



# Ensemble | Engine for Social Simulation

*Melanie Dickinson*

# Table of Contents

Motivating Ensemble & Social Simulation  
How does Ensemble work?  
Social State  
History  
Character Desires  
Actions  
Trigger Rules

---

# Ensemble

A social AI / social simulation engine that  
lets you define a social world, and  
calculates character desires, actions, and  
effects for you based on that definition

---

# Ensemble

A social AI / social simulation engine that lets you define a social world, and calculates character desires, actions, and effects for you based on that definition

---

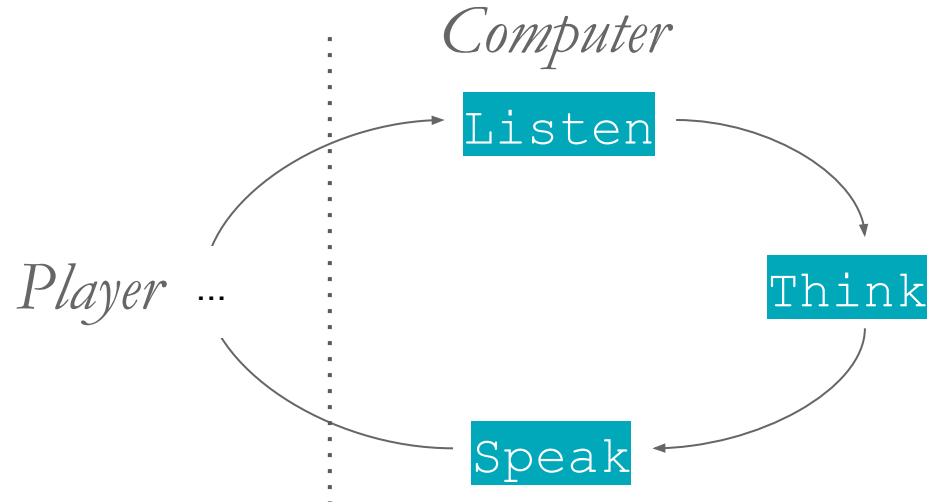
**But why?**

Social simulation allows you to make  
social interaction with fictional characters  
**playable**

Social simulation allows you to make  
social interaction with fictional characters  
**deeply interactive**

# Social simulation allows you to make social interaction with fictional characters **deeply interactive**

(In terms of Chris Crawford's loop:  
deepens the "Think" phase)



- Chris Crawford on Game Design, 2003

It lets you talk about things that are meaningful  
to everyday life

*(More so than combat & territory acquisition, anyway)*

---

It lets you talk about things that are meaningful  
to everyday life **and activate social pleasures**

It lets you talk about things that are meaningful to everyday life and activate social pleasures

---

E.g., **The feeling of hanging out with good friends**



Adventure Time, "Holly Jolly Secrets"

It lets you talk about things that are meaningful to everyday life and activate social pleasures

---

E.g., The feeling of hanging out with good friends

**Feeling enmeshed in a community**

↓ Stardew Valley — small town chats



@meldckn

It lets you talk about things that are meaningful to everyday life and activate social pleasures

---

E.g., The feeling of hanging out with good friends

**Feeling enmeshed in a community**



[Bad News](#)

— generates and simulates a community

```
Hiring of Hamil Eklof as Lawyer at Law Offices of Hamil Eklof in 197
Name change by which Marie Hendy became known as Marie Dewitz in 197
Name change by which Marie Hendy became known as Marie Dewitz in 197
Hiring of Hamil Eklof as Lawyer at Law Offices of Hamil Eklof in 197
Death of Herman Books in 1973
Move to apartment at 516 Dublin Avenue (Unit #30) by Jon Elbers in 1
Hiring of Robert Schlotman as Owner at Rogus Brewery in 1973
Retirement of Clifford Books as Owner at Rogus Brewery in 1973
Move to apartment at 516 Dublin Avenue (Unit #30) by Jon Elbers in 1
Retirement of Clifford Books as Owner at Rogus Brewery in 1973
Death of Herman Books in 1973
Move to apartment at 516 Dublin Avenue (Unit #30) by Jon Elbers in 1
Name change by which Marie Hendy became known as Marie Dewitz in 197
Hiring of Robert Schlotman as Owner at Rogus Brewery in 1973
Retirement of Clifford Books as Owner at Rogus Brewery in 1973
Move to apartment at 516 Dublin Avenue (Unit #30) by Jon Elbers in 1
Name change by which Marie Hendy became known as Marie Dewitz in 197
Hiring of Robert Schlotman as Owner at Rogus Brewery in 1973
Purchase of house at 417 9th Street by Hamil Eklof in 1973
```

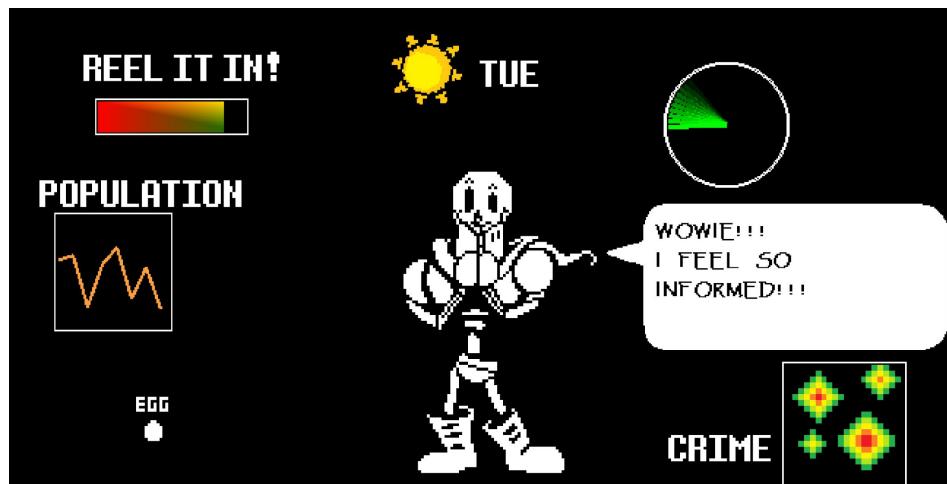
It lets you talk about things that are meaningful to everyday life and activate social pleasures

---

E.g., The feeling of hanging out with good friends

Feeling enmeshed in a community

**Interacting with compelling characters**



↑ *Undertale — dating minigame*

@meldckn

It lets you talk about things that are meaningful to everyday life and activate social pleasures

---

E.g., The feeling of hanging out with good friends

Feeling enmeshed in a community

Interacting with compelling characters

**Interacting with cuties**

→ Assassins Creed Origins – petting cats



# It lets you imagine and experiment with different ways of being with each other

---

Don't like some societal roles & rules?  
Use playful design to imagine new ones!

