The Ensemble Engine: Next-Generation Social Physics

Ben Samuel, Aaron A. Reed, Paul Maddaloni, Michael Mateas, Noah Wardrip-Fruin
University of California Santa Cruz, Expressive Intelligence Studio
{bsamuel, aareed, pmaddalo, michaelm, nwf}@soe.ucsc.edu

ABSTRACT
Despite being central to many game stories, dynamic social relationships in video games are difficult to make playable in meaningful ways. To help address this issue, this paper presents the Ensemble Engine (EE), the first publicly available "social physics" engine. The Ensemble Engine is inspired by the lessons learned from more than five years building the *Comme il Faut* (*CiF*) social physics engine, and a number of games employing it (including *Prom Week*). The Ensemble Engine retains the most successful aspects of *CiF,* while also making major improvements in areas such as the flexibility of its action structure and expressivity of its rules. The system is authored in an open-standard language (JavaScript) and includes an authoring tool to increase accessibility for game researchers and creators. Through these improvements and this dissemination strategy, the Ensemble Engine represents an opportunity for the potential of social physics to become much more broadly explored.

Categories and Subject Descriptors
K.8.0 [Personal Computing]: General – Games. I.2.4 [Artificial Intelligence]: Knowledge Representation Formalism and Methods – Representations (procedural and rule-based).

General Terms
Design

Keywords
Game design, social simulation, interactive narrative, authoring tools, javascript.

1. INTRODUCTION
Social relationships are important to the stories of many games. The dramatic twists and turns of social relationships are central to action-adventure titles (e.g., the *Uncharted* [15] games), role-playing games (e.g., the *Final Fantasy* [20] series), and even survival games (e.g. *State of Decay* [24]). Unfortunately, players can almost never directly influence these relationships, which usually change at fixed, predetermined progression points in the game. By contrast, other systems within games (such as combat) do allow the player to have direct influence and agency..

Some games have provided somewhat more dynamic social relationships. For example, some role-playing games allow the player character to move relationship values up and down with non-player characters, gating the possibility of character-specific content such as quests, battles, and romances [1]. But these systems are simple, and interactions with them are limited, such that players are not engaged in the same sense as, for example, they are with same game's combat systems.

The standout example of playable social interactions is *The Sims* [22], which is a hugely successful series, though rarely effectively imitated. But *The Sims* is also quite limited in the potential to have specific realizations of social interactions (e.g., characters cannot use language), the meaningful use of history, the involvement of multiple characters in actions, and so on. Other popular and critically acclaimed games such as Telltale's *The Walking Dead* [26] present the illusion that characters have richer memories and social relationships than they actually possess [9], meaning that these dynamics ultimately never strongly influence the game's plot .

In recent years, a different approach to social dynamics has emerged, one that we call "social physics" [12]. It is influenced by social psychology [7] and some computational representations of social behavior [3, 11]. This idea enables dynamic social interactions to become the centerpiece of a game, and for the events and developments in its social world to be richly represented. The two primary examples of social physics engines so far are the *Comme il Faut* system (abbreviated *CiF*, and used to create *Prom Week* [14], *Mismanor* [21], a game for the *SIREN* conflict resolution project [27], among other playable experiences), and the *Versu* system (used to create a Regency-era comedy of manners, a set of modern office comedy stories, and the ambitious *Blood and Laurels*) [5].

Unfortunately, both of these systems have only been available to their creators and those working directly with them. This has made it impossible for a wider group of game creators to explore the possibilities of social physics. It has also prevented social physics gameplay from being combined with other types of gameplay, a potentially fruitful marriage for genres ranging from Role Playing Gamess to Twine stories (Twine is a tool for creating interactive, nonlinear stories [23]).

In response, we introduce the Ensemble Engine (EE), a freely-available social physics engine[1]. The design of the system is directly informed by our experience of more than five years of building the *CiF* system, and using it to create multiple games. Put simply, a social physics system looks at all of the social factors impacting a character, or group of characters, and determines how the characters might best react to the current social state to suit their desires, as well as directly changing the social state itself (as appropriate). *EE's* most notable features—in

[1] The Ensemble Engine is being actively developed at the time of this writing. The latest version of the engine and its documentation, along with a sample game, tutorials, and authoring environment, can be found at https://games.soe.ucsc.edu/project/ensemble-engine.

addition to being the first available social physics engine—are flexibility, being domain-agnostic, and ease of use.

CiF was originally written in ActionScript3, and developed in conjunction with the game *Prom Week* [13, 14]. *Prom Week* is an example of AI-based game design [4], a paradigm in which an AI architecture and a game using that AI are developed in tandem to expand the expressive range of both systems. *Prom Week*, and by association *CiF*, received recognition as a technical feat: *Prom Week* was selected as a finalist in the 2012 IndieCade festival main competition, and was a finalist in the 2012 Independent Games Festival competition in the category of Technical Excellence. It also garnered praise (and some touchingly personal anecdotes of play sessions) from reviewers in both games and story-oriented contexts [10, 16].

