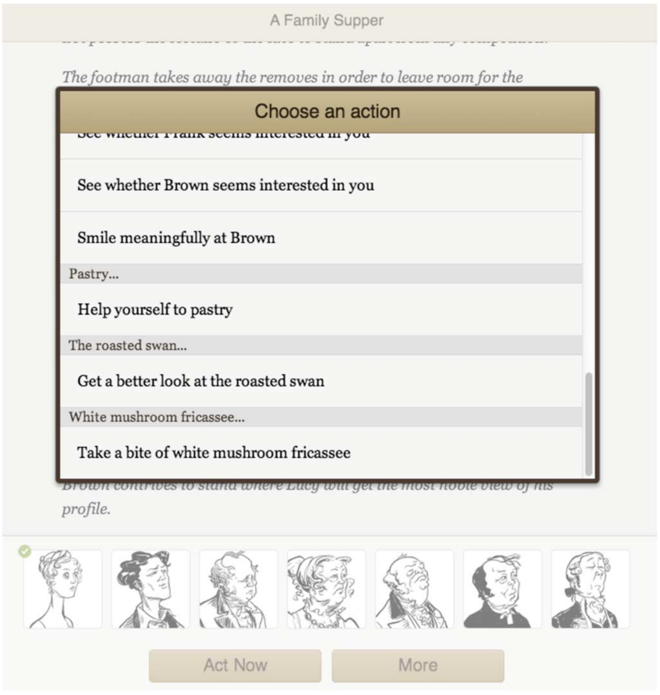

A Family Supper
The footman takes away the removes in order to leave room for the
Choose an action
See whether Frank seems interested in you
See whether Brown seems interested in you
Smile meaningfully at Brown
Pastry...
Help yourself to pastry
The roasted swan...
Get a better look at the roasted swan
White mushroom fricassee...
Take a bite of white mushroom fricassee
Brown contrives to stand where Lucy will get the most noble view of his profile.
Act Now
More

Fig. 1. Screenshot of Versu in action.

"The first course is ready laid out on the table, with salmon at one end and a dish of turbot at the other, together with a dish of macaroni, muffin pudding, and larded sweetbreads.

Lucy is at the Quinn dinner table, with their other guests.

The meal has only just begun, but a curious sort of unease hangs over the room; a sense of apprehension whose source she cannot identify.

Mrs. Quinn gulps her wine down enthusiastically.

Mr. Quinn oversees the serving of a turtle soup from the great tureen in front of him."

At this point, Lucy has several options to participate in the dinner, by sipping her wine or having something to eat. She also has interpersonal options, including the ability to check out other people and see whether they seem to be attracted to her. She chooses to look at Brown.

"Lucy silently studies Brown.

She concludes that he cannot be wholly indifferent to her charms."

Encouraged by what she has seen, she will now give him a flirtatious sort of look.

Manuscript received October 05, 2012; revised May 06, 2013; accepted October 08, 2013. Date of publication October 25, 2013; date of current version June 12, 2014.

The authors are with Linden Lab, San Francisco, CA 94111 USA (e-mail: richardprideauxevans@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2013.2287297

> "Lucy gives Brown a demure yet intriguing look under her lashes.
>
> Brown smiles at Lucy in a way that conveys a strong encouragement."

Success! Lucy has established an initial relationship with Brown.

> "Brown rises from the table, taking his glass of wine.
>
> Brown (to Lucy): You are truly a Muse!
>
> Brown: I apologize, but I must go: I feel a sonnet coming on.
>
> He gives a flourishing parting bow. He swaggers out of the dining room on his way to the study."

Now other characters react to Brown's departure.

> "Mr. Quinn: Perhaps Brown is sickening for something."

Now Lucy can choose how to interpret Brown's abrupt exit. She could be dismayed by his poor manners, or intrigued by his decision to go write a sonnet. We select for her to be sad that he has left.

> "Lucy watches Brown go with a disappointed expression in her eyes.
>
> Frank expresses a similar concern over Brown's health."

Lucy's sadness about Brown going means that her eating options become centered on expressing unhappiness.

> "Lucy picks joylessly at the dish of buttered macaroni."

In addition, because everyone is talking about Brown's ill health and strange behavior, she has the conversational option to defend him by mentioning a positive evaluation she has made about him.

> "She defiantly says that Brown is so handsome."

Because Lucy has expressed admiration for Brown, and Frank is the jealous type, we trigger another one of the situations available in this dining scene. Frank challenges Lucy about her crush.

> "Frank (to Lucy): Oh, enough! Pray spare us more of your calf-eyes.
>
> Frank (to Lucy): Do you truly think that a man like Brown would be so captivated by a girl like you as to marry you? Do you not realize that he has his choice of London? And he has hardly lived as a monk, I may add!"

II. A Simulationist Interactive Drama

Versu is an interactive drama. It is an improvisational play, rather than an interactive story. The player is encouraged to perform her character, to improvise within the dramatic situation that she has been thrown into. The smallest comment, the slightest look, even not saying something—these moment-to-moment actions are noticed by the other participants and amplified. In this respect, Versu resembles Façade. But while Façade uses an architecture built around beats and joint behaviors (JBs), Versu is an agent-driven simulation in which each of the NPCs makes decisions independently.

When building an interactive narrative system, one of the fundamental design decisions is whether the individual agents or a centralized drama manager (DM) gets to decide what happens [21]. At one end of the spectrum, a strong story system is one in which the DM makes all the decisions. The NPCs have no individual autonomy; they are just puppets of the DM. At the other end of the spectrum is the strong autonomy system, in which the NPCs make decisions based on their own individual preferences, unaware of authorial narrative goals. Façade occupies the middle ground on this axis: a centralized DM chooses the next beat, but the individual agents and JBs have some limited control over how the beat is played out.

Versu, by contrast, takes the strong autonomy approach. Each character chooses his next action based on his own individual beliefs and desires. There is a centralized DM, but it is rare indeed for the DM to override the characters' autonomy and force them do something. Instead, the DM typically operates at a higher level—by providing suggestions, or tweaking the desires of the participants.

Why did we choose a strong autonomy approach in Versu? There were two main reasons. First, a true simulation provides much more opportunity for replayability. In Façade, many of the behaviors are hard-coded to the particular characters who are involved in them. In Façade, there is no general making cocktails activity. Instead, there is the particular activity of Trip making cocktails. Because the scripts and characters are entangled together, it would be hugely nontrivial to replace a character in Façade with another (replacing Trip with Captain Kirk, say). It would involve rewriting most of the behaviors.

In Versu, the social practices are authored to be agnostic about which characters are assigned to which roles. This means that we can assign various different characters to the roles, and everything just works. In a Versu-authored version of the Façade situation, you could play Grace, or Trip, or the guest. You could assign different characters to the various roles, and see what happens.

The simulation makes a clear distinction between roles in a story and the characters playing those roles. A romance might have a hero, a heroine, a friend, and a jealous rival. The player is free to assign various characters to those roles. She could make Mr. Darcy the hero, and Elizabeth Bennett the heroine; or, she could make Mr. Collins the hero, and Miss Bates the heroine. The episode will play out very differently depending on which characters are playing which roles. If there are n roles and $k(> n)$ characters to play those roles, we have $k!/(k-n)!$ permutations.

Relatedly, the scripts in Façade assume that the player is always playing the guest, and the hosts are always played by NPCs. If you wanted to rework it so that you could play Trip or Grace, this would involve a major rewrite. In Versu, because the social practices are authored to be character agnostic, you can play the same story from multiple perspectives. You can try out the job interview from the perspective of the interviewee, or the interviewer. You can even play it as the DM. An episode with two roles could be played by two humans in multiplayer, one human playing either role, or both roles played by the AI.

More generally, an episode with n roles has 2^n permutations of player-NPC assignments. The first reason, then, that we committed to a simulationist architecture is because we wanted to maximize replayability.

The second reason for choosing strong autonomy is that a simulationist architecture allows the player more control over the outcome. A simulated system has clear rules which the player can learn and internalize. Once she has confidence that she has understood the internal mechanisms, she can use these to anticipate the consequences of future action, and plan how to achieve her goals. A nonsimulated system risks being just a series of arbitrary puzzles, in which the player is forced to guess the changing whims of the designer. A simulation, by contrast, uses the same models repeatedly. The player can build up confidence that she understands the underlying system—and increased understanding can yield increased control.

III. Addressing the Design Questions Raised by Façade

Some of the crucial initial design decisions in Versu were made by looking hard at what Façade achieved. We focused on Façade in particular because it is such a substantial achievement. In [8], when evaluating the successes and failures of Façade, Mateas and Stern mention three outstanding issues in particular.

- The speed of content production and global agency. Because of the intricate animation overlaying and parallel behaviors which needed to be authored for most actions, adding a new piece of content to Façade was a time-consuming task. In the end, after three plus years in development, they only had time to author 27 beats. The amount of global agency (the ability for the player to affect the overall arc of the story) is limited by the amount of content, so in the end, the player did not have as much ability to affect the outcome of the story as the authors had initially hoped.
- Feedback. Façade involved three social "head games," played one after the other (an affinity game, a hot-button game, and a therapy game). By design, the state of each game was not communicated directly (via numbers of spreadsheets or sliders), but indirectly by gesture and tone of voice. (The authors wanted to maintain the sense that this was a drama rather than a computer game.) But this design decision made it very hard for the player to tell the state of the simulation.
- Interface. In Façade, the player can type any text she wishes. But the parser will attempt to shoehorn all the player's sentences into one of 30 parametrized discourse acts. Unfortunately, the player's utterance cannot always be fitted into one of these 30 specific actions, and even if it could, the parser often cannot see how it could. The player can feel like she is fighting the parser, rather than using it effortlessly as a tool to communicate with.

We tried to make sure that Versu had good answers to the three issues which Mateas and Stern identified.

- The speed of content production. To speed up content production, we eschewed Façade's 3-D procedurally animated characters for (procedurally generated) text and static images.
- Feedback. We added various visualizations to the user interface (UI) so the player can see at a glance the state of the simulation.
- Interface. We replaced the parser with a simpler menu interface. The affordances provided by each social practice are displayed explicitly to the player. She acts by clicking on a button.

We will go through each of these three decisions in turn. But first, a point of clarification: it may look like each of these responses to issues in Façade are just simplifications that allowed us to avoid the hard problems that Mateas and Stern were brave enough to tackle head on. But these decisions were not just a cop-out—they were pursued in order to allow us to be more ambitious in our simulational goals. We will return to this point repeatedly below.