[@pixelatedcrown](#) — #ggj14 prototype →



@meldckn

# It lets you talk about social issues using systems

---

Use procedural rhetoric to talk about things like racism, sexism, unconscious bias, institutional discrimination, activism, etc



E.g., [societal violence & blending in](#), [terrorism](#), [activist organizing](#), [self-segregation](#)

# Social interactions & relationships are narratively rich

---

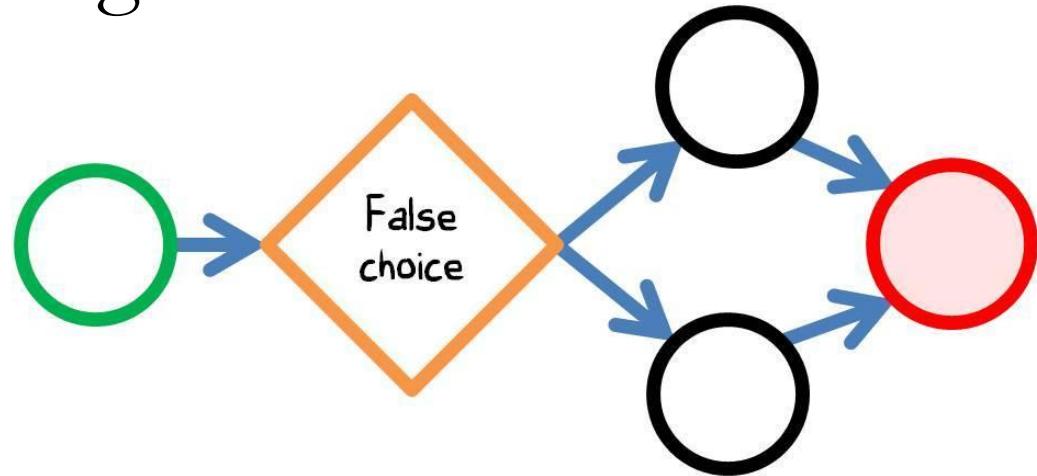
Can evoke strong emotions. Powerful source of tension in stories.



# Playable social interaction is underexplored in games

---

Usually baked into  
instantial elements,  
so not deeply  
interactive



*A simple "string of pearls"*

Social simulation with Ensemble allows you to make social interaction with fictional characters playable

# Ensemble

A social AI / social simulation engine that lets you define a social world, and calculates character desires, actions, and effects for you based on that definition

---

A Javascript library and [little language](#) that you call within the main code of a game or simulation

# Ensemble is...

## A JSON language

A JSON-embedded language in which you define a social world

```
schema.json *          volitionRules.json *          actions.json *          history.json *
```

```
1  t "schema": [ 2    { 3      "category" : "trait", 4      "comment" : "A personality aspect of one character", 5      "isBoolean" : true, 6      "directionType" : "undirected", 7      "actionable" : false, 8      "defaultValue" : false, 9      "types" : [
```

## A Javascript library

Whose functions you call within the main code of a game or simulation.  
Calculates character desires, actions, and effects based on your social world

```
ensemble.js *
```

```
1  (function () {/** 2   * @license almond 0.3.0 Copyright (c) 2011-2014, The Dojo Foundation All Rights Reserved. 3   * Available via the MIT or new BSD license. 4   * see: http://github.com/jrburke/almond for details 5   */ 6   //Going sloppy to avoid 'use strict' string cost, but strict practices should 7   //be followed. 8   /*jshint sloppy: true */ 9   /*global setTimeout: false */10 11  var requirejs, require, define;12  (function (undef) {
```

## An authoring tool

That helps you define [some of] your social world without having to write in tedious JSON

Ensemble Tool

Load New Schema

ACTIVE SCHEMA

Console Social Record Viewer Rules Viewer Rules Editor

Trigger Volition

Filter Rules: Everyone wants to increase closeness  
If: x is anyone, and y is anyone, Then: x has more volition (+5 ) to increase close-with for y

Mischief-makers like to make people laugh  
If: joker is mischievous, and audience is anyone, Then: joker has more volition (+5 ) to increase makes-laugh for audience

status actionable, undirected, boolean, duration 3

# Ensemble can be found on...

[github.com/ensemble-engine/ensemble](https://github.com/ensemble-engine/ensemble)

+ See the [releases page](#) for downloadables

```
schema.json * volitionRules.json * actions.json * history.json * ensemble.js *
```

```
1  "schema": [
2    {
3      "category" : "trait",
4      "comment" : "A personality aspect of one character",
5      "isBoolean" : true,
6      "directionType" : "undirected",
7      "actionable" : false,
8      "defaultValue" : false,
9      "type" : [
10        "string"
11      ]
12    }
13  ],
14  "volitionRules": [
15    {
16      "id": "rule1",
17      "name": "Rule 1"
18    },
19    {
20      "id": "rule2",
21      "name": "Rule 2"
22    }
23  ],
24  "actions": [
25    {
26      "id": "act1",
27      "name": "Action 1"
28    },
29    {
30      "id": "act2",
31      "name": "Action 2"
32    }
33  ],
34  "history": [
35    {
36      "id": "hist1",
37      "name": "History 1"
38    },
39    {
40      "id": "hist2",
41      "name": "History 2"
42    }
43  ],
44  "ensemble": [
45    {
46      "id": "ens1",
47      "name": "Ensemble 1"
48    },
49    {
50      "id": "ens2",
51      "name": "Ensemble 2"
52    }
53  ]
```