Heartened by the success of CiF, and inspired to overcome some of the system's design challenges, the authors were driven to create the Ensemble Engine: a next-generation social physics engine with a focus on enhanced expressivity and accessibility. The Ensemble Engine boasts a flexible action structure, expressive social rules which govern character behavior, and comes bundled with a powerful authoring interface. This new engine is the next iteration of *CiF*-style social physics, completely re-written from the ground up in JavaScript, currently the dominant language for web-based playable experiences. JavaScript was chosen to allow for rapid prototyping of ideas, flexibility in usage for projects, and its wide coverage of platforms, from PC to mobile.

Besides *Prom Week, CiF* and its metaphors have been successfully used in a variety of projects thus far, and have worked well within their parameters. It has been used in a DARPA funded project for cultural training. The social interactions baked into this simulation employed the affordances given by the social engine, and more information can be found in our AIIDE and AAMAS papers [18, 19]. Additionally, an earlier iteration of the Ensemble Engine was utilized in a non-disclosed, complete game prototype in collaboration with a major game studio, for which we expect to report to the game community within a year.

It is completely domain agnostic, and its social structures can be customized to suit the needs of any social world. We have implemented (and describe below) a new action structure, more expressive rules, and an improved authoring tool.

While our primary motivation for creating the Ensemble Engine is to aid the spread of social physics, a secondary motivation is to make a family of artificial intelligence techniques more broadly available to both independent game creators and game researchers. We are working with game researchers to discover new approaches enabled by the Ensemble Engine, as well as working it into an introductory game design course to help inspire new generations of game designers. The Ensemble Engine approach to rule authoring and action nomination has the potential to make a variety of innovative game projects easier to create, and we look forward to helping explore them as widely as possible.

2. EXPRESSIVE FEATURES

This section will cover several of the features that give the Ensemble Engine its ability to represent complex social worlds

- The Ensemble Engine allows users to easily specify the categories of social state in the world via a *schema*. Once these categories are defined, users can reference them in social rules to govern the considerations of the entities that populate their playable experience.
- Moreover, the rules of the Ensemble Engine are capable of describing and referencing complicated social situations involving any number of characters.
- The Ensemble Engine also introduces a new structure for character actions. These actions take advantage of EE's enhanced rule structure and a character-to-role binding process for increased authoring flexibility.

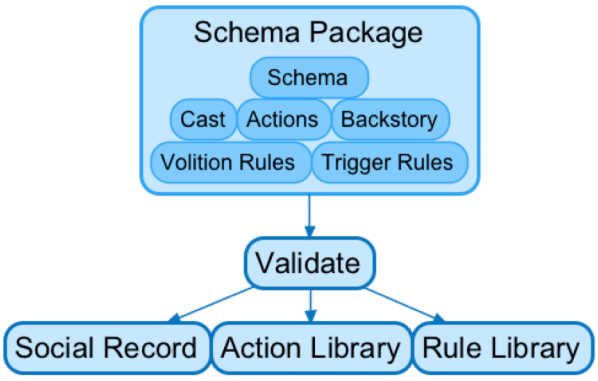

Figure 1. Data flow of schema package components. Social world authors create a schema package. After a validation process, elements of the schema package populate the initial state of the social record, and the action and rule libraries.

2.1 Introduction of Schema

The schema (see Figure 1) is an easy way for users of the Ensemble Engine to define the categories of state in their system. This section begins by describing how state is internally represented in EE. We explain how the data-driven schema system of the Ensemble Engine can be tailored to the specifics of any given playable experience. We then detail the specifics that make up a category defined by a schema and the advantages schemas have over hard-coded alternatives. Finally, we discuss other ways that the Ensemble Engine and its users benefit from a data-driven approach.

2.1.1 Categories in the Ensemble Engine

One of the primary responsibilities of the social engine is to track state. The truths of the diegetic world are stored in a Social Record (SR). Some example SR records from *Prom Week* include character descriptors (e.g., Monica has the trait *arrogant* and the status *popular*), mutual relationships between two characters (e.g., Simon and Zack are *friends*), and non-reciprocal directed attitudes from one character to another (e.g., Gunter has an *affinity* of 87 for Phoebe).

In the original CiF, the categories of traits, statuses, and relationships could each be thought of as a category of social facts, and any given social fact of a category referred to a specific type within that category. Some trait types present in *Prom Week* include arrogant, jealous, brainy, attractive, strong, and clumsy. Some *Prom Week* status types include being popular, feeling sad, and feeling angry. The relationship types in *Prom Week* are friendship, enemies, and dating.

These categories were created in the service of *Prom Week*, though in retrospect there was some undesirable overlap between them. For example, traits were meant to be immutable facets of a

character, whereas statuses represented more ephemeral states or moods, which would eventually fade away unless the source of the status persisted. Statuses were additionally broken into two distinct types of "directed" and "undirected," identical except in whether they referred to a status between two characters (such as being angry at someone) or a single character (such as feeling happy).