A. Text Output

Façade wholeheartedly embraced the "holodeck" vision of interactive drama in which character behavior is rendered realistically in multiple modalities: 3-D characters, parallel animation, recorded speech. This, of course, is one of the main reasons why it took so long to add a new behavior: it is time consuming getting the animations blending correctly and achieving synchronization with other actors.

Now there is nothing wrong with realistic rendering of characters, but neither is it necessary. We agree with Salen and Zimmerman [26] that the level of immersion does not necessarily increase with extra levels of realism. A text output can be just as immersive as a 3-D animated environment. To think otherwise is to be seduced by what Salen and Zimmerman call the "immersive fallacy." Versu does not use fancy 3-D animation or voice actors; the output is dynamically generated text and static images. Text output certainly makes it quicker to produce behavior (later, we will describe how we managed to produce an order of magnitude more behaviors than Façade in a shorter time frame). But that is not its only advantage. It is not just that text is cheaper than 3-D animation; it is also more expressive.

1) Text as an Expressive Medium—Interiority: Before we started developing Versu, when working on *The Sims 3*, we came across a revealing situation which highlighted the advantages of text output for revealing interiority. There was a chronically shy Sim who was hosting a party. Some of the guests had rung the door, and were waiting to be ushered in. The shy Sim was sitting on the couch, deciding what to do. Debugging his internal state, we could see that he was conflicted: the norms of social propriety dictated that you should answer the door when invited guests come over. But his own chronic shyness gave him a strong countervailing reason for not answering the door: he very much wanted to be alone. Within the decision-making system that we were using, the Sim had a hard choice between answering the door and refusing to do so. But neither of these options captured what the Sim wanted to express: what should have happened is that the Sim answered the door reluctantly. Here, we express his internal conflict through an adverbial modifier.

Now, in a 3-D game with animated polygonal characters, adding an adverbial modifier to an action is a hugely expensive process: we would need a separate animation for answering the door reluctantly, and we would also need a separate walk cycle for reluctant walking. Given a large set of animated actions, each adverbial modifier we add would require a prohibitively large number of additional animations. But in a text game, adverbial modifiers are much cheaper: we can modify a verb by simply appending an adverb to the sentence. Adverbial modifiers are useful in many ways: we can use them to express internal state, to express the reason for action, and also to express individual personality.

2) Text as an Expressive Medium—Individuality: We wanted each character in Versu to have a unique personality and we wanted their individuality to be expressed throughout their actions. Text output made it feasible for each character to have a unique text override for many actions. For example, in most 3-D games, each character uses the same generic walk cycle. But in Versu, each character has an individual way of walking: Brown swaggers, Frank Quinn walks ponderously, George Wickham strides, Lady Catherine hobbles, while the pug dog waddles.

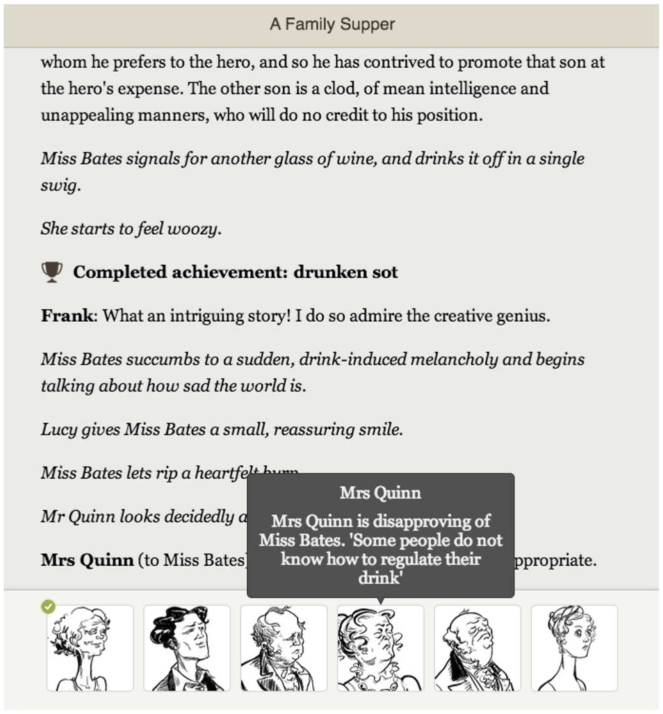
B. Feedback

When designing ways to help the player understand the social simulation, we were guided by Wardrip-Fruin's concept of the Sim City effect [34]. When playing a game which simulates some aspect of experience that the player is already familiar with, the player starts by using her own model of how it works. But the simulation will inevitably diverge from reality in various ways. If things go badly, the divergence between the player's understanding of the phenomenon and the simulation's model of the phenomenon will prevent the player from understanding or manipulating the system. The Sim City effect occurs when the user interface helps the player to transition from her original model of how the thing actually works to how the simulation models it. If this works properly, the player ends up with an accurate model of how the simulation models the phenomenon, without having had to read a manual or a textbook.

Our simulation is based on fine-grained emotional states, relationships, and social practices. We made sure that the user interface exposed these to the player transparently.[1]

To help the player understand the characters' moods and relationships, we added a portrait of each character at the bottom of the screen (see Fig. 2). Each character has various emotional states he can be in (based on Ekman's typology [4]), and each character has a different portrait for each emotional state. When the player clicks on a character portrait, she sees why that character is in that particular mood: each character remembers who the emotion is directed toward (e.g., I am annoyed with Brown), and the event which prompted the emotional change (e.g., Brown's insult).

Our simulation is unusual in that there are multiple independent social practices running concurrently (this is described in detail below). To help the player understand the state of the various social practices that are currently in play, we organized


A Family Supper

whom he prefers to the hero, and so he has contrived to promote that son at the hero's expense. The other son is a clod, of mean intelligence and unappealing manners, who will do no credit to his position.

Miss Bates signals for another glass of wine, and drinks it off in a single swig.

She starts to feel woozy.

Completed achievement: drunken sot

Frank: What an intriguing story! I do so admire the creative genius.

Miss Bates succumbs to a sudden, drink-induced melancholy and begins talking about how sad the world is.

Lucy gives Miss Bates a small, reassuring smile.

Miss Bates lets rip a heartfe

Mr Quinn looks decidedly a

Mrs Quinn (to Miss Bates ppropriate.

Mrs Quinn

Mrs Quinn is disapproving of Miss Bates. 'Some people do not know how to regulate their drink'

Fig. 2. Each character's emotional state is displayed along with explanatory text.

the affordances around the practices which initiated them. For example, if the player's character is in the middle of a dinner party, and Brown has just made a rude remark, there will be two social practices running concurrently: the dinner party (providing affordances to eat, drink, etc.) and the current conversation (providing affordances to disapprove of Brown, forgive him, etc.). The affordances are arranged in categories, grouped by the social practice that instantiated them, so that the player begins to understand the underlying simulation state. The text for each social practice is carefully worded to display its current state.

IV. Architecture Overview

Our simulation is built up out of two types of objects: agents and social practices (see Fig. 3).

A social practice describes a type of recurring social situation. Some social practices (e.g., a conversation, a meal, a game) only exist for a short time, while others (e.g., a family, the moral community) can last much longer.

A practice coordinates agents via the roles they are playing. For example, a greeting practice sees the two participants under the descriptions of greeter and recipient.

The main function of the social practice is to describe the actions the agents can do in that situation. A greeting practice, for example, tells the greeter how he can greet the recipient. It also tells the recipient the various ways she can respond.

The practice provides the agent with a set of suggested actions, but it is up to the agent himself to decide which action to perform, using utility-based reactive action selection. This is described in Section IX.

[1]A danger with exposing the simulation internals is that the experience starts to seem less like a drama and more like a computer program. We worked hard to make sure that the emotions, relationship states, and social practices were exposed in a way that kept the player immersed within the world we were creating.

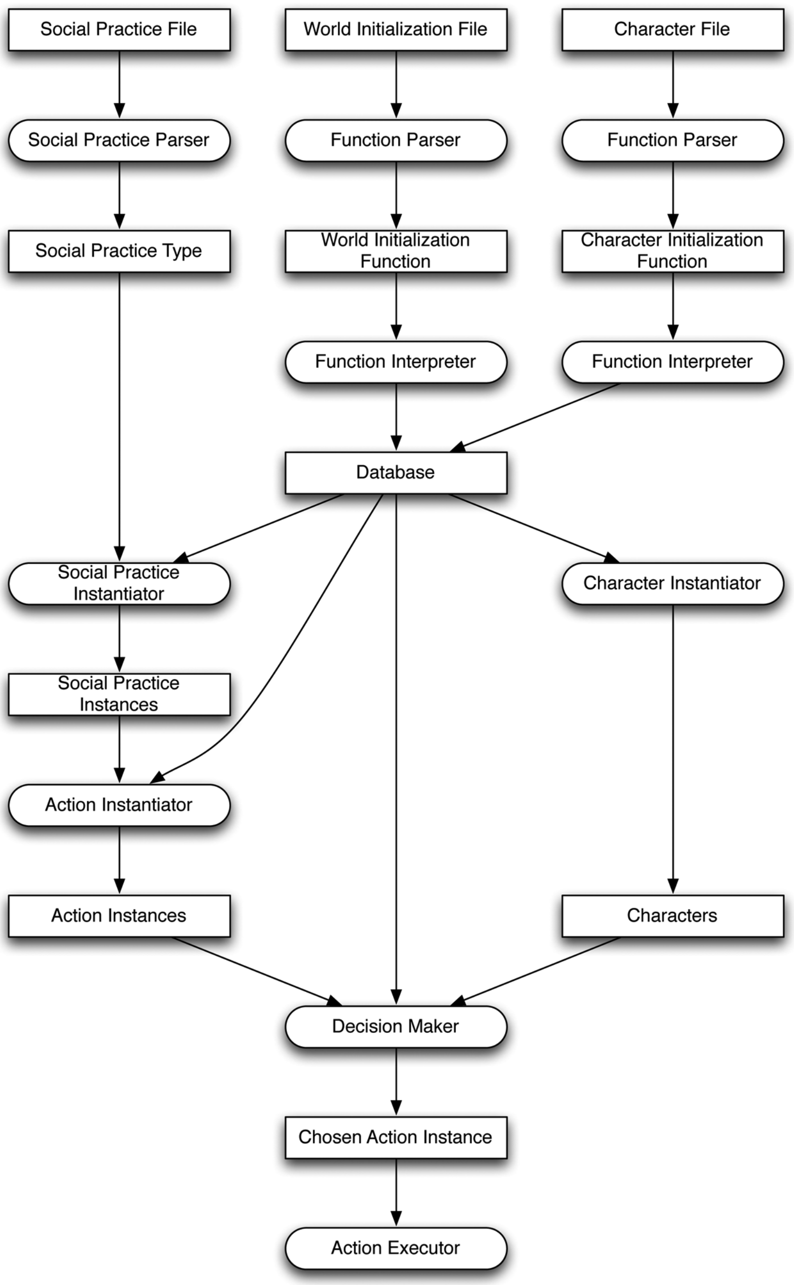


Fig. 3. The architecture of the simulator. Data is placed in rectangles, and processes in ovals. This diagram shows the information flow when an NPC makes a decision. The same architecture is used for player choice—except the Action Instances are sent directly to the user-interface, rather than to the Decision Maker.