Ensemble Tool				Load New Schema
Console	Social Record Viewer	Rules Viewer	Rules Editor	
Volition	Filter Rules:		New Volition Rule	ACTIVE SCHEMA
	<input checked="" type="checkbox"/> Everyone wants to increase closeness		volitionRules	trait undirected, boolean introverted • sensitive • mischievous • jocular • crotchety • affable • gregarious • stoic • unflappable • aloof • anyone
	<input checked="" type="checkbox"/> Mischief-makers like to make people laugh		volitionRules	status actionable, undirected, duration 3

## **HOW TO USE ENSEMBLE**

1. Define JSON social world
2. Call functions in ensemble.js

## HOW TO USE IT

### 1. Define social world

Write JSON definitions for schema, cast, history, volition rules, trigger rules, actions. Often collected in a folder called `data`

`data/`

`schema.json`

`cast.json`

`history.json`

`volitions.json`

`triggers.json`

`actions.json`

## HOW TO USE IT

### 2. Call Ensemble functions

calculateVolitions	:	see what volitions a character has
getActions	:	see what actions a character wants to do
set	:	change social state
get	:	get social state
runTriggerRules	:	apply rules that conditionally change social state
dumpSocialRecord	:	print all history to the console
setNextTimeStep	:	progress timestep

→ the authoring tool does this for you (or at least makes it easier)

# HOW TO USE IT

## When prototyping

Use the authoring tool and a favorite text editor

The screenshot shows the Developer Tools console tab with the log tab selected. It displays the following log entries:

```
> top frame > □ Preserve log
domain/data/actions.json'; fileContents now > [object]
Adding '/Users/Melanie/Documents/Research/ensemble/literary-social-
domain/data/cast.json'; fileContents now > [object, Object]
Adding '/Users/Melanie/Documents/Research/ensemble/literary-social-
domain/data/history.json'; fileContents now > [object, Object]
Adding '/Users/Melanie/Documents/Research/ensemble/literary-social-
domain/data/schema.json'; fileContents now > [object, Object, Object]
Adding '/Users/Melanie/Documents/Research/ensemble/literary-social-
domain/data/triggerRules.json'; fileContents now > [object, Object, Object, Object]
Adding '/Users/Melanie/Documents/Research/ensemble/literary-social-
domain/data/volitionRules.json'; fileContents now
> [Object, Object, Object, Object, Object]
parsedData
> Object {fileName: "triggerRules", type: "trigger", rules: Array[0], source_file: "/Users/Melanie/Documents/Research/ensemble/literary-social-domain/data/triggerRules.json"}
parsedData
> Object {fileName: "volitionRules", type: "volition", rules: Array[3], source_file: "/Users/Melanie/Documents/Research/ensemble/literary-social-domain/data/volitionRules.json"}
TODO: Make Validate for reading in actions!
Validate
```

The screenshot shows the Ensemble Tool interface. At the top, there is a schema editor window displaying a JSON schema definition for 'schema.json'.

```
OPEN FILES
schema.json
volitionRules.json
actions.json
history.json
cast.json
Ensembler.js
TextStream.js
triggerRules.json
```

```
1 {
  "schema": [
    {
      "category": "trait",
      "comment": "A personality aspect of one character",
      "isTrait": true,
      "direction": "undirected",
      "actionable": false,
      "defaultValue": false,
      "types": [
        "introverted"
      ]
    }
  ]
}
```

The main area of the interface is the 'Ensemble Tool' window, which has tabs for 'Console', 'Social Record Viewer', 'Rules Viewer', and 'Rules Editor'. The 'Social Record Viewer' tab is active, showing a list of volitions and their outcomes.

Volition	Because	Acceptance
ara has more volition (+1) to increase close-with for galene	Because: Everyone wants to increase closeness (1)	galene would accept (1), because: Everyone wants to increase closeness (1)
> volitions(ara,hecate)		
ara has more volition (+1) to increase close-with for hecate	Because: Everyone wants to increase closeness (1)	hecate would accept (1), because: Everyone wants to increase closeness (1)
> volitions(ara,lyssa)		
ara has more volition (+1) to increase close-with for lyssa	Because: Everyone wants to increase closeness (1)	lyssa would accept (1), because: Everyone wants to increase closeness (1)
> volitions(ara,mnemosyne)		
ara has more volition (+1) to increase close-with for mnemosyne	Because: Everyone wants to increase closeness (1)	mnemosyne would accept (1), because: Everyone wants to increase closeness (1)

Below the table, there is a command input field labeled 'Command: |'.

On the right side of the interface, there is a sidebar titled 'ACTIVE SCHEMA' containing definitions for various traits, status, attitude-toward, and relationships.

**trait**: undirected, boolean  
introverted • sensitive • mischievous • jocular • crotchety • affable • gregarious • stoic • unflappable • aloof • anyone

**status**: actionable, undirected, boolean  
embarrassed • alone

**attitude-toward**: actionable, directed, numeric 0-10 (default 0)  
close-with • makes-laugh • dislike • friendly-towards • accepting-of • attracted-to • protective-over • playful-with • interested-in • admiring-of • infatuated-with • admirer-of • shy-around • suspicious-of • intimidated-by • awkward-around • (11 more)

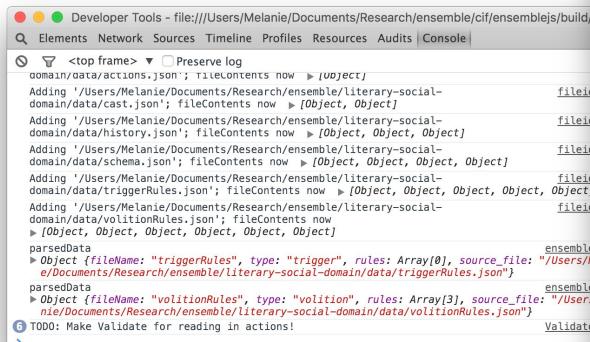
**relationship**: actionable, reciprocal, boolean  
constant-companions • kindred-spirits • acquaintances • barely-know-each-other • old-friends • partners • ex-partners • housemates • accomplices • peers • colleagues • creative-collaborators • rivals