The distinctions between these three categories, which evolved during the design of a specific social game were a bit unclearclasses were a bit fuzzy. Though directed statuses, undirected statuses, and traits are three unique concepts, they shared certain properties, such as all referring to Boolean aspects of state. Common properties were noted among other classescategories as well (e.g., an affinity score from one character to another is in many ways a scalar version of a directed status). Realizations such as these helped the authors derive common properties between social facts such as *isBoolean* to distinguish between a fact referencing a boolean (e.g., traits and statuses) or a scalar (e.g., networks such as affinity). Similarly, there seemed to be three primary *directionTypes*: undirected (traits, statuses), directed (networks, directed statuses), and reciprocal (relationships). Once these common properties (and others, see section 2.1.2) were recognized, they became the basis for defining a social schema. In short, the Ensemble Engine*'s* social categories are determined by these defining properties, rather than hard-coded and imposed by the system.

The properties of each category affect how the Ensemble Engine processes that category. The Ensemble Engine does not have a separate evaluation function for each category; it breaks functionality up by properties. For example, all code dealing with Booleans is defined in one place, as is all code dealing with reciprocity.

Thanks to this design choice, it became easier to describe the distinctions of different categories, and to mix and match properties to generate entirely new categories. The larger variety of categories enabled by doing this allows social physics to be applied to a wider variety of game genres. For example, it is now relatively simple to define a new category that describes numeric facts that only apply to an individual character. Individual numeric traits are part of many games, including most role-playing games that might make use of numbers such as these to represent character attributes like strength, agility, and charisma.

EE also uses a data-driven approach to allow for the easy editing of categories. Users can edit a JSON file called a schema, which contains all of the types for their playable experience. The Ensemble Engine then reads it in, and makes use of the contents of this file. The rules written by the user are validated, and it ensures that the types referenced in the rules are specified in the schema file, raising an error if there is a mismatch, and then directing the user to the file with the offending rule.

2.1.2 The Components of Categories in the Ensemble Engine

To give a sense of the range of categories that can be created in the Ensemble Engine, we present the properties that can be defined:

category – A string representing the name of the category.

isBoolean – If true, the category represents a Boolean fact. The category is a scalar if this is set to false.

directionType – Can be "directed" (affects two or more entities, from one to others), "undirected" (applies to only one entity), or "reciprocal" (meaning that the category affects two entities, and will always be the same value between the two entities). If used in the spirit of the original CiF, these entities will most likely take the form of characters in the playable experience. If used outside of a social-based context, these entities could be anything that have a relation to each other, e.g. floors in a procedural dungeon generator, the properties of a texture, or the rules to a game itself. This ability allows the use of non-character elements of a game to act more dynamically and change based on rules provided to EE. (Any future reference to characters in this paper is shorthand for both traditional agents and these more abstract entities.)

types – An array of strings representing the potential typesinstances of the classcategory (e.g., the "status" classcategory might have types *popular*, *sad*, and *happy*).

defaultValue – The initial value for each type applied to all characters at the start of a playable experience, unless otherwise specified. Should be true or false if *isBoolean* is true, or an integer otherwise.

duration – How many discrete time steps the classcategory should remain true, if boolean. If unspecified, assumed to be infinite. Ensemble keeps track of time via these time steps, which are incremented when the client game choses to do so.

minValue/maxValue – The minimum and maximum value types of this category, if numeric.

actionable – A boolean value specifying whether this category is permitted to be part of character intent formation. One of the key components of the Ensemble Engine is calculating volitions (i.e. desires) of characters and determining the actions they want to take, in hopes of adjusting the current social state to fulfill these desires. Any category can potentially be the subject of intent formation if *actionable* is set to true. Though the terminology is steeped in social metaphor, intents are largely a means of categorizing actions, with the volition formation process acting to nominate an action for performance within one of these categories. Though the internal terminology feels best suited for a system with agents forming volitions, the Ensemble Engine works just as well for more abstract entities "taking actions" to change their qualities and their connections to other entities.

2.1.3 Benefits of the Ensemble Engine Category Structure

The new category structure of the Ensemble Engine is very flexible. In addition to being customizable—category names such as "relationships" and "statuses" can be named by the user to terms they find more appropriate—the system has significantly enhanced expressive power through the new combinations of categories now accessible by EE.

Some of these new combinations, such as the aforementioned individual numeric traits, make the Ensemble Engine more compatible with existing game genres and conventions. Through experimentation, some interesting atypical combinations can be discovered. For example, a category with *isBoolean* set to true, a finite duration and a default value of true could represent a condition that will keep recurring unless characters actively work to prevent it from happening, such as in a social world where sickness is the norm and taking medication only temporarily alleviates it.

In short, users of *the Ensemble Engine* have the power to define novel categories that are pertinent to their playable experience.

2.1.4 Other Schemata Components

The schema file is only one part of the new data-driven approach employed by Ensemble. Other files allow Ensemble Engine users to specify information pertaining to the characters (or other entities) of the world (the cast), the starting history of the world (backstory between characters, or any starting state that differs from the default values specified in the schema file), the trigger and volition rules of the world (discussed in more detail below), and the actions that the cast can take in the world (also discussed below).