In Versu, we allow multiple practices to exist concurrently. During a dinner party, for example, there will be multiple practices operating at once:

- eating and drinking (and commenting on the meal);
- the conversation about politics;
- the rising flirtation between Frank and Lucy;
- responding to the fact that Mr. Quinn has spilled the soup.

Each of these practices provides multiple affordances. The agent's set of options is the union of the affordances from each of the practices he is participating in.

Some practices are organized into states, so that they can provide different affordances in different situations. But a social practice is significantly more powerful than a finite-state machine, in two main ways. First, each practice can store arbitrary persistent data,[2] while the only memory a state machine has is the state it is in.[3] Second, the only possible effect of a finite-state machine's action is transitioning from one state to another. A Versu action can do much more than change the state of the practice: performing an action can result in any sentence being added to the world database. The results of adding new sentences can be that relationships are updated, new beliefs or desires are formed, old practices are deleted, or new practices are spawned.

[2]The whist game, for example, stores which cards have been assigned to which players, which suit is trumps, whose turn it is to play, and the score. See [10] for an early description of how practices need their own memory to "keep score."

[3]Because the finite-state machine's memory is limited by the number of states, it is not Turing complete.

V. The Architecture

In this diagram, boxes represent data and ovals represent procedures which operate on that data.[4]

We will start from the top of the diagram and work downward. Creating an episode involves three types of scripts:

- scripts defining the social practices that can be instantiated during the episode;
- scripts defining the initial state of the characters that may be participating in the episode;
- a script defining the initial world state.

All scripts are authored in a high-level domain-specific-language designed specifically for this simulation: Praxis.[5]

When the episode starts, we execute the world initialization function, and then execute the character initialization function for each character who is selected to participate in the episode. At this point, the database contains the initial world state.

Social practices can be parameterized by arguments, so they can be multiply instantiated with different values for the arguments. Each practice type specifies a set of actions together with their preconditions. The actions available to a character at any given moment are determined by all the actions of the currently instantiated practices whose preconditions are satisfied.

Agents score each available action (see Section IX) and then execute the highest scoring action. Actions may modify the database and/or generate text.

The diagram is almost symmetric, with social practices on the left-hand side, and characters on the right-hand side. The reason it is not entirely symmetric is that there are multiple social practice instances for each social practice type (e.g., there may be two instances of the whist game practice type occurring simultaneously in two different rooms), but there is only one character instance for each character file.

The diagram does not call out the DM explicitly. This is because each episode's DM is implemented as a special type of social practice. A DM is not a new type of entity: it is just a particular type of practice.[6]

VI. How We Represent the World

A. The World Is Everything That Is the Case

In the Praxis system, the simulation state is entirely determined by a set of sentences in a modal logic.[7] The set of sentences is the complete simulation state. There are no objects, or pointers, as traditionally conceived. Representing the world state as a set of sentences has a number of advantages.

- Visibility. The entire world state is completely open to inspection. Nothing is hidden. If you want to know, for example, if there is anyone in the current simulation state who is in the same room as someone they dislike, you just need to form the query. There is no need to ever write code to access the state of the world because the entire state of the world is represented in a uniform manner and is already open to view.
- Debuggability. You can place logical breakpoints to detect which practice is responsible for making a fact true. This is much more powerful than traditional code breakpoints or data breakpoints.
- Serializability. Because the world state is represented in a uniform manner, it is trivial to serialize and deserialize the world state.

The main advantage is visibility. We found, in previous simulations we have worked on, that the main factor which makes it hard to improve the quality of the AI is the difficulty in seeing the entire simulation state. Bugs lurk in the darkness. In this architecture, where the world is a set of sentences, nothing is hidden.

The simulator comes with a runtime inspector, which gives developers complete access to the internal state of the simulation. The inspector allows them to:

- find out what is true;
- print all facts about an object, an agent, or a process;
- find all instantiations of a term with free variables (e.g., find me everybody who Brown has annoyed);
- find out why an action's preconditions have failed;
- find out what is causing a fact to become true.

B. Sentences, Practices, Agents, and Affordances

The world is a set of sentences in a formal logic. Sentences which contain the distinguished `process` keyword make practices active. Sentences containing the distinguished `agent` keyword makes agents active. These practices provide affordances to the agents. When an agent chooses to perform one of these affordances, the world state is changed: sentences are added and removed from the database. The database changes can change the set of available practices, and the loop begins again. In summary:

- sentences being true make practices and agents active;
- practices propose affordances to agents;
- performing affordances updates sentences in the database (which might mean that different practices are now available, providing different affordances).

VII. Exclusion Logic

The world state is defined as a set of sentences in a logic called exclusion logic [5]. This modal logic is particularly well suited for modeling simulation state in general, and social practices in particular.

[4]Sometimes if a procedure is used in multiple ways (e.g., the function interpreter which sometimes interprets a world initialization function and sometimes a character initialization function), that procedure is drawn twice, to make it easier to see the flow of data through the system.

[5]Praxis contains a number of decisions which are logically independent: 1), the decision to model social practices as first-class objects; 2) the use of a strongly typed logic-formalism to model simulational state; and 3) the use of exclusion logic as the logic of choice.

[6]The DM is described in Section XI.

[7]A modal logic, in the broad modern sense, is a logic which contains non-truth-functional operators for talking about relational structures (see [1, p. xi]). Previously, the term "modal logic" was restricted to logics which treat the standard modalities of necessity, possibility, knowledge, belief, time, deontic modalities, etc.

Given a set S of symbols, the literals[8] in exclusion logic are defined as all expressions of type X in

$$X ::= S \mid S.X \mid S!X.$$

We use the "!" and "." operators to build up trees of expressions. For example:

- brown.sex!male: Brown is male;
- brown.class!upper: Brown is upper class;
- process.dinner.dining_room.participant.brown: Brown is one of the people having dinner in the dining room;
- process.dinner.dining_room.participant.lucy: Lucy is one of the people having dinner in the dining room.

Asserting that $A.B$ is claiming both A and one of the ways in which A is B. Saying that $A!B$, by contrast, is to say that B is the only way in which A is the case. The semantics of exclusion logic are described in [5].

The two things that distinguish exclusion logic from traditional predicate logic are:

- the ability to directly represent tree structures;
- the exclusion operator.

A. Exclusion Logic Literals Represent Tree Structures

Consider the following facts about Brown:

- brown.sex!male;
- brown.class!upper;
- brown.in!dining_room;
- brown.relationship.lucy.evaluation.attractive!40;
- brown.relationship.lucy.evaluation.humour!20.

This is a declarative representation of a tree structure, implemented as a trie [9]. A group of shared literals has a shared prefix (in this case, "brown"), and the subtree can be referred to directly by its prefix. The subtree can be removed in one fell swoop by deleting its associated prefix. For example, we can remove all of the facts about Brown by deleting the term "brown."[9] A prefix (referring to a subtree) is the Praxis equivalent of an object in an object-oriented programming language.

The structure of literals allows us to express the lifetime of data in a natural way. If we wish a piece of data d to exist for just the lifetime of an object t, then we make the prefix of t be the prefix of d. For example, if we want the beliefs of an agent to exist just as long as the agent, then we place the beliefs inside the agent:

```
mr_collins.beliefs.clergymen_should_marry
```

Or if we want the state of a game to exist just as long as the game itself, then we place the state inside the social practice for the game:

```
process.whist.data.whose_move!brown
```

[8]The database (representing the current world state) is just a collection of ground literals. But the queries that can be expressed in the Praxis language are all propositions of type P, using the familiar connectives

$$P ::= X \mid P \wedge P \mid P \vee P \mid \neg P \mid P \rightarrow P \mid (\forall v)P \mid (\exists v)P.$$

[9]Compare this with prolog, where it is much harder to remove all sentences containing a particular symbol. You can remove all predicates of a *certain arity*, but you would have to separately remove the various different groups of predicates of different arities.

A related advantage of exclusion logic is that we get a form of automatic currying [32] which simplifies queries. If, for example, Brown is married to Elizabeth, then we might have "brown.married.elizabeth" in the database. In exclusion logic, if we want to find out whether Brown is married, we can query the subterm directly; we just ask if "brown.married." In traditional predicate logic, if married is a two-place predicate, then we need to fill in the extra argument place with a free variable. We would need to find out if there exists an X such that "married(brown, X)." This is more inefficient as well as being more verbose.

B. Automatic Removal of Invalid Data

The exclusion operator supports the automatic cleanup of data which is no longer referenced.[10] For example, a social practice p might have two states a and b. State a might have two pieces of information $x1$ and $x2$. State b might have three pieces of information $y1$, $y2$, and $y3$. Being in state a would be represented as

$$p!a.x1 \wedge p!a.x2.$$

State b would be represented as

$$p!b.y1 \wedge p!b.y2 \wedge p!b.y3.$$

Now, if we are in state a (because the statement $p!a$ is true), and we switch to state b (by inserting $p!b$ into the database), all the local data from state a are automatically removed from the database according to the update rules for exclusion logic.

C. Simple Postconditions

When expressing the preconditions and postconditions of an action, STRIPS[11] has to explicitly describe the propositions that are removed when an action is performed:

```
action move(A, X, Y)
  preconditions
    at(A, X)
  postconditions
    add at(A, Y)
    remove at(A, X).
```

STRIPS finesses the frame problem [12] by using the closed-world assumption: anything that is not explicitly specified as changing is assumed to stay the same. But there is still residual awkwardness here: we need to explicitly state that when A moves from X to Y, A is no longer at X. It might seem obvious to us that if A is now at Y, he is no longer at X; but we need to explicitly tell the system this. As Orkin writes [19]:

> "It may seem strange that we have to delete one assignment and then add another, rather than simply changing the value. STRIPS needs to do this, because there is nothing in formal logic to constrain a variable to only one value."

Exclusion logic is a formal logic designed to express directly the natural idea that certain variables can only have one value.

[10]This is a form of simplified belief revision, or garbage collection.