+ New

# HOW TO USE IT

## When prototyping

Use the authoring tool and a favorite text editor



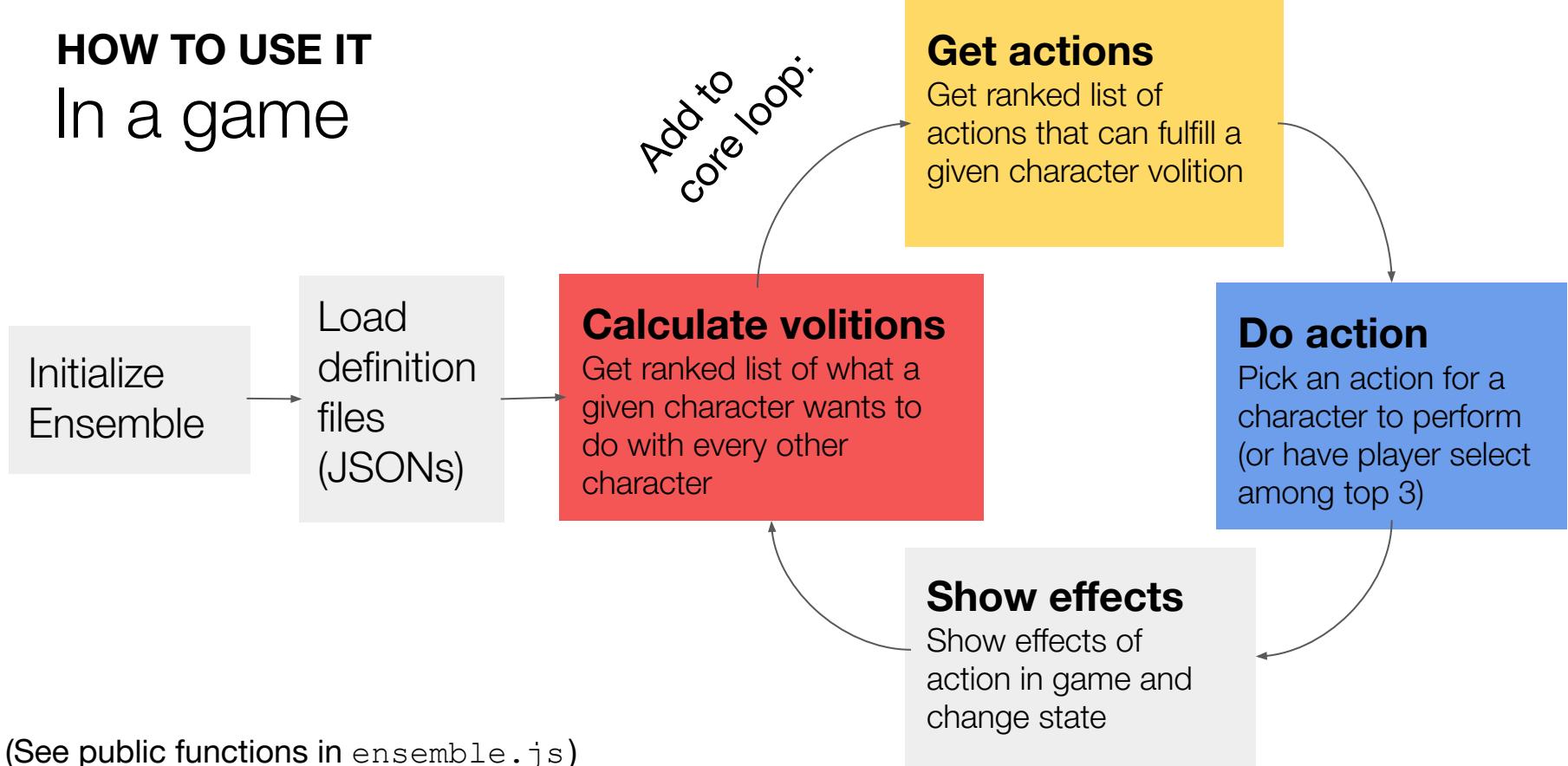
(Recommend also having the Developer Tools Console open)

The screenshot shows the Ensemble Tool interface with the following components:

- Schema Editor:** A code editor window titled "schema.json" showing JSON schema code. A gear icon is highlighted with a blue box.
- Ensemble Tool Window:** A main window titled "Ensemble Tool" containing:
  - Console Tab:** Displays log entries similar to the developer tools.
  - Social Record Viewer:** Shows a list of volition records:
    - ara has more volition (+1) to increase close-with for galene
    - hecate would accept (1), because: Everyone wants to increase closeness (1)
    - Because: galene would accept (1), because: Everyone wants to increase closeness (1)
  - Rules Editor:** Shows a list of volition records:
    - ara has more volition (+1) to increase close-with for hecate
    - hecate would accept (1), because: Everyone wants to increase closeness (1)
    - Because: ara has more volition (+1) to increase close-with for hecate
  - Rules Viewer:** Shows a list of volition records:
    - ara has more volition (+1) to increase close-with for lyssa
    - lyssa would accept (1), because: Everyone wants to increase closeness (1)
    - Because: ara has more volition (+1) to increase close-with for lyssa
  - Volitions Tab:** Shows a list of volition records:
    - ara has more volition (+1) to increase close-with for mnemosyne
    - mnemosyne would accept (1), because: Everyone wants to increase closeness (1)
    - Because: ara has more volition (+1) to increase close-with for mnemosyne
- ACTIVE SCHEMA:** A sidebar listing traits, status, attitude-toward, and relationships with their definitions.
- Bottom Status Bar:** Shows log files and tab size.

# HOW TO USE IT

## In a game



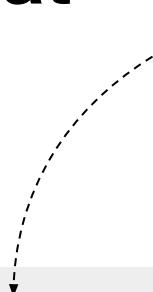
# How Ensemble works at a high level:

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

# How it works at a high level:

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# How it works at a high level:



**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

# How it works at a high level:

The diagram illustrates a process flow. At the top left, there is a dashed oval. A dashed arrow points from this oval down to a horizontal bar. From the center of this bar, another dashed arrow points down to a rectangular box. Inside this box, the text describes the workflow: 'Volition rules' (in red), 'use social state' (in blue), 'to form volitions' (in red), 'for characters, which are translated into actions.'

Volition rules use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# How it works at a high level:

The diagram features a central light gray rectangular box containing text. Four dashed arrows originate from the top and bottom edges of this box and curve outwards towards the corners of the slide. The text inside the box reads: "Volition rules use social state to form volitions for characters, which are translated into actions." The words "Volition rules", "social state", "volitions", and "actions" are highlighted in red, blue, red, and yellow respectively.