The data-driven approach also allows for faster iteration, with changes seen as soon as the game is restarted. One of the greatest hurdles while developing *Prom Week* was the slow compilation pipeline; expediting this process should make development with the Ensemble Engine more pleasant.

2.2 Social Rules

In a world driven by a social engine, the cast-held considerations that govern intent formation are defined through rules. The left-hand side, or LHS, of rules is comprised of a collection of predicates, where each predicate asks a question pertaining to a single fact of the current state of the world. If all of the predicates of the LHS of the rule hold true, then the right-hand side (RHS) of the rule is valuated. The two different types of rules have different content in the RHS. *Volition* rules will either add to or detract from a cast member's desire to perform certain types of actions. *Trigger* rules will directly change the state of the world by adding to or removing records from the SR if the LHS of the trigger rule is met. This is the general functionality of rules from the original *CiF*, and this structure has not changed much for EE.

What has changed, however, is the binding process of roles in rules. Any given predicate in a rule involves at most two roles, a *first* and a *second*. If the predicate in question describes a fact from an *undirected* category, then only *first* will be filled in, and the *second* will be blank (because the predicate only pertains to a single person). Otherwise the predicate will involve both a *first* and a *second* character.

It should be noted that even though any given predicate can involve at most two people, rules could involve more than two people through a combination of predicates. Take for example a Social-engine-based representation of the classic adage, "the enemy of my enemy is my friend." One could represent the LHS of this volition rule using three predicates:

1) X and Y are not enemies.
2) X and Z are enemies.
3) Y and Z are enemies.

As we can see, all three of these predicates involve two roles, but the rule as a whole involves three roles, X, Y, and Z. Of these three roles, X and Y both dislike Z while not having any particularly notable disdain for each other. One can imagine the RHS of this rule would increase X's volition to befriend Y.

The Ensemble Engine allows for any number of roles to be defined within a rule. For example, *EE* could verify if a character had been the butt of 4 or more practical jokes, or if that character had made a romantic proposition to 4 or more love interests, while simultaneously retrieving the names of the jokers or the lovers. Moreover, it could place additional constraints on the search for these characters (e.g., find four practical jokers who have the trait regretful, or potential love interests who already have the dating relationship with someone else). These refined rules can cover many nuanced situations that authors might like to affect character behavior and volition formation.

To illustrate characters reasoning about nuanced situations, we will use as example the situation where:

1) X and Y used to be dating.
2) Z and Q used to be dating.
3) X and Q are now dating.
4) Y and Z are now dating.

Here, X and Z (or Y and Q, depending on whose perspective you take) have essentially switched romantic partners. One can imagine that if this scenario were ever to arise in real life, it would affect the way that all parties involved view each other. In *CiF*, authoring this situation would have been less convenient as it would have to be captured in more than a single rule. These additional rules would apply intermediate statuses on the characters, serving as a faux binding process. These statuses could then be reasoned over in even more additional rules. Thus, while not strictly impossible with CiF, EE's rules can scale with more than three roles, making it much easier for authors to write for.

One final note on the improvement of rules: though the authors have been using X, Y, and Z as a shorthand for proper role names in this paper, any string can be used. As long as the string is consistently spelled across the predicates of a rule, the Ensemble Engine will recognize it to be referring to the same role. Although this may seem like a small change, it is a quality of life improvement for content authors, making rules more human-readable during the editing process. Since rule editing occupies much of a social engine author's time, this small improvement can lead to an eased development process through prolonged use.

2.3 Flexible Action Structure

After forming volitions based on the current social state, each character (the initiator) determines how they wish to affect the world by engaging in a social exchange with another character (the responder). Actions are tied to a specific intent, which were discussed in the overview of the schemata. EE has a hierarchy for actions: Intents in the Ensemble Engine may now be followed by one or more actions, each of which in turn can point to additional actions. Actions in the Ensemble Engine are defined in a JSON file with the following components:

Name: A string representing the name of the action.

Conditions: An array of predicates representing hard preconditions that must hold true for this action to be considered. Using a similar binding process to the one described in the previous section, a record of all potential candidates for each role in the conditions is kept. If there is no valid binding, the preconditions do not hold, and the action is deemed impossible for this particular initiator and responder at the given time step.

Influence Rules: Identical in structure to the volition rules discussed above, they take an entity's base volition score (or desire to engage in the action, computed via intent formation), and add or subtract from that score to modify the entity's volition for this particular action. Influence rules tied to actions serve two functions. First, they help determine which action within a single intent a character might be more inclined to perform (e.g., a brash character might prefer to *StartDating* through a *Pick-Up Line*, while a reserved one might opt to simply *Ask Out*). Second, they

help determine the best candidates for any non-initiator and non-responder characters specified in the conditions. That is, the conditions state the required qualities any tertiary characters have, whereas the influence rules describe their preferred characteristics.

leadsTo: An array of action names. If all of the conditions hold for the current action, then the Ensemble Engine will evaluate each of the actions in the *leadsTo* array. The starting score of each of these actions is the ending score of the current action determined by evaluating its influence rules.

isAccept: Actions are categorized as "accepted" or "rejected" when the responder character receives the intent of the social game positively or negatively, respectively. This Boolean value specifies whether this is an action that should be played when the responder accepts (true) or rejects (false). Though this concept was originally implemented in CiF to solve a design problem in *Prom Week*, the authors believe its utility generalizes to other games.