[11]These comments apply equally to the planning domain definition language (PDDL).

The interpretation of the exclusion operator means we do not need to specify the facts that are no longer true:

```
action move(A, X, Y)
  preconditions
    A.at!X
  postconditions
    add A.at!Y
```

When using exclusion logic, the postconditions are shorter and less error prone. The "!" operator makes it clear that something can only be at one place at a time.

D. The Exclusion Operator Helps the Author Specify Her Intent

The semantics of the exclusion operator remove various error-prone bookkeeping tasks from the implementer. But perhaps the exclusion operator's main benefit is that it allows the Praxis scriptwriter to specify her intent more precisely. When we specify that, for example[12]:

```
A(agent).sex!G(gender).
```

We are saying that an agent has exactly one gender. This exclusion information is available to the type checker, which will rule out any piece of code which suggests that an agent can have multiple genders.

Some modern logic-programming languages are starting to add the ability to specify uniqueness properties of predicates [31], but they treat uniqueness properties as metalinguistic predicates. Praxis is the first language to treat exclusion as a first-class element of the language.

E. Support Tools for Praxis

A domain-specific language needs a number of support tools before it becomes really usable for production work. Praxis comes with a number of tools[13] to make the scriptwriter's life easier.

- A type-checking system to find errors early. Praxis is strongly, but implicitly typed[14]: the author does not have to specify the types of all variables; instead, the system will infer the types of all variables and complain if a consistent assignment cannot be found.
- An inspector for giving the author runtime access to the precise state of the simulation.
- A playback facility. The game stores the exact set of actions chosen by the player during a game, and writes them to a file. This allows the author to reproduce exactly a previous playthrough. This playback feature required that we kept the simulator fully deterministic at all times.
- A stress-test tool which runs hundreds of instances of the game simultaneously, with all characters controlled by the computer. By running multiple instances of the game at high speed, we are able to find bugs and anomalies quickly.

[12]This is a typing judgement in Praxis, saying that a variable A (of type `agent`) has a unique sex G (of type `gender`).

[13]When designing the inspection tools, we were heavily indebted to the sophisticated authoring tools in Inform 7.

[14]Compare ML and Haskell.

VIII. Social Practices

A. Regulative Versus Constitutive Views of Social Practice

At the heart of the technical architecture is a commitment to a constitutive view of social practices. We will explain what this means by contrasting it with the alternative regulative view of social practices.

1) The Regulative View of Social Practice: Imagine a group of agents, each with his own set of goals and available actions. At any decision point, an agent chooses the action which best satisfies his goals (or expected goal satisfaction, once we take probabilities into account).

In this individualist picture, a social practice is just a set of restrictions on available actions which allows us to collectively increase our utility. For example, the driving-on-the-left practice restricts our actions (we can no longer drive on the right). We are prepared to accept this limitation on our freedom because it lowers the probability of a collision.

2) The Constitutive View of Social Practice: According to the regulative picture, agents could act before they participated in practices. They already had goals, and already understood what actions were available to them. These were already given. All the social practices do is allow us to solve various coordination problems.

The constitutive view (first articulated explicitly by Rawls [20])[15] rejects this assumption. According to the constitutive view, the action is only available within the practice. A nice example from Rawls' original paper is the game of *Chess*: you can move a carved piece of ivory from one square to another, but you can only move your pawn to King 4 if you are playing *Chess*. Again, we can utter a series of noises which sounds like "Ay doo," but this only constitutes a marriage vow in the context of a wedding ceremony. The action is only available in the practice.

One way to see the need for a constitutive view of practice is to consider the vast array of actions we could possibly do now. Sitting here right now, we could lend the stranger on our left 10 pounds; we could tell the other stranger on our right that Paris is the capital of France; we could ring up our spouse and enumerate the prime numbers.

With an infinite number of actions available to us, why are we not overwhelmed with choice? How do we ever find the time to make a decision?

In the constitutive view of practice, the affordances are always embedded in the practices. The agent does not see the action as available unless he is already participating in the practice which makes it visible. The agent is not overwhelmed by an infinite number of choices because he only sees the affordances that are provided by the social practices he is in. It is this constitutive model of social practices which lies at the heart of our simulation. In our implementation, we take this idea literally: every affordance is contained within a practice and is only available if that practice is instantiated.

[15]But this view has a long history and can certainly be traced back to Wittgenstein [29] and Heidegger [3], and arguably to Hegel [2] and beyond.

B. *Role Agnosticism*

Tomasello [33] has argued that one of the critical differences between the great apes and humans is that, although both apes and humans can participate in practices, only humans are able to effortlessly switch roles. As soon as a human infant (as young as two) participates in a practice, she adopts a bird's eye view of the practice, which means she is able, after participating once, to suggest we play again, with roles reversed. Other creatures do not exhibit the ability to play a practice from multiple roles. A great ape can participate in many sorts of practice, but he is locked in to the role which he played. He cannot see the practice from a bird's eye perspective. This is a distinguishing feature of humans.

One of our big goals in Versu was that each practice would be role agnostic: it would be authored without knowing which roles were being played by NPCs and which by human players.

C. *Representing Social Practices*

A social practice is a hierarchical collection of affordances, providing various options to its participants (who are characterized solely in terms of the roles they are playing).

A social practice is represented with the keyword `process`. Processes are specified with declarations, for example:

```
process.greet.X(agent).Y(agent)
  action "Greet"
    preconditions
      // They must be co-located
      X.in!L and Y.in!L
    postconditions
      text "[X] says 'Hi' to [Y obj]"
end
```

Here the term associated with the process is:

```
process.greet.X.Y.
```

The processes can then be instantiated any number of times by adding sentences to the knowledge base. For example, if we add the following assertion:

```
process.greet.jack.jill
```

then the process will be active with substitutions [jack/X, jill/Y]. If, furthermore, Jack and Jill are in the same location:

```
jack.in!hill
jill.in!hill.
```

Now the preconditions of the action are satisfied and the `greet` action will be available to Jack on Jill. If we added another term:

```
process.greet.jill.jack
```

then another instance of the process will be active with substitutions [jill/X, jack/Y], and the `greet` action will be available to Jill on Jack.

The language for describing actions has a number of the features described in PDDL [14]:

- disjunctive preconditions;
- negation in preconditions (using negation as failure);
- quantified preconditions (both universal and existential quantifiers can be nested arbitrarily);
- expression evaluation (the ability to perform numeric calculations);
- domain axioms (the ability to define new predicates and relations in terms of existing relations);
- conditional effects in postconditions.

D. *Respecting the Normative*

Many of the practices we authored had their own individual sense of the normative. During a conversation, you should respond when spoken to, you should respect the salient topics, etc. [25], [30]. During a meal, you should be polite about the food, etc. But the player still has a choice; she can always violate the norms if she wants to. The major requirements for modeling the normative were that:

- NPCs understand what they should and should not do;
- NPCs, if left to their own devices, should respect the norms (unless they have some particularly acute personality deviation which overrides their urge to respect the social mores);
- but the player should be free to violate these norms at any time;
- if the player violates a norm, the others should notice and respond accordingly.

To get NPCs to respect norms, we add postconditions to norm-violating actions to mark that a norm has been violated. We similarly add postconditions to actions which should be performed to mark that a requirement has been respected. We give most[16] agents strong desires to respect the social norms. When deciding what to do, the agents will see the consequences of their actions. If they see a norm-violation consequence, that will be a major disincentive.

Although NPCs typically do not violate norms, the player is allowed to do whatever she likes. If the player does step outside the bounds of propriety, the NPCs should notice and respond accordingly. Getting drunk, insulting the wine, refusing to answer when spoken to, all these norm violations are only fun to play if they are noticed. When a norm violation occurs, the practice which kept track of the norm spawns a subpractice whose job is to mark that a violation has occurred. This responsive practice will provide options to the others: disapproving, forgiving, getting angry, and even (in extreme circumstances) evicting the character altogether.

IX. Agents

A. *Autonomy*

When an agent is deciding between various possible actions, he looks at the consequences of each action, and chooses the action which best satisfies his desires. He uses forward chaining, rather than goal-directed backward chaining.[17]

[16]The DM will occasionally lower these desires for certain agents (e.g., our rakish poet, Brown) when it wants them to behave outlandishly for dramatic purposes.

[17]SHOP [17] uses a similar approach: "Since SHOP always knows the complete world-state at each step of the planning process, it can use considerably more expressivity in its domain representation than most planners."

This is a form of utility-based reactive action–selection [11], [23], rather than a full-blown planner. But it is an unusual utility-based method in that it is highly responsive to the fine details of the simulation.

In most systems which use utility-based decision making, the agent's estimation of the consequences of the action is much simpler than the actual consequences. For example, in *The Sims*, when a Sim decides to go to the toilet, the actual consequences of the action are:

- routing into the bathroom;
- if there is someone already in the room, he expresses frustration, and exits;
- otherwise, he locks the door, sits down, and relieves his bladder motive.

These are the actual consequences. But the estimated consequences are much simpler. When considering going to the toilet, all he sees about the future is:

- he will relieve his bladder motive.

This discrepancy (between the actual and estimated consequences) creates all sorts of issues. One particularly aggravating problem is that the estimated consequences miss all the conditional effects: effects which may or may not happen depending on various other aspects of the simulation state. For example, whether the going to the toilet will be successful depends on whether there is somebody already in the bathroom. But the Sim does not consider this when planning: he thinks the action will always be successful. If there is, in fact, someone else in the bathroom, the Sim will attempt to use the toilet, but he will be thwarted by the other person. Then, he will try to choose another action; his bladder motive will still be unsatisfied, and he will attempt to use the toilet again, etc. This behavior can repeat indefinitely.[18] Now there are various kludges we can put in to avoid this particular problem. But the best way to fix this general class of problems is to address the root cause: instead of using a simplified estimation of the consequences, compute the actual consequences for decision making.

The NPCs in Versu look at the actual consequences of an action when deciding what to do. When considering an action, they actually execute the results of the action, rather than some crude approximation. Then, they evaluate this future world state with respect to their desires. Then, we undo the consequences of the world state.[19]

This sort of decision making is broad rather than deep. It does not look at the long-term consequences of an action, but at all the short-term consequences. By looking at a broad range of features, it is able to make decisions which would typically only be available to long-term planners. For example, the NPCs are able to play a strong game of whist. When considering the various cards the whist player can play, the decision maker computes the various features of a move (whether it counts as winning the trick, whether it counts as throwing away a card, trumping, getting rid of a suit, etc.). These conditional effects determine the score of playing the card. Using such simple appraisals, and giving the NPCs suitably weighted desires to perform actions which satisfy these appraisals, is all that is needed for the NPCs to automatically play a strong game. There is no need for any separate sui-generis whist-playing decision procedure.