Volition rules use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# How it works at a high level:

The diagram features a central rectangular box containing text. Above the box, three dashed arrows point downwards towards its top edge. From the bottom edge of the box, a single dashed arrow points upwards. The text inside the box reads: "Volition rules use social state to form volitions for characters, which are translated into actions."

Volition rules use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# How it works at a high level:

The diagram consists of four large, light-gray rounded rectangles arranged in a sequence. A dashed arrow points from the first rectangle down to the second. Another dashed arrow points from the second rectangle down to the third. A solid arrow points from the third rectangle up to the fourth. Inside the first rectangle, the text 'Volition rules' is in a red box. Inside the second rectangle, 'social state' is in a blue box. Inside the third rectangle, 'volitions' is in a red box. Inside the fourth rectangle, 'actions.' is in a yellow box.

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions.**

# How it works at a high level:

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# How it works at a high level:

Current social facts  
about author-defined  
schema categories



History

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

# How it works at a high level:

Current social facts  
about author-defined  
schema categories

+

History

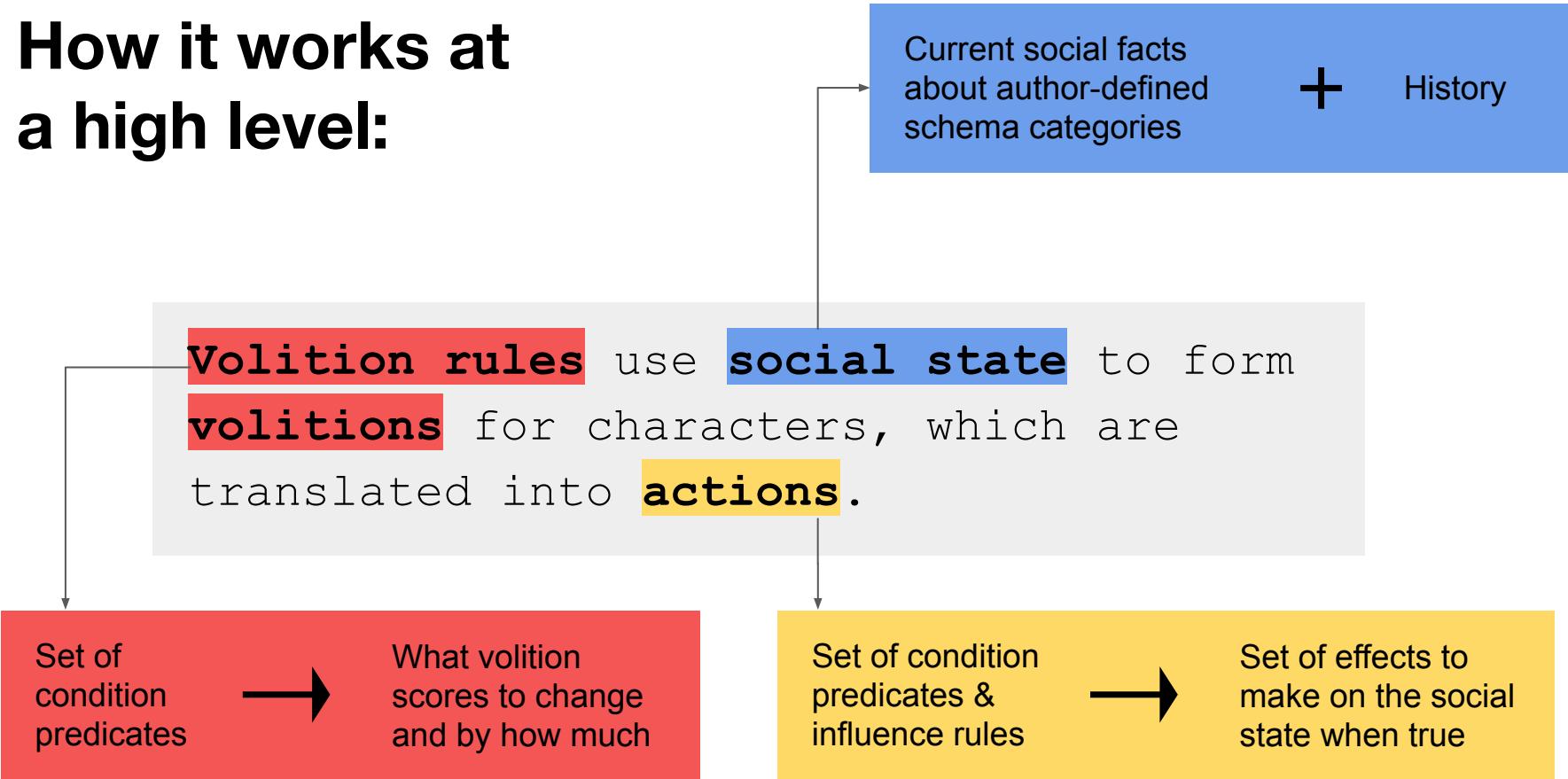
**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

Set of  
condition  
predicates



What volition  
scores to change  
and by how much

# How it works at a high level:



# How it works at a high level:

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# Social State

Current social facts  
about author-defined  
schema categories



History

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# Social State

Current social facts  
about author-defined  
schema categories



History

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

schema.json

history.json

# Social State

Current social facts  
about author-defined  
schema categories



History

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

schema.json



history.json



# Schema.json

Where you define what concepts exist in your social world

- Your world's 'data types'
- We recommend new authors start with a pre-built schema, before defining their own

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "isBoolean" : true,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : false,  
      "types" : [  
        "introverted",  
        "sensitive",  
        "mischievous",  
        "crotchety",  
        "gregarious",  
        "stoic",  
        "unflappable",  
        "aloof",  
        "anyone"  
      ],  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

## category

Name of a schema concept. Can be any string (no underscores).

E.g., "trait",  
"relationship",  
"status"

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : true,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : false,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

## types

Names of individual types within this schema category. Each type defines a separately maintained state variable for each character or pair of characters. An array of strings. The strings can be anything.