Effects: An array of predicates specifying how carrying out the action should affect the state of the world. Both the initiator and responder can be referenced here, as well as any additional characters bound to roles specified in conditions and influence rules.

This new structure for actions allows for functionality that was difficult to implement in *CiF*. One marked improvement is that this new system allows actions that refer to more than two people. Players of *Prom Week* may recall that some social exchanges did involve three characters, but the third character was chosen in a manner that was largely outside of authors' (and entirely outside of players') control. Now, thanks to influence rules being able to adjust volitions for not only the initiator and responder, but other characters as well, the Ensemble Engine authors can create actions that focus on three or more characters.

One such action that the *Prom Week* authors always wanted to implement but never could without making severe concessions was *Spread Rumors*. As originally envisioned, the player would select an initiator, a responder, and a victim to be the target of the rumors. The list of potential victims would be filtered to only candidates that made sense given their relationship to the initiator and responder. This proved impossible given *CiF's* architecture, and was ultimately implemented in the same way as all other social exchanges in *Prom Week:* players selected an initiator and a responder, and the system selected a third party that simply satisfied hard preconditions. Although this provided a modicum of control for authors, the difference between a boolean restriction and adjusting a weighted volition has serious consequences for an author's ability to fine-tune social behavior. For instance, if the initiator's friend just made a pass at their date, then perhaps the initiator would want to damage the reputation of their so-called friend. This social situation is now possible with EE.

Another exciting possibility of this new structure is hierarchical actions. Authors can define an action structure as simple or complex as they require, and easily divide up specific actions more elaborately than others, if more authorial control is needed, simply by changing a terminal into a nonterminal, which leads to more granulated, sequential actions. This structure also gives authors more information about the path taken to reach a certain terminal. That is to say, users need not only look at the *terminal* action (i.e., the deepest action in the hierarchy where all preconditions are true), but can instead incorporate the entire

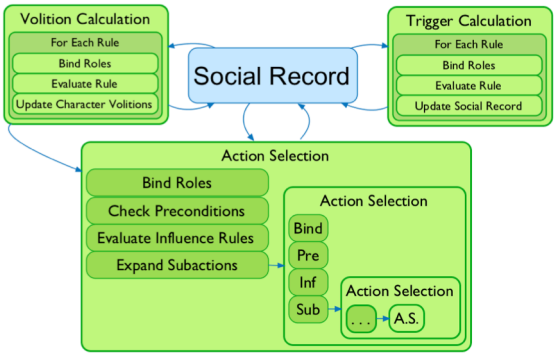

Figure 2. Three major processing elements of the Ensemble Engine.

lineage of actions that led to that terminal. Through the use of this lineage, additional context can be gleaned that might affect the performance of the action, or the effects the action has on the state of the world, or both. Since a single action can have multiple parents (i.e., multiple actions that include the action as a child), there is the potential for myriad variations on any action given the hierarchy chain that led to it.

Currently the native support to leverage the power of hierarchical actions of the Ensemble Engine is limited to providing the chain of actions that led to a terminal action. Taking advantage of the new action structure will make the Ensemble Engine capable of expressing even more varieties of actions, and is exciting future work.

3. NEW AUTHORING TOOL

A new authoring tool has been created alongside the development of the Ensemble Engine incorporating several design insights discovered through the use of the authoring tool created for *Prom Week*.

3.1 Made by Authors, for Authors

The authoring tool for the Ensemble Engine was built from the ground up, designed and implemented by the lead author of *Prom Week*. By having the authoring environment be designed by an expert user, who had written hundreds of instantiations and social rules for the prior version of the system, the Ensemble Engine design tool was able to include many "wish list" features missing from the original tool. Beyond being an expert user, the Ensemble Engine tool designer had prior experience researching and creating authoring tools and visualizations [6], as well as constructing whole works of interactive storytelling [17]; this work also informed the design of the new tool.

As works of interactive storytelling become more dynamic, the demands on the tools used to craft them will grow more complex as well. The need for author-programmer-designers, those that have an intimate understanding of both the requirements of the tool, and the technical skill to construct it with an eye for user experience, is already great and will only increase as more complicated interactive story systems proliferate. However, the necessity of simple interfaces for authoring, so that creating these experiences is not limited to a small subset of all possible authors, also becomes more important. We believe the new authoring tool represents a substantial step up in usability from our original version.

3.2 The Features of the Tool

When the tool is first opened, the user is prompted to select a folder that holds a social schema file and associated data (i.e., a cast of characters, pre-defined history, trigger and volition rules, etc.). This data populates the various features of the tool. The authoring tool is split into four major components: a debug console, an SR History Viewer, a rule viewer for volition and trigger rules, and a rule editor.