Our short-term planner elegantly handles large dynamic sets of goals, allowing characters to select actions that advance multiple goals simultaneously.

Utility is computed by summing the satisfied desires. A want is a desire to make a sentence true, and that sentence can be any sentence of exclusion logic, to any level of complexity. So, for example, Brown (our rakish poet) likes annoying upper class men. He wants it to be the case that there exists an `other` such that:

```
Other.sex!male and
Other.class!upper and
is_displeased_with.Other.brown.
```

Each want comes with an associated utility modifier. This want is tagged with a utility modifier of 20. Every separate instantiation of this desire gives an additional 20 score. If, for example, Brown can see that one remark would simultaneously annoy three upper class men, then that remark would score three times higher than a remark that just annoyed one.

B. Individuality

When attempting to model individual personalities, one common method is to implement a small finite set of personality traits, and define a character as a combination of these orthogonal traits [6]. We wanted a more expressive system, in which there were an infinite number of personalities—as many personalities as there are sentences in a language.

The fact that the planner looks at the entire simulation state means that the range of things that agents can desire is very wide. Some examples of individual desires are:

- The doctor is sexist: he wants $\exists X\ leader!X \wedge X.sex!male$;
- Brown enjoys annoying upper class men: he wants $\exists X\ \ is\_displeased\_with.X.brown \wedge X.sex!male \wedge X.class!upper$;
- Peter does not like to be alone: he hates it when $\exists L\ peter.in!L \wedge \forall X\ X.character \wedge X \neq peter \rightarrow notX.in!L$;

The expressive range of the logic is what allows us to specify such fine-grained personalities.

C. Relationships

We use role evaluation (based on Sacks' membership categorization devices [24]) to model many different sorts of relationships described.

At any moment, we are simultaneously participating in many different practices. In *Pride and Prejudice*, Darcy, for example, is simultaneously a member of the gentry, a friend of Mr. Bingley, a potential suitor. For each role he is playing, one crucial question is how well he is playing that role. Different people

[18]The Sims is not an isolated example. FEAR contains exactly the same discrepancy between the planner's understanding of the action effects and the actual game-play effects. In [19], notice the two separate fields in the action class: m_effects for the planner's understanding of the effects, and the ActivateAction() function for the actual game-play effects.

[19]It is only because we are able to undo actions that this approach is workable. The language instruction primitives of the Praxis language were designed to be efficiently undoable. If we could not undo an action, the only way we could compute the actual consequences would be by making a copy of the entire simulation state, performing the action in that copy, and then throwing it away. But copying the entire simulation state is prohibitively expensive when we are considering so many actions for so many agents.

have different evaluations of how well someone is playing a role.

This concept, role evaluation, is at the heart of the relationship model. Just as agents can be participating in multiple social practices concurrently, just so an agent can have multiple concurrent views about another agent.

Some of the role evaluations we use for our Jane Austen episodes include:

- how well bred someone is;
- how properly he is behaving;
- how attractive someone is (evaluating someone as a potential romantic partner);
- how generous, intelligent, amusing, etc., he is;
- how well they are performing their familial role (father, mother, daughter, etc.);
- whether they are a good husband or wife.

In *Pride and Prejudice*, characters evaluate each other according to their success at these various roles:

- the Bingley sisters find Elizabeth lacking in style, taste, beauty;
- they judge Jane to have low family connections;
- Mrs. Bennet judges Charlotte to be plain.

Characters remember the reason for these evaluations[20]:

- Mr. Bingley is a good suitor because he was so affable at the dance;
- Mr. Darcy is a bad suitor because he was rather rude at the dance;
- Jane is a bad catch because her family is so badly connected.

Characters make these evaluations public when prompted (or unpromptedly):

- Mrs. Bennet says Darcy is a bad match for Elizabeth on account of his rudeness;
- the Bingley sisters say Jane is a bad match for Mr. Bingley on account of her low connections.

These public evaluations can be communicated from one character to another:

- Mr. Wickham tells Elizabeth that Darcy is dishonorable and she believes him.

Sometimes, people disagree about their evaluations of a character:

- Elizabeth and Jane disagree about whether Mr. Wickham's dark hints mean that Darcy is blameworthy.

These evaluations affect the characters' autonomous behavior:

- characters will be invited over if they are evaluated sufficiently highly;
- characters will propose if they evaluate the other as a suitable match;
- characters will display gratitude for doing favors (e.g., the wife and daughters are grateful to Mr. Bennet for paying Mr. Bingley a visit).

These evaluations also affect the tone of their autonomous behavior:

- Miss Bennet is not gracious when Mrs. Bennet apologizes because she is not valued highly;
- their effusiveness in looking after somebody depends on their evaluation (e.g., the Bingley sisters are less effusive in their concern for Jane because they do not rate her highly).

1) Updating Role Evaluations: Characters can acquire evaluations in three ways:

- they can start with the evaluation hard coded into them;
- they can acquire the evaluation when they interpret one of the other's actions;
- they can hear someone else's evaluation and decide to believe it.

Characters acquire new evaluations of others when they see them performing actions, and interpret those actions in a particular light. These interpretations are themselves actions which the characters decide to do: they have a choice how to interpret others' actions.

2) Relationship States: Relationship evaluations are multiple and asymmetric: x may judge y according to multiple different roles, and x's views on y may not be the same as y's views on x.

But as well as these multiple views, we also model a single symmetric notion: the public relationship state between the characters. This is the official long-term stance between the characters: whether they are friends, lovers, siblings, or enemies.

D. Reactions

In *The Sims*, there is a curious design asymmetry. The player chooses how to act, but not how to react:

- the player chooses which action her Sim performs;
- but when another character does something to the player's Sim, the player does not get to choose how to react; her Sim reacts automatically.

In Versu, we wanted to restore the symmetry, allowing the player to choose how to react as well as how to act.

For example, when Lucy recounts her art lessons from an attractive teacher, there are many ways of interpreting her remark. A character who admires Lucy may jealously resent the implication that she was attracted to someone else. One attuned to issues of status may realize that Lucy has been sent to an expensive private seminary, and respond by congratulating her on her superior education—or irritably tell her not to brag about herself. Another might simply proceed with a conversation about schooling, art, or romance, riffing on the topics brought up by Lucy's remark.

We choose how to perceive the world, and we are responsible for how we perceive it. One of our guiding design goals was that these interpretations should themselves be interpretable by subsequent interpretations. So if we decide that Lucy was being improper to talk about her art teacher in that way, our decision to perceive her action that way is itself evaluable by others. They may, for instance, decide that I am being prudish. And this interpretation of my interpretation is itself evaluable, and so on.[21]

Reactivity is implemented as a type of social practice. When an action is performed, a social practice is spawned. This reacting practice proposes various different sorts of responses to

[20]The evaluations are stored inside the agent in terms of the form *Agent.relationship.Evaluated.role.Role!Value!Explanation*.

[21]This is something that Sacks [24] has emphasized.

the initiating action. When a response is performed, this response is just another action, which can itself spawn subsequent reactive practices, and so on. Each of these responsive practices is short-lived and destroys itself if anything more recent happens, to prevent agents reacting to "old news."

E. Emotions

We used a fine-grained set of emotional states, based on Ekman's work [4]. An advantage of a text representation was that it was easy to express the various emotional states. It would be much harder to express the fine-grained distinction between, for example, being embarrassed and humiliated if we had to show it in a 3-D face. In order for an emotional state to be intelligible to another, the character should be able to explain it. To do this, the character remembers who the emotion is directed toward (e.g., I am annoyed with Brown), and the event which prompted the emotional change (e.g., Brown's insult).

An agent changes emotional state when performing an action. Reactions, in particular, are a rich source of emotional state changes.

Our representation of agents' emotional states is simple and straightforward: the agent has only one emotional state at a time, and any new emotional state always overrides the previous one. The agent also remembers his previous emotional state, so that we can have autonomous decisions based on mood switching. An agent, for example, may not like to laugh when he is already in a bad mood.

F. Beliefs

The world state is shared among the agents. We do not, for memory reasons, give each agent his own separate representation of the world. Instead, we give them all access to the one authoritative world model. This means, of course, that misunderstandings, etc., cannot be implemented fully.

For specific cases where we want false beliefs, or factual disagreements, we store individual beliefs for that specific issue. So, for example, early on in the ghost story, when there are various spooky (but inconclusive) happenings, the characters can disagree about whether these events are caused by a ghost or there is some more scientific explanation. Another example is that, typically, interpersonal evaluations are shared and accessible to all. But if we want some people to know—and others to be unaware—that Frank loves Lucy, then we store this as a belief, and gate certain actions on whether the actor believes that X loves Y, rather than on whether it is true that X loves Y.

G. Character Arcs

The most complex part of a character description file is the character arc: a story arc for that individual character, describing how his objectives and emotions change over time. This arc references only facts about that individual character, so it can be brought into any story in which the character is placed. Our philandering poet, Brown, for example, might choose to take another mistress, but once he has seduced her, we might want him to start to feel bored and trapped once again. If a social practice provides choices about external low-level actions, the character arc represents internal high-level choices: Does Brown want to take another mistress, or focus on improving his poetry? The author designs the character arc to give the character choices about what he wants to be, not just what he wants to do.

The character arc also specifies a variety of possible epilogs for the character. Each epilog describes the end state of the game from the perspective of that character's defining life choices. If Brown decides to distract himself from his malaise by taking another mistress, this could end up with him ditching her, or with him deciding—to his own great surprise—that he will remain committed to her after all. The character arc has its own sense of drama, separate from the story: it is here that the character may achieve self-awareness, sink into despair, or transform himself.

True character transformation comes about in the moments where a character decides to set aside an old want, adopt a new one, or act in contravention of his own desires. This ability—the ability to choose an affordance that is not what the character simulation would prefer—is not available to NPCs; it is only available to the player. The natural result of this is that the characters controlled by humans have opportunities for change and development, while the characters controlled by NPCs will not exercise those opportunities; instead, they will continue to play supporting roles.

The mixing of character arcs with episode story management also produces a productive interference when it comes to the meaning of the stories as they are experienced and interpreted by the player. For instance, the story manager of the murder mystery might dictate that the characters can identify and confront the murderer and then choose either to turn him in to the law or to help conceal his guilt. These might be the only possibilities recognized by the episode structure. However, different motivations might manifest themselves as a result of the character arcs. For instance, if the character who discovers the murderer's guilt is betrothed to the murderer's son, this presents a motivational question to the player: Should she push to convict out of a love of justice, knowing that this will swamp the family in scandal, lead to the end of her engagement, and cause a disappointing end to her personal story arc?