E.g.,

“flustered”,  
“bffs”,  
“admires”

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : true,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : false,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

## **isBoolean**

Defines whether the types in this schema category are scalar-valued or boolean. Can only be true or false.

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : true,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : false,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

E.g., This example is saying that, in the social world we're defining, characters can be introverted or not (same for extroverted). If isBoolean were false, characters could be some degree of introverted and extroverted (like introverted with a value of 2 on a scale of 1 to 10)

## **directionType**

Defines whether the types in this schema category are about one person or about a relationship between two characters. If it's about two characters, it can be symmetrical (reciprocal) or assymetrical (directed). Can only be "undirected", "directed", or "reciprocal".

```
{  
    "schema": [  
        {  
            "category" : "trait",  
            "types" : [  
                "introverted", "extroverted"  
            ],  
            "isBoolean" : true,  
            "directionType" : "undirected",  
            "actionable" : false,  
            "defaultValue" : false,  
            "comment" :  
                "Aspect of a character's personality"  
        }  
    ]  
}
```

E.g., This is saying that "traits" in this world only pertain to one character (undirected)

## actionable

Defines whether characters can form desires (or volitions) to change the values of these types for a particular character or pair of characters. Can only be true or false.

Even if this is true, you still have to author volition rules for characters to actually form volitions around the relevant schema types.

```
{  
    "schema": [  
        {  
            "category": "trait",  
            "types": [  
                "introverted", "extroverted"  
            ],  
            "isBoolean": true,  
            "directionType": "undirected",  
            "actionable": false,  
            "defaultValue": false,  
            "comment":  
                "Aspect of a character's personality"  
        }  
    ]  
}
```

E.g., This is saying that characters *cannot* form volitions to change their own traits (or the traits of others) in this world.

## **defaultValue**

Specifies the default value for all variables of the category and types defined here. Either true or false if the category isBoolean, or a number if not.

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : true,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : false,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

E.g., Characters start with all their trait values (introverted and extroverted) equal to false.

# If social state is numerical:

- isBoolean should be false

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : false,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : false,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

# If social state is numerical:

- `isBoolean` should be false
- `defaultValue` should be scalar

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : false,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : 0,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

# If social state is numerical:

- `isBoolean` should be false
- `defaultValue` should be scalar
- `minValue` and `maxValue` should be defined

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : false,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : 0,  
      " minValue" : 0,  
      " maxValue" : 10,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

# If social state is numerical:

**minValue**  
**maxValue**

Specifies the range of values that variables of this category can hold. (e.g., if an effect would increase a variable past its `maxValue`, it would stay at its current value instead.)

```
{  
  "schema": [  
    {  
      "category" : "trait",  
      "types" : [  
        "introverted", "extroverted"  
      ],  
      "isBoolean" : false,  
      "directionType" : "undirected",  
      "actionable" : false,  
      "defaultValue" : 0,  
      "minValue" : 0,  
      "maxValue" : 10,  
      "comment" :  
        "Aspect of a character's personality"  
    }  
  ]  
}
```

# If social state expires over time:

- duration should be defined

```
{  
    "schema": [  
        {  
            "category" : "mood",  
            "types" : [  
                "angry", "embarrassed"  
            ],  
            "isBoolean" : false,  
            "directionType" : "undirected",  
            "actionable" : false,  
            "defaultValue" : false,  
            "duration" : 5,  
            "comment" :  
                "A temporary emotion or mood"  
        }  
    ]  
}
```

# If social state expires over time:

- duration should be defined

## duration

Specifies how long variables of this schema category last, in timesteps. Must be a number. Good for moods and other temporary state.

```
{  
    "schema": [  
        {  
            "category" : "mood",  
            "types" : [  
                "angry", "embarrassed"  
            ],  
            "isBoolean" : false,  
            "directionType" : "undirected",  
            "actionable" : false,  
            "defaultValue" : false,  
            "duration" : 5,  
            "comment" :  
                "A temporary emotion or mood"  
        }  
    ]  
}
```

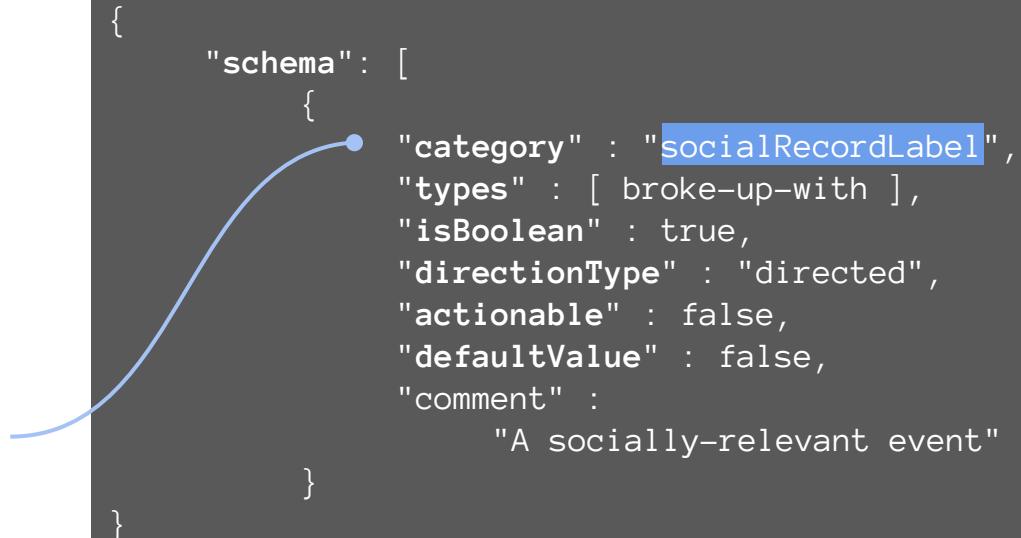
E.g., This now says that traits last for 5 timesteps. Five timesteps after a variable trait is changed, it will revert back to its category's defaultValue

# If you want events

- These are special concepts that get added to the history when triggered
- Make a category with a special name:

**socialRecordLabel** — If this is a type of event that happens between two people (`directionType == directed` or `reciprocal`). E.g., someone breaking up with someone else, or