3.2.1 The Debug Console

The debug console simulates a traditional command prompt, and recognizes a handful of special commands for querying and changing the state of the rule system. These commands include adding and removing records from the Social Record, seeing the volitions of all of the characters given the current social state, and having characters carry out actions, as well as adding their consequent effects to the SR.

This enables users to quickly set up social situations and verify that the system is behaving as intended, and was inspired directly from the difficulties of authoring in *Prom Week*. This was most apparent when testing specific edge cases of social state involved the time-consuming steps of either recreating the situation through game play, or setting up the perfect social state by editing (and rebuilding) the external library of authored content. Including this debugging functionality in the console encourages a more rapid cycle of discovering problems, and iterating on rule design, as well as fixing bugs in the system itself.

3.2.2 The SR History Viewer

Any social changes, either through console commands or as a result of character actions, are added to the SR History viewer, along with any records specified in the loaded history file. The history viewer shows the user what the social state of the world was at any given point in history, allowing them to observe each time step of the system. At each time step, the social facts added that turn are highlighted in green, so the user can pinpoint the specific moment a change occurred.

As a social world becomes full of trigger rules and complex actions, it can be difficult to keep a holistic view of the system in one's head. The history viewer assists the user in pinpointing the time step when an undesired social change occurred. This assists in the discovery of the causal rule or action.

3.2.3 The Rule Viewer

The rule viewer (see Figure 3) provides an overview of all rules written for the system, showing trigger and volition rules in two separate tabs. Each rule is summarized with its hand-authored name (a simple string meant to describe the essence of what the rule is capturing), as well as a generated natural language description of the predicates composing the rule. This lets the user see a large number of rules at a glance. If the user hovers their mouse over a rule, a tooltip with the original predicate object appears. If a rule is clicked on, the rule will open in the rule editor described below.

Rules can be organized into files according to whatever system the user likes. If there are multiple authors writing rules, each author might have their own rule file. Alternatively, authors could split volition rules up into files based on which intent they refer to, or cluster trigger rules based on the primary change they make to the social state. The rule's origin file is also displayed in the rule viewer.

Lastly, the rule viewer can filter rules based on any word that appears in any part of the rule. This could be the name, any part of any of the predicates, or the file the rule originates from. This is useful both for finding a specific rule, but also to quickly see how many rules have been written for each intent of the system, which can help guide future authoring effort.

3.2.4 The Rule Editor

The rule editor (see Figure 4) allows one to edit existing rules in the system, or create new ones. As previously discussed, rules consist of a descriptive name, a LHS of predicates representing the conditions that must hold for the rule to fire, and a RHS of predicates describing what changes should transpire if the LHS evaluates to true. The editor is designed to make authoring predicates simpler, and to enforce correctness in their structure and content.

A predicate is a flexible construct, based on the properties defined in the user's social schema (see 2.1.2). A directed predicate, for example, needs to define two characters, while an undirected

Figure 3. The Rule Viewer, showing a filterable list of the volition rules authored in the loaded schema package. Clicking a rule will open it in the rule editor.