Such dilemmas as these are not explicitly modeled by Versu; rather, they fall out of the creative interference between characters who have personal motivations and desires for the outcome, and narrative cruxes that force dramatic change.

X. The Core Model of Beliefs, Emotions, Relationships, and Evaluations

A. Interpractice Communication via the Core Model

Social practices typically track internally one or several variables pertinent to that practice: for instance, a dinner practice might keep track of how many guests had been served wine, or which course it was; a whist practice might track which agents were playing in the game, what cards they had been dealt, and which suit was trumps.

This kind of internal information, however, does not allow the different practices to communicate with and affect one another. Instead, the agent information described above—consisting of relationship states, beliefs, emotions, and evaluations—serves to communicate between practices.

To be effective, that is, to cause repercussions for other characters and changes within the story, a character's actions within Versu need to change one or more of these core model elements.

Changes to emotional states might last for a few turns and might affect the way the character is described doing things (and her appearance on screen), but a character will experience many emotions in the course of a game. Unless a given mood affects one of the longer term decisions, its unlikely to determine how the story ends.

By contrast, character evaluations of others play into relationship state decisions. A character who evaluates her flirt as cruel or bumbling will have the opportunity to begin a "breaking up" practice, which will end the romantic relationship state between these characters and produce a negative "rejected flirt" relationship state instead. It does not matter to the core model how the character's flirt made himself unacceptable: it might be that he clumsily spilled his wine at dinner, or stepped on her foot during a dance, or said something inconsiderate to her mother. It might be that he committed one large transgression or a series of smaller ones. Regardless of the originating practice, when the character's evaluation of him becomes too negative, the opportunity to break up will present itself.

Relationship states and character self-evaluations are the most significant and lasting part of the core model.

During play, a relationship state may function to make particular actions available: for instance, the practice allowing a couple alone together to kiss will become functional only if the pair are in a romantic relationship state.

In addition, each character in the drama begins with certain relationship or self-belief goals. For instance:

- Miss Bates begins wishing to be friends with someone;
- the poet Brown, who has a chip on his shoulder about his illegitimacy, may have a desire to form an inimical relationship with an upper class man;
- Lucy starts out hoping either to find a protector or else to become more confident in herself.

Succeeding or failing to achieve these goals affects story outcomes, as each story concludes by narrating both how the extrinsic episode ended (Was the murderer identified? Was he tried?) and how the character arc went for the player character (Did Elizabeth conclude the story engaged to Mr. Darcy?).

B. *Using the Core Model to Promote Player Agency*

Most of the player's agency in the game, therefore, comes from using the affordances made available by the social practices to affect the core model in some fashion. In order to maximize this sense of agency, we identified the following design goals.

- Player actions should be rewarded either with information about the world state or changes to the world state.
- Changes to short-term qualities should happen frequently.
- Changes to short-term qualities should lead to a chance for the player to back down or persist in an attempt. For instance, if the player's character has evaluated another as somewhat attractive, this should lead to opportunities to either flirt with or ignore the object of affection, allowing the player to indicate whether it is really her intention to try to develop a relationship there.
- Changes to long-term qualities should be strongly marked. If a player becomes someone's friend, flirt, enemy, etc., that event should entail several moves of interaction and be marked out clearly by the user interface. (Currently, making a friend, a flirt, or an enemy is an achievement which is strongly noted when it occurs.) These events are key to the story and should be noted as significant accomplishments.
- Relationship state changes should require active choice from both characters. It should never be possible for an NPC unilaterally to change his or her relationship to the player, though the NPC might offer the player an unavoidable choice, either prong of which will have a significant state-changing outcome (e.g., "marry me or we will break up").

C. *Using the Core Model With Autonomous Agents*

The use of autonomous agents also introduces an additional design goal into the system. As explored above, each agent may have arbitrarily complex preferences. Lucy likes to evaluate other characters positively. Miss Bates likes to be in a good mood. Mr. Quinn likes to have a bad opinion of Frank Quinn. However, these preferences can only produce significantly different behavior between agents if the range of available affordances, and the effect of those affordances, is sufficiently great. Otherwise, agents will be comparatively scoring just a handful of options, and they are unlikely to produce distinct behavior, even if their formulas for scoring differ widely. This produced an additional design goal:

- individual actions should ideally produce change to multiple qualities or types of quality.

This is especially easy to demonstrate with respect to conversation. A given line of conversation may accomplish any of the following tasks:

- shift the speaker to a new belief about the world;
- communicate a belief;
- shift the speaker to a new emotion;
- communicate one or more emotions;
- shift the speaker to a new evaluation of another character;
- communicate an evaluation of another character;
- shift the speaker to a new self-evaluation;
- communicate the self-evaluation of the speaker;
- count as an atomic action to which others may react.

For example, Mr. Collins may speak about how Lady Catherine, his patroness, has complimented the sermons he preached at her parish. This quip[22] is marked to do each of the following things:

- communicate that Mr. Collins has a positive self-evaluation about intelligence, inviting listeners to accept or reject this view of Mr. Collins;
- communicate that Mr. Collins admires Lady Catherine's status and patronage, inviting listeners to accept or reject this view of Lady Catherine;
- shift Mr. Collins to the "pleased" emotional state, because he enjoys dwelling on the compliments he has received.

[22]Dialog is authored in units called quips, which are combinations of a text template together with a collection of possible effects on social and emotional state.

Likewise, Lucy might say, “Oh no! I’m afraid there must be a ghost here!” when she has not previously believed in ghosts. That quip would have the following effects:

- shift to a belief in ghosts, changing Lucy’s mentality on this point for the remainder of the game;
- communicate a belief in ghosts, allowing other characters to accept this belief or rebut it;
- shift to a fearful emotion;
- communicate a fearful emotion, allowing other characters to offer Lucy comfort;

so that it would be possible for other characters to respond by reassuring her, telling her that she is superstitious to believe such a thing, or agreeing that there probably is a ghost.

Coding speech and other social actions in this way allows agent-driven characters to make a more nuanced use of their various wants. It also produces texts in which characters appear to react to both surface and subtext, as in this final example: Mrs. Elton might say, “What a handsome parlour you have; it is almost as graciously appointed as my sister’s!” This quip would accomplish each of the following:

- communicate a moderately positive status view of the person addressed;
- communicate a superior status view of her own family;
- constitute a “be complimentary” action.

Some characters might respond to the “compliment” by thanking Mrs. Elton; others might reply by being self-deprecating about their status or by putting down Mrs. Elton’s family in order to correct her self-evaluation that they disagreed with.

XI. The Story Manager

Unlike some other systems [22], Versu does not have a general-purpose DM which dynamically combines story elements to produce a wide variety of different stories. Instead, each individual episode has its own individual story manager, which encodes the author’s understanding of the narrative goals for that particular episode. A particular story (say, a murder mystery) has certain key recognizable moments (the victim makes himself unpopular, the victim is killed, the body is discovered), and the story manager is responsible for making sure these events happen at the right moment.

The story manager for an episode is a high-level director who does not like to micromanage. Given a stock of characters and a set of social practices, all our story manager likes to do is initiate practices, watch their progress, and insert occasional changes. It leaves the performance of those practices, and the individual decisions, to the individual autonomous NPCs.

The story manager is a reactive process (itself implemented as a social practice). It starts by creating characters and placing them in initial social situations. Once these characters have been given some interesting goals, it often leaves those autonomous characters to their own devices for some time, before the next intervention. The story manager watches what is going on, spawning new social practices, and tweaking individual goals, to keep things moving.

For example, in the murder mystery, the story manager wants, after the meal has finished, for the people to gather together to perform some sort of group activity: reading together, music, whist. But it does not mind which particular activity, so it spawns different practices on different occasions, leading to significantly different runthroughs.

Our story managers are significantly less ambitious than some systems [22]. They do not plan ahead, anticipating the narrative consequences of various dramatic moves, scoring each move according to narratological criteria. Instead, our story managers are reactive processes, handcrafted for each episode. This gives the author strong control over the outcome and quality of the story, at the expense of emergence at the narrative level.

XII. Related Theory: Computational Models of Social Practice

A. *Schank and Abelson’s Scripts*

Schank and Abelson’s work on scripts [27], [28] has been hugely influential. They were one of the first in the AI community to articulate the important idea that an individual action is not intelligible on its own: its intelligibility comes from the social practice in which it is embedded. They used the term script to describe a computer model of a routine social practice: eating at a restaurant, traveling on a bus, visiting a museum. A script is a state graph containing a distinguished path which is marked as “normal.” (For example, in the restaurant script, the normal path involves the customer ordering, eating the meal, and then paying for it.)

The script describes coordination of multiple actors: characters were assigned to roles and the script understood which actions were expected for each role in each state. The script also achieves continuity over time: an individual agent’s sequence of actions over time is intelligible as a sequence of causally linked actions as the script travels through various states.

Schank and Abelson’s theory accommodates the important idea that multiple scripts can be running concurrently, and can interfere with each other. One example they give is:

> “John was eating in a dining car. The train stopped short. John’s soup spilled.”

Here, the eating script and the being-on-a-train script are running concurrently. A problem in the train script then causes an interference in the eating script.

Schank and Abelson’s work on scripts was a major source of inspiration to us. But our model of social practice is different in a number of ways. First, Schank and Abelson used scripts as a way of understanding natural language stories, while we use social practices as a way of guiding autonomous behavior in an interactive system. Second, Schank and Abelson use so-called “scruffy” methods to model social practices (conceptual dependency theory is a graph-based representation without a formal semantic), while we model the entire simulation state declaratively using a modal logic. (In Versu, we tackle “scruffy” research problems with “neat” methods.) Third, a Schankian script describes a social practice from a particular perspective: from the viewpoint of one distinguished role. Schank and Abelson are explicit about this [27, p. 210]:

> “A script must be written from one particular role’s point of view. A customer sees a restaurant one way; a cook sees it another way.”

Again, [28 p. 152]:

"A script takes the point of view of one of these players, and it often changes when it is viewed from another player's point of view."

In Versu, by contrast, a social practice is authored from a bird's eye perspective: the domain-specific language supports an authoring style in which practices are agnostic about which particular character is playing which role. In Versu, a restaurant script is written once and incorporates both the customer's and the cook's perspectives.