**socialRecordLabelUndirected** — If it's a type of event that only happens to one person. E.g., someone coming out



# Custom properties

You can also include any other key-value pair in any Ensemble JSON object. For example, having a human-readable description of the object is often handy. You might also have domain-specific data that interfaces with your game engine here

```
{  
    "schema": [  
        {  
            "category" : "trait",  
            "types" : [  
                "introverted", "extroverted"  
            ],  
            "isBoolean" : true,  
            "directionType" : "undirected",  
            "actionable" : false,  
            "defaultValue" : false,  
            "comment" :  
                "Aspect of a character's personality"  
        }  
    ]  
}
```

# Schema.json Fields In Brief

Schema	
Key	Value
category	String
types	Array of strings
directionType	String: "directed", "undirected", or "reciprocal"
isBoolean	Boolean: true or false
actionable	Boolean: true or false
defaultValue	Boolean if isBoolean==true, number if isBoolean==false
<i>Sometimes</i>	
<i>minValue</i>	<i>Number (only needed if isBoolean==false)</i>
<i>maxValue</i>	<i>Number ("")</i>
<i>duration</i>	<i>Number</i>

# Schema through the lens of the Authoring Tool

Much easier to write in, but it's helpful to know the underlying representation.

### Edit Schema Category

Category Name **trait**

Direction  Undirected  Directed  Reciprocal

Data Type  True/False  Numeric

Default  True  False

Duration?

Actionable?

Types introverted x sensitive x mischievous x  
jocular x crotchety x affable x  
gregarious x stoic x unflappable x  
aloof x anyone x NEW

Statistics These schema entries involve this category:

- No matching trigger rules.
- *Everyone wants to increase closeness* and *2 other* volition rules
- *At time 0: antheia is mischievous* and *15 other* social records
- No matching actions.

**Delete** **Close**

# Schema through the lens of the Authoring Tool

Much easier to write in, but it's helpful to know the underlying representation.

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)  
closeness • attraction

**mood** undirected, boolean, duration 3  
confident • unsure

**attribute** actionable, undirected, numeric  
0-->100 (default 0)  
strength • intelligence

**trait** undirected, boolean  
named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,  
single turn  
romantic-failure • romantic-advance

**socialRecordLabelUndirected**  
undirected, boolean, single turn  
embarrassing • self-involved

+ New

# More Social State

Current social facts  
about author-defined  
schema categories



History

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

schema.json

history.json

# History

Current social facts  
about author-defined  
schema categories



History

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

schema.json

history.json

# History.json

Where you initialize social state between characters, and fill in the game's backstory (what happened before play begins)

```
{  
  "history": [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category": "trait",  
          "type": "mischiefous",  
          "first": "ara",  
          "value": true  
        }, {  
          "category": "relationship",  
          "type": "best-buds",  
          "first": "calypso",  
          "second": "ara",  
          "value": true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

An array of objects.

```
{  
  "history": [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category" : "trait",  
          "type" : "mischiefous",  
          "first" : "ara",  
          "value" : true  
        }, {  
          "category" : "relationship",  
          "type" : "best-buds",  
          "first" : "calypso",  
          "second" : "ara",  
          "value" : true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

An array of objects.

Each of those objects have a time value (“pos” for position), and “data”, which itself is an array of state objects.

```
{  
  "history":  
  [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category" : "trait",  
          "type" : "mischiefous",  
          "first" : "ara",  
          "value" : true  
        }, {  
          "category" : "relationship",  
          "type" : "best-buds",  
          "first" : "calypso",  
          "second" : "ara",  
          "value" : true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

An array of objects.

Each of those objects have a time value (“pos” for position), and “data”, which itself is an array of state objects.

Each state object in a data array indicates what state variable you’re talking about, its value, and the character(s) who has that state value.

```
{  
  "history": [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category" : "trait",  
          "type" : "mischiefous",  
          "first" : "ara",  
          "value" : true  
        }, {  
          "category" : "relationship",  
          "type" : "best-buds",  
          "first" : "calypso",  
          "second" : "ara",  
          "value" : true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

Fields for a state object within the data array:

**category**  
What schema category you're referencing

**type**  
What type within that category that you're referencing

```
{  
  "history": [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category": "trait",  
          "type": "mischiefous",  
          "first": "ara",  
          "value": true  
        }, {  
          "category": "relationship",  
          "type": "best-buds",  
          "first": "calypso",  
          "second": "ara",  
          "value": true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

Fields for a state object within the data array:

**category**

What schema category you're referencing

**type**

What type within that category that you're referencing

```
{"schema": [ { "category" : "trait", "types" : [ "mischievous" ], ... } ] }
```

```
{ "history": [ { "pos": 0, "data": [ { "category" : "trait", "type" : "mischievous", "first" : "ara", "value" : true }, { "category" : "relationship", "type" : "best-buds", "first" : "calypso", "second" : "ara", "value" : true } ] } ] }
```

# History.json

Fields for a state object within the data array:

## first

The name of the character who has this state

```
{  
  "history": [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category": "trait",  
          "type": "mischievous",  
          "first": "ara",  
          "value": true  
        }, {  
          "category": "relationship",  
          "type": "best-buds",  
          "first": "calypso",  
          "second": "ara",  
          "value": true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

Fields for a state object within the data array:

## first

The name of the character who has this state

## second

If the schema category you're referencing is between two people (i.e., its directionType == reciprocal or undirected), you'll need the name of the second character this state is about

```
{  
  "history": [  
    {  
      "pos": 0,  
      "data": [  
        {  
          "category": "trait",  
          "type": "mischievous",  
          "first": "ara",  
          "value": true  
        }, {  
          "category": "relationship",  
          "type": "best-buds",  
          "first": "calypso",  
          "second": "ara",  
          "value": true  
        }  
      ]  
    }  
  ]  
}
```

# History.json

Fields for a state object within the data array:

## value

The state value. Boolean if schema category's isBoolean==true, or a number if false

A diagram illustrating the JSON structure of History.json. On the left, a blue box highlights the 'value' field. Two curved arrows point from this box to two 'value' fields in the JSON code on the right. The first arrow points to the 'value' field in the first object of the 'data' array, which is true. The second arrow points to the 'value' field in the second object of the 'data' array, which is also true.

```
{
  "history": [
    {
      "pos": 0,
      "data": [
        {
          "category": "trait",
          "type": "mischiefous",
          "first": "ara",
          "value": true
        },
        {
          "category": "relationship",
          "type": "best-buds",
          "first": "calypso",
          "second": "ara",
          "value": true
        }
      ]
    }
  ]
}
```