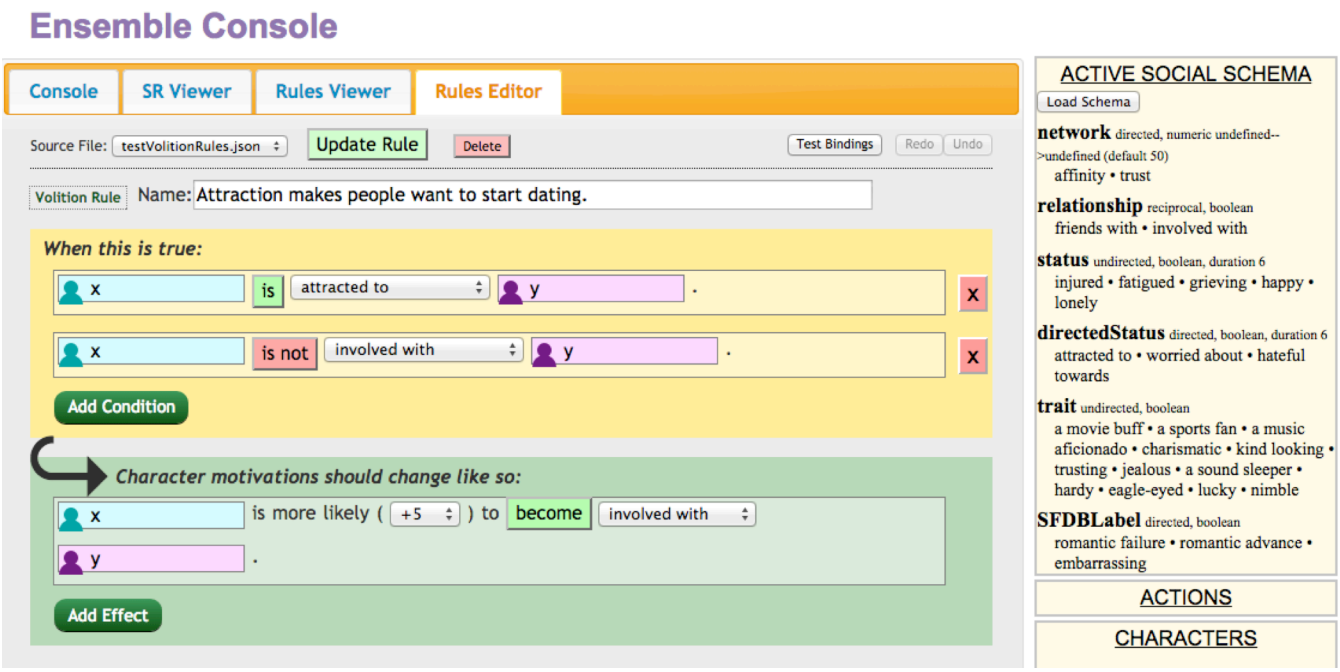

Figure 4. The Rule Editor, showing a dynamically constructed predicate editor for a volition rule.

predicate needs to reference only one. Though the authoring tool for the prior system showed all possible controls for any predicate, this added unnecessary cognitive load to authors, who had to think about which controls were contextually appropriate for any given situation. In the new rule editor, authors specify the category and type for each predicate added to the rule through a drop down menu. Once those are selected, the tool generates an editor for that specific predicate, showing only the currently relevant controls. For example, if the type is Boolean, then the author only needs specify if the predicate has to be true or false. Conversely, if the type is numeric, the author will have to specify the number and an operator (e.g., affinity greater than 40, less than 80, etc.). This ability of the tool to alter its context based on the predicate being authored expedites the authoring process and eliminates one cause of malformed predicates.

The tool also makes it easier for authors to track character roles within a predicate. This paper has discussed how any given predicate can only mention two roles, but a rule can reference as many roles as the author wishes, each role being determined by a unique string of the author's choosing. Authors specify these roles by filling in text boxes. The tool gives a unique color to each role, which provides visual confirmation that roles are assigned correctly; a different color tends to be much easier to catch than a missing letter. Authors can also click the *role* slot to cycle through the roles already established for that rule, further reducing potential typos and reducing typing time.

Lastly, the rule editor has safeguards to protect authors from losing work. Authors can undo and redo changes to correct recently made mistakes—functionality which was much sought-after in the original tool. The tool also saves a backup copy of the volition and trigger rule files every time a rule is edited (up to a user-configurable number of backup files). This provides a history of the rules (useful for archival purposes, or reverting back to a recent change), but also mitigates damage done by unexpected shutdowns of either the tool or computer. These features improve author confidence, and ameliorates some of the effects of using a homebrew design tool, which can include crashes or other unexpected behavior as versions change and the codebase rapidly evolves.

3.3 Debugging Features

The authoring tool also can test the bindings of the rule being edited in regards to the current social state established in the console. When testing bindings, authors select a character for each role specified in the rule, and the tool reports if the rule would currently fire given those role bindings. This addresses the difficulty of easily verifying that abstract rule definitions will behave as expected when placed into particular social situations, a frequent difficulty during *Prom Week*'s authoring. When combined with the SR History Viewer, it gives authors another tool to diagnose why rules that they believe should fire are failing to do so (and vice versa).

Though currently a standalone application, we hope to eventually distribute a browser-based version of the authoring tool that can be easily integrated with any browser-based Ensemble Engine experience. This would allow all the debugging functionality of the authoring tool to operate on the social state generated by a live EE experience. Authors should be able to open up the tool in the same window as their playable experience via a keypress and immediately browse the SR, explore the volitions of their characters, and verify that any given binding of characters to roles is behaving as expected. This planned enhancement should help with debugging immensely.

4. CONCLUSION AND FUTURE WORK

Dynamic and nuanced social relationships are an integral part of many game stories, as evinced by some of the most popular games on the market. Unfortunately, most games fall short in making this vital aspect of these experiences playable, basing social game progress on prewritten branches or binary quest completion rather than exploring procedurally driven social relationships. The authors believe one reason for this is the lack of "off the shelf" social AI systems available for game developers to pick up and

plug in to their games. This paper presents EE, the latest iteration of the social physics engine *CiF*, as a potential candidate for fulfilling this role, being for social AI what Box2d [2], Havok [8], and many others are for Newtonian physics.

EE improves upon its predecessor in a variety of ways, including more expressive rules, more flexible actions, and a user-friendly authoring tool that doubles as an Ensemble Engine debugger. Although the Ensemble Engine has already made marked improvements, there is still more work to be done. While some aspects of a social state (such as rules) can be created entirely within the editor, presently the only way to author elements like actions is by hand-editing JSON files. Ultimately, we hope the authoring tool can be extended to eliminate this awkward step. Once the authoring tool can communicate with playable Ensemble Engine experiences, a new degree of powerful real-time debugging will be made possible. Extending the tool to take advantage of this capability will require hands-on experience designing and iterating new experiences. We also hope to translate the Ensemble Engine into additional languages as well; a C# version is particularly appealing for easy integration with the Unity [25] game engine.