B. Moses and Tenenholtz's Work on Normative Systems

Moses and Tenenholtz [16] have also developed a computational model of social practices. They define a normative system as a set of restrictions on available actions such that, when these restrictions are respected, there is a guarantee that a certain desirable condition obtains. For example, on a busy road, the desirable condition might be that no cars hit each other, and the restriction might be that all cars drive on the right-hand side of the road. Part of the power of their work is that, using a type of modal propositional logic, they can prove that certain norms guarantee certain desirable properties.

Their approach is related to ours in that they use formal logic to describe social systems. But there is one fundamental difference: they see social systems as restrictive rather than constitutive. Imagine an agent who already has a set of available actions. A normative system, in their sense, provides a restriction on the set of actions. The restriction on the agent's freedom is offset by the (provable) desirable properties of everyone obeying that restriction: forcing me to drive on the right is a restriction on my ability to drive on the left, but the guarantee that I can drive without collision offsets that restriction. Versu, by contrast, uses a constitutive model of social practice in which social practices make new actions available: placing a card down on the table only counts as trumping with the Jack of Spades within the context of the whist game in which it is embedded.

XIII. Related Interactive Systems

Starting with *Tale Spin* [15], there have been many attempts to generate narrative using a simulationist agent-driven architecture. The recurring problem with these attempts has been that the generated stories lacked narrative coherence. It has proven very hard for an author to achieve any sort of narrative control by fiddling with the parameters of individual agents. When building Versu, we were hoping to find a spot in design space that has the generativity of simulation, while also having a satisfying degree of narrative coherence and author ability.

In a recent paper [21], Riedl and Bulitko provided a clear taxonomy for describing design choices in interactive narrative systems. The following are the two fundamental questions that they considered.

1) Authorial intent. To what extent does the human author's storytelling intent constrain the narrative? How much of the story is decided in advance by the author, and how much is generated by the player and computer during play?
2) Virtual character autonomy. Does each individual character make up his own mind, or is each character merely a puppet controlled by a centralized DM?

In terms of authorial intent, Versu is somewhere in the middle between a manually authored choose-your-own adventure, on the one hand, and an automatically generated emergent narrative, on the other hand. Each episode in Versu comes with its own episode-specific DM: a reactive agent which mostly sits back and watches, occasionally intervening to push the narrative forward. In this respect, Versu is rather like Façade: for each episode, there is a specific situation which has been carefully authored, but the exact path taken through the narrative landscape, the how and the why, is up to the player to determine.[23]

In terms of character autonomy, Versu is strongly simulationist. Each character chooses his next action based on his own individual beliefs and desires. It is very rare indeed for the DM to override the characters' autonomy and force them to do something. Instead, the DM typically operates at a higher level, by creating new social practices, or tweaking the desires of the participants.

Unusually, the DM is also modeled as an autonomous agent, and chooses which (metalevel) action to do based on her own (metalevel) desires. This means that, in some episodes, the player can actually be the DM.

A. Comparisons With Façade

1) Playing the Same Scene From Multiple Perspectives: Whereas Façade chose a middle ground between the story-driven and agent-driven approaches, Versu is much more heavily agent driven and simulationist. At the heart of Versu's simulation of social practices is a distinction between the roles in the social practice and the characters who are assigned to those roles. Because the player can play different roles in a situation, and can assign many different permutations of characters to the roles, a Versu situation has much more variation and replayability than the Façade scenario. If we were to implement a version of the Façade scenario in Versu, the player would not be constrained to playing the guest—she could also play Trip or Grace. Further, in a Versu version of Façade, the player could assign different characters to the roles. The player could assign, say, Mr. Darcy to play the male host, and Elizabeth Bennett to play the female host, and see how differently it plays out.

2) The Difference Between JBs and Social Practices: A story-driven interactive narrative lets a single DM determine what happens next. (This provides continuity at the cost of emergence.) At the opposite extreme, an agent-driven interactive narrative lets each individual agent determine what he will do next. (This provides emergence at the cost of continuity.) One of the most striking architectural ideas in Façade is the use of JBs which coordinate a group of agents and are intermediate between individual agents (on the one hand) and a single DM (on the other hand). A JB can express synchronization between its participants and can enforce continuity between

[23]Versu's DM is rather less ambitious. Façade models a sense of rising tension and chooses the next beat by finding the one which most closely matches the intended current tension. We do not do this.

individual behaviors. But a JB, as Mateas and Stern use it, is more restrictive than a social practice in that:

- a JB is a way of coordinating NPCs—there is no equivalent of a JB for coordinating PCs—or for coordinating a mixture of PCs and NPCs;
- when an NPC is deciding whether to enter a JB, this decision is not based on his individual personality or desires. Deciding whether to join in depends only on whether the NPC can participate (whether he has an individual behavior which matches the specification of the JB), and not on whether he wants to.[24]

A social practice is like a JB in that it is responsible for coordination and continuity. But it is more general in two ways. First, a JB coordinates NPCs only, while a social practice can coordinate both NPCs and PCs together. Second, a social practice does not wrest control from the individual agent, instead it always respects the individual agents' autonomy. It provides suggestions, but leaves it up to the individual agent what to do. In Versu, the social practice is a coordinating entity which is intermediate between individual agents and a DM, but this coordinating entity does not wrest control from the individual agents. In Versu, it is always the individual agents who decide what to do.

What this discussion shows is that the broad question of virtual character autonomy turns out, on closer inspection, to be divided into three separate questions.

- What sort of entity controls decision making?
- What sort of entity provides coordination between agents?
- What sort of entity provides continuity over time between the actions of a single agent?

Façade answers all three questions in the same way: the JB provides coordination, continuity, and is also responsible for making decisions. In Versu, there are separate answers to these questions: the social practice provides coordination and continuity, while the individual agent makes the decisions.

3) The Cost of Content Production: The scenario in Façade takes about 20 min for the player to complete. This episode took three plus years to create. A comparable 20-min length episode in Versu takes about two months to create. Versu also contains longer episodes. The ghost story, which takes 45–60 min to complete, took six months to produce.

In terms of the number of behaviors, Façade has 30 parametrized speech acts created during the three plus years in development. Versu, by contrast, has more than 1000 parameterized actions, authored in one year in development.

We attribute our faster content production time to two main factors. First, using text output rather than 3-D animated characters saved us a lot of production time. Second, the domain-specific language in which we authored behaviors (Praxis) was a very high-level declarative language for creating content. A high-level declarative language can express behavior more compactly than a procedural language.[25]

	Decision Making	Coordination	Continuity
CYOA	DM	DM	DM
The Sims	Agent	Speech Act	-
Prom Week	Agent	Speech Act	-
Façade	DM/JB	JB	JB
Versu	Agent	Practice	Practice

Fig. 4. Different ways of handling three aspects of character autonomy.

B. Comparisons With Prom Week

"Prom Week" [13] is a social simulation of high-school student social life developed by a group at the University of California San Diego (UCSD, La Jolla, CA, USA).[26] At a high level, Prom Week has a lot in common with Versu: they are both aiming to provide interesting individual characters in dramatic situations. But at the ontological level, there are some fundamental differences.

Activity in Prom Week involves a sequence of discrete speech acts. In Versu, individual actions are coordinated by social practices which provide meaning to sequences of actions. For example, a game of whist involves a sequence of actions by multiple participants. These actions make sense as a whole because they are all contributing to the one unifying practice: the game.

A concrete example is as follows: suppose one character suggests to some others that they retire to the drawing room to listen to some music. Now in Prom Week, this request would be an individual speech act: others would accept or reject this proposal, and then the speech act would be over. But in Versu, this suggestion is part of a larger social practice: the group deciding what they should do next. This larger practice involves the group as a whole achieving consensus (or failing to achieve consensus) on what they should do. Others may agree with the proposal to listen to music, or they may suggest an alternative pastime (reading, dancing, whist). Others may take sides, attempt to dominate, or back down. In Versu, the simple request speech act is embedded within a larger practice which gives it intelligibility and provides continuity. In Prom Week, by contrast, behavior is just a sequence of isolated and unrelated speech acts.

C. Situating Versu in Design Space

We conclude this section by relating Versu to other games in terms of the three separable aspects of character autonomy (see Fig. 4).

- What sort of entity controls decision making?
- What sort of entity provides coordination between agents?
- What sort of entity provides continuity over time between the actions of a single agent?

In a choose your own adventure (CYOA), there is one entity (the static preauthored story graph, functioning as a nonreactive

[24]See [7, p. 75]: "If all agents respond with intention to enter messages, this signals that all agents in the team have found appropriate behaviors in their local behavior libraries." The only way in ABL for an individual agent's personal preferences to affect the decision to join in is if that preference is added as an explicit precondition: "Note that the preconditions can also be used to add personality-specific tests as to whether the Woggle feels like playing follow the leader" [7, p. 78]. As Mateas later acknowledges, "A possible objection to pushing coordination into a believable agent language is that coordination should be personality-specific; by providing a generic coordination solution in the language, the language is taking away this degree of authorial control" [7, p. 226].

[25]Façade is authored in ABL, a procedural domain-specific language built on top of Java.

[26]We were advisors on this project, and have had many fruitful discussions with the developers over the years.

DM) which decides what happens next, and provides coordination between agents and continuity between action.

In *The Sims*, the individual agent decides what to do next. Coordination between agents is limited to individual speech acts.[27] In *The Sims*, there is no continuity between actions over time.

Prom Week is similar in that decisions about what to do are always made by the individual agent. Coordination between two agents lasts for the duration of an individual speech act and involves an initiating sentence followed by a response. Again, there is no continuity between actions over time.

Façade uses a DM to decide what happens next. It uses JBs, described above, to achieve coordination between agents, and continuity over time.

Versu uses the social practice to achieve coordination and continuity, but always lets the individual agents decide what to do.

XIV. Limitations and Further Work

A. Evaluation

Versu is out now on the App Store for iPad, and soon for other devices. Although we have not attempted to evaluate the system through controlled experiments, user feedback has been very positive. At the time of writing, we have a 5/5 rating on the U.K. store and a 4/5 rating on the U.S. store.

The press coverage has also been positive. *Polygon* wrote that Versu "looked quite simple at first, but became more extraordinary by the moment." *BookRiot* described Versu as "a remarkable set of storytelling tools." *Rock Paper Shotgun* wrote "The simplicity with which it all appears betrays just how complex a social AI project this really is (. . .) The potential for this within text adventures and interactive fiction seems madly enormous." *New Scientist* wrote that Versu "captures the nuances of social interaction in a way not seen before."

B. Limitations of the Agent Model

When using a utility-based decision maker, tweaking the various desires (there are already over 300 desires in the system) so that the preferred action scores higher is a difficult and time-consuming tuning problem. The inspector helps authors understand autonomy tuning problems, by showing the scores of each action—and the reasons why the action gets the score it does—but even with this tool, it is a difficult and unrewarding problem.