In addition to improving the Ensemble Engine itself, we hope to create a website and community for Ensemble Engine authors. This community would provide a space to showcase work, lend helping hands, and share influence rules and actions with other authors. Much like 3D modelers can currently browse databases of user-created objects, we envision a community where two authors working on "office comedy" stories could discover and share sets of relevant social rules with each other.

In the shorter term, a group of students have already taken part in a*n* Ensemble Engine workshop to learn the basics of the new system, and a fully-fledged Ensemble Engine jam is planned to create a sampling of playable experiences. There are also plans to integrate *EE* into an introductory games studies class, to help influence a new generation of game developers.

The Ensemble Engine is actively being developed at the time of this writing. The latest version of the engine and its documentation, as well as the authoring tool, tutorials, and a sample game, can be found at the following URL: https://games.soe.ucsc.edu/project/ensemble-engine

It is the hope of the authors to see an influx of new digital games with novel mechanics based on social relationships at their core. We believe that the Ensemble Engine is a step towards achieving that goal.

5. ACKNOWLEDGMENTS

This work would not be possible without the hard work that went into the original *Comme il Faut* system initially conceived by Josh McCoy, and further designed with the help of Mike Treanor, both of whom served as consultants for this latest work. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under grant number NSF DGE 1339067. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] Bioware 2012. Mass Effect 3. Bioware.

[2] Catto, E. 2007. Box2d.

[3] Dias, J. and Paiva, A. 2005. Feeling and reasoning: A computational model for emotional characters. *Progress in artificial intelligence*. (2005), 127–140.

[4] Eladhari, M.P.E. Al et al. 2011. AI-Based Game Design : Enabling New Playable Experiences. *Technical Report, UCSC-SOE-11*. 27, (2011), 1–13.

[5] Evans, R. and Short, E. 2014. Versu—A Simulationist Storytelling System. *Transactions on Computational Intelligence and AI in Games*. (2014), 113–130.

[6] Garbe, J. et al. 2014. Author Assistance Visualizations for Ice-Bound , A Combinatorial Narrative. *Foundations of Digital Games 2014* (2014).

[7] Goffman, E. 1959. *The Presentation of Self in Everyday Life*. Anchor.

[8] Havok Inc 2011. Havok Physics. Havok Inc.

[9] Here's a chart of every choice in The Walking Dead: Season 1: 2013. *http://venturebeat.com/2013/03/31/the-walking-dead-season-one-plot-graph/*. Accessed: 2015-02-08.

[10] Impressions: Prom Week: 2012. *http://www.rockpapershotgun.com/2012/02/16/impressions-prom-week/*. Accessed: 2012-02-02.

[11] Marsella, S.C. et al. 2004. PsychSim: Agent-based modeling of social interactions and influence. *Proceedings of the international conference on cognitive modeling* (2004), 243–248.

[12] McCoy, J. et al. 2010. Comme il Faut 2 : A fully realized model for socially-oriented gameplay. *Proceedings of Foundations of Digital Games (FDG 2010) Intelligent Narrative Technologies III Workshop (INT3)* (Monterey, California, 2010).

[13] McCoy, J. et al. 2012. Prom Week. Center for Games and Playable Media.

[14] McCoy, J. et al. 2013. Prom Week : Designing past the game / story dilemma. *Proceedings of Foundations of Digital Games (FDG 2013)* (2013).

[15] Naughty Dog 2011. Uncharted: Drake's Fortune, Uncharted 2: Among Thieves, Uncharted 3: Drakes' Deception. Sony Computer Entertainment.

[16] Prom Week: 2012. *http://alastairstephens.com/prom-night/*. Accessed: 2012-02-17.

[17] Reed, A. et al. 2014. Ice-Bound: Combining Richly-Realized Story With Expressive Gameplay. *Foundations of Digital Games 2014* (2014).

[18] Shapiro, D. et al. 2013. Creating Playable Social Experiences through Whole-body Interaction with Virtual Characters. *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-13)* (Boston, Massachusetts, 2013).

[19] Shapiro, Daniel Tanenbaum, K. et al. 2015. Composing Social Interactions via Social Games. *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems* (Istanbul, Turkey, 2015).

[20] Square Enix 2010. Final Fantasy VII - XIII. Square Enix.

[21] Sullivan, A. et al. 2012. The Design of Mismanor: creating a playable quest-based story game. *Proceedings of the International Conference on the Foundations of Digital Games* (Raleigh, NC, 2012).

[22] "The Sims Studio" 2009. The Sims 3. Electronic Arts.

[23] Twine: 2009. *twinery.org*. Accessed: 2015-04-24.

[24] Undead Labs 2013. State of Decay. Microsoft Studios.

[25] Unity Technologies 2005. Unity3d.

[26] Vanaman, S. et al. 2012. The Walking Dead. Telltale Games.

[27] Yannakakis, G. et al. 2010. Siren: Towards adaptive serious games for teaching conflict resolution. *Proceedings of ECGBL*. (2010), 412–417.