Another limitation with the system is the way agents' beliefs are expressed. To simplify the implementation, all beliefs are represented as sentences involving a two-place predicate and a pair of constants. We can represent Mrs. Quinn's belief that Lucy is compromised by Frank Quinn, or Darcy's belief that the ghost was killed in the study, or Elizabeth Bennett's belief that she should not marry Mr. Collins, but we cannot represent:

- beliefs involving universal quantifiers, e.g., "everyone has become insane";
- beliefs involving existential quantifiers, e.g., "the murderer is one of the guests";
- beliefs about others' beliefs, e.g., "Mr. Quinn believes that Lucy believes that Mrs. Quinn is the murderer."

[27]Performing a speech act spawns an invisible object that lasts only for the duration of the act, and coordinates the animations and responses of the two participants.

C. Limitations of Our Representation of Social Practices

1) Simplifying Assumptions of the Social Model: Our model of social practices simplifies in two ways. First, the agents have a shared understanding of the state of the social practices. It is not possible in this model for two agents to have divergent understandings of the state of the situation (for example, disagreeing about whose move it is in a game of *Chess*). Instead of modeling each individual agent's beliefs about the state of the practice, we just model the practice once, and give agents access to it. Second, even if we did give each agent his own individual model of the practice, so that they could diverge, there would still be a shared understanding of the practice: both agents would agree, for example, that greeting is something you do when you are sufficiently well acquainted. A deeper model of practices would allow individual agents to have their own interpretations of the practice.

2) Multiple Concurrent Practices Complexify Authoring: One of the striking things about the architecture is that it allows multiple practices to exist concurrently. Most of the time, there are many practices running at once, each providing various options to the agents. It is because there are so many practices that the player has such a wide range of options at any time.

But this complexity comes at a price: the fact that there are multiple concurrent practices complicates the tuning and debugging of the scenes. Each action from each practice needs to be scored against all the other actions from all the other concurrent practices.

Allowing multiple concurrent practices generates another, deeper authoring problem. Sometimes something happens that is so obviously important that all the other things that are going on should be forgotten, for the moment. For example, in our murder-mystery episode, when the dead body is discovered, there may be many other things that were going on: there may be a budding flirtation between two of the guests, some of the characters may have become drunk, or violated one of the norms of Regency England, and may be getting told off. But when the body is discovered, the affordances from these other practices should be suppressed. The seriousness of the situation should mean jokes and flirtations are not even considered. We currently use a rather simple mechanism for suppressing the affordances from other practices: a dominating practice is one which, when active, suppresses the affordances from any other practices. But this dominating mechanism is too broad and crude for the case in hand: there are some other practices which should coexist with the discovery of the body, for example, weeping at the loss of a loved one. What we really want (but do not know quite how to implement) is Heidegger's idea of a public mood [3], which opens up a range of possibilities and closes off others. In our example, when the dead body is discovered, this should create a public mood of shock, and this mood should reveal various practices (grieving, examining the body, wondering who could have done such a thing) while obscuring others (drinking, joking, flirtation).

D. User-Generated Content

Once she is up to speed with Praxis, a writer can produce a 20-min episode (with a rich variety of end states and the ability to play from multiple perspectives) in one to two months. Although this is a significant increase in content-production speed over Façade, for example, the speed of content production is still an issue for us.

In order to make content production possible in a reasonable amount of time, we have built, on top of the Praxis language, an authoring tool called Prompter. Graham Nelson (the author of the Inform language for interactive fiction [18]) has assisted in the design and implementation of Prompter, which allows writers rapidly to create scenes and dialog in a format that resembles a play script. The script is marked up with additional text, indicating the emotional and evaluative effects of a given piece of dialog or action. The Prompter software then converts the script into raw Praxis.

Prompter-generated episodes can also include other, pure-Praxis files at need, which makes it possible to coordinate generated data with newly invented social practices, props, and behavior.

The existence of Prompter has significantly sped up our internal writing process, making it possible to create a substantially branching 20-min episode in less than a week, rather than in one to two months, as was formerly the case.

Our next steps include sharing Prompter with beta users and developing it further as a front end for Versu development, and releasing it as part of the eventual SDK. The intention is that skilled and dedicated programmers will be able to add new Praxis modules, but that people primarily interested in a more writerly experience with Versu will be able to use Prompter exclusively and still create compelling new stories.

Acknowledgment

The authors would like to thank T. Barnet-Lamb, C. Gingold, I. Holmes, I. Horswill, M. Mateas, and G. Nelson for feedback and guidance throughout the project.

References

[1] P. Blackburn, M. de Rijke, and Y. Venema, *Modal Logic*. Cambridge, U.K.: Cambridge Univ. Press, 2002.
[2] R. Brandom, *Making It Explicit*. Cambridge, MA, USA: Harvard Univ. Press, 1998.
[3] H. Drefyus, *Being-in-the-World*. Cambridge, MA, USA: MIT Press, 1990.
[4] P. Ekman, "Basic emotions," 1990.
[5] R. Evans, *Introducing Exclusion Logic as a Deontic Logic. Deontic Logic in Computer Science*. New York, NY, USA: Springer-Verlag, 2010.
[6] R. Evans, "Representing personality traits as conditionals," in *Proc. Artif. Intell. Simul. Behav.*, 2008, pp. 64–82.
[7] M. Mateas, "Interactive drama, art, artificial intelligence," Ph.D. dissertation, Schl. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2002.
[8] M. Mateas and A. Stern, "Writing Façade: A case study in procedural authorship," in *Second Person, Role-Playing Story Games Playable Media*. Cambridge, MA, USA: MIT Press, 2007, pp. 183–208.
[9] D. Knuth, *Digital Searching. The Art of Computer Programming Volume 3: Sorting and Searching*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1997.
[10] D. Lewis, "Scorekeeping in a language game," *J. Philosoph. Logic*, p. 379, 1979.
[11] P. Maes, "How to do the right thing," *Connection Sci. J.*, vol. 1, pp. 291–323, 1989.
[12] J. McCarthy and P. Hayes, "Some philosophical problems from the standpoint of artificial intelligence," *Mach. Intell.*, vol. 4, pp. 463–502, 1969.
[13] J. McCoy, M. Mateas, and N. Wardrip-Fruin, "Comme il Faut: A system for simulating social games between autonomous characters," in *Proc. 8th Digit. Art Culture Conf.*, 2009, pp. 1–8.
[14] D. McDermott, "PDDL—The planning domain definition language (version 1.2)," Yale Center for Computational Vision and Control, New Haven, CT, USA, 1998.
[15] J. R. Meehan, "Tale-spin, an interactive program that writes stories," in *Proc. 5th Int. Joint Conf. Artif. Intell.*, 1977, vol. 1, pp. 91–98.
[16] Y. Moses and M. Tenenholtz, "On computational aspects of artificial social systems," in *Proc 11th DAI Workshop*, 1992, pp. 108–131.
[17] D. Nau, Y. Cao, A. Lotem, and H. Muoz-Avila, "SHOP: Simple hierarchical ordered planner," in *Proc. Int. Joint Conf. Artif. Intell.*, 1999, pp. 968–973.
[18] G. Nelson and E. Short, "The Inform 7 Manual," [Online]. Available: http://inform7.com/learn/man/index.html
[19] J. Orkin, "Three States and a Plan: The AI of FEAR," 2006.
[20] J. Rawls, "Two concepts of rules," *Philosoph. Rev.*, vol. LXIV, pp. 3–32, 1955.
[21] M. Riedl and V. Bulitko, "Interactive narrative: An intelligent systems approach," *AI Mag.*, vol. 34, no. 1, 2013.
[22] D. L. Roberts and C. L. Isbell, "A survey and qualitative analysis of recent advances in drama management," *Int. Trans. Syst. Sci. Appl.*, Special Issue on Agent Based Systems for Human Learning, pp. 179–204, 2008.
[23] J. Rosenblatt, "Maximising expected utility for behaviour arbitration," in *Proc. Austral. Joint Conf. Artif. Intell.*, 1996, pp. 96–108.
[24] H. Sacks, *Lectures on Conversation*. Norwell, MA, USA: Kluwer, 1989.
[25] H. Sacks, E. Schlegoff, and G. Jefferson, "A simplest systematics for the organization of turn-taking for conversation," *Language*, vol. 50, pp. 696–735, 1974.
[26] K. Salen and E. Zimmerman, *Rules of Play*. Cambridge, MA, USA: MIT Press, 2003.
[27] R. Schank and R. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry Into Human Knowledge Structures* , ser. Artificial Intelligence. New York, NY, USA: Psychology Press, 1977.
[28] R. Schank and R. Abelson, "Scripts, plans, knowledge," in *Proc. Int. Joint Conf. Artif. Intell.*, 1975, vol. 1, pp. 151–157.
[29] T. R. Schatzki, *Social Practices: A Wittgensteinian Approach to Human Activity and the Social*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[30] E. Short, "NPC conversation systems," in *IF Theory Reader*. Norman, OK, USA: Transcript, 2011.
[31] Z. Somogyi, F. Henderson, and T. Conway, "The execution algorithm of mercury: An efficient purely declarative logic programming language," *J. Logic Programm.*, vol. 29, pp. 17–64, 1996.
[32] C. Strachey, "Fundamental concepts in programming languages," *Higher-Order Symbolic Comput.*, vol. 13, pp. 11–49, 2000.
[33] M. Tomasello, *Origins of Human Communication*. Cambridge, MA, USA: MIT Press, 2008.
[34] N. Wardrip-Fruin, *Expressive Processing*. Cambridge, MA, USA: MIT Press, 2007.

Richard Evans studied philosophy at Cambridge University, Cambridge, U.K., and artificial intelligence at Edinburgh University, Edinburgh, U.K.

He was the CEO of Little Text People and is now a Senior Architect at Linden Lab, San Francisco, CA, USA. He is the technical architect for Versu. He has been working on multiagent simulations for 20 years. He was the AI Lead Engineer on *Black&White* and *The Sims 3*.

Emily Short studied classics and physics at Swarthmore College, Swarthmore, PA, USA.

She was the Chief Textual Officer at Little Text People and is now a Creative Director at Linden Lab, San Francisco, CA, USA. She is the creative director for Versu. She specializes in interactive narrative, especially dialog models. She is the author of over a dozen works of interactive fiction, including Galatea and Alabaster, which focus on conversation as the main form of interaction, and *Mystery House Possessed*, a commissioned project with dynamically managed narrative. She is also part of the team behind Inform 7, a natural-language programming language for creating interactive fiction.