**Now we're going to use social state to determine what characters want to do with each other**

# By connecting our social data types...

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# By connecting our social data types... to character desires

**Volition rules** use **social state** to form  
**volitions** for characters, which are  
translated into **actions**.

# Volitions & Volition Rules

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

Set of condition predicates



What volition scores to change and by how much

# Volitions & Volition Rules

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

Set of condition predicates



What volition scores to change and by how much

volitionRules.json



# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

# Connecting Action Nodes

## leadsTo

An array of action node names which are subactions of this one

All intent-level nodes need to leadTo something. But normal action nodes can also have a set of subactions.

```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your **schema**.

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)

closeness • attraction

**mood** undirected, boolean, duration 3

confident • unsure

**attribute** actionable, undirected, numeric

0-->100 (default 0)

strength • intelligence

**trait** undirected, boolean

named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,

single turn

romantic-failure • romantic-advance

**socialRecordLabelUndirected**

undirected, boolean, single turn

embarrassing • self-involved

+ New

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)

closeness • attraction

**mood** undirected, boolean, duration 3

confident • unsure

**attribute** actionable, undirected, numeric

0-->100 (default 0)

strength • intelligence

**trait** undirected, boolean

named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,

single turn

romantic-failure • romantic-advance

**socialRecordLabelUndirected**

undirected, boolean, single turn

embarrassing • self-involved

+ New

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)

closeness • attraction

**mood** undirected, boolean, duration 3

confident • unsure

**attribute** actionable, undirected, numeric

0-->100 (default 0)

strength • intelligence

**trait** undirected, boolean

named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,

single turn

romantic-failure • romantic-advance

**socialRecordLabelUndirected**

undirected, boolean, single turn

embarrassing • self-involved

+ New

...But you probably won't want characters to form volitions about every schema concept

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)

closeness • attraction

**mood** undirected, boolean, duration 3

confident • unsure

**attribute** actionable, undirected, numeric

0-->100 (default 0)

strength • intelligence

**trait** undirected, boolean

named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,

single turn

romantic-failure • romantic-advance

**socialRecordLabelUndirected**

undirected, boolean, single turn

embarrassing • self-involved

+ New

...But you probably won't want characters to form volitions about every schema concept

- Some schema categories describe innate character qualities, that shouldn't change over the course of the game

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)

closeness • attraction

**mood** undirected, boolean, duration 3

confident • unsure

**attribute** actionable, undirected, numeric

0-->100 (default 0)

strength • intelligence

**trait** undirected, boolean

named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,

single turn

romantic-failure • romantic-advance

**socialRecordLabelUndirected**

undirected, boolean, single turn

embarrassing • self-involved

+ New

...But you probably won't want characters to form volitions about every schema concept

- Some schema categories describe innate character qualities, that shouldn't change over the course of the game
- And some schema categories change, but not as a result of intentional goal formation

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence

## ACTIVE SCHEMA

**feeling** actionable, directed, numeric 0-->100 (default 0)

closeness • attraction

**mood** undirected, boolean, duration 3

confident • unsure

**attribute** actionable, undirected, numeric

0-->100 (default 0)

strength • intelligence

**trait** undirected, boolean

named hero • named love • named  
rival • anyone

**socialRecordLabel** directed, boolean,

single turn

romantic-failure • romantic-advance

**socialRecordLabelUndirected**

undirected, boolean, single turn

embarrassing • self-involved

+ New

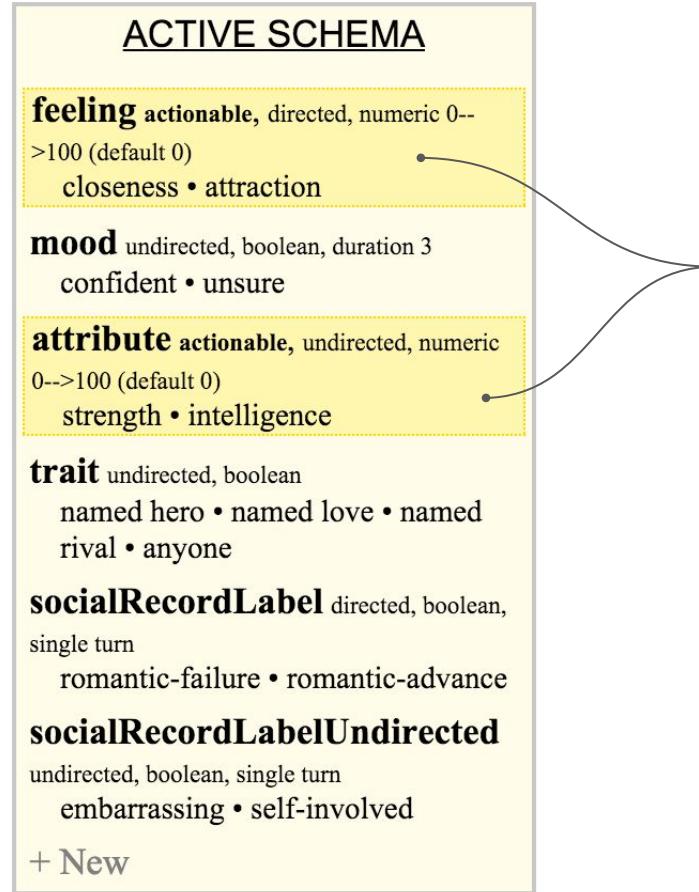
...But you probably won't want characters to form volitions about every schema concept

**Only actionable ones**

# Volitions

Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your schema.

E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence



(The tool highlights schema categories that you defined as actionable)

# VolitionRules.json

Where you define a set of rules that calculate character desires by modifying volition scores

## One rule :=

- Array of conditions
- Array of effects
- Both reference schema concepts in terms of category and type  
(similar to history file)

```
{ "fileName": "volitionRules",
  "type": "volition",
  "rules": [
    {
      "name": "Introverts like to be alone",
      "conditions": [
        { "category": "trait",
          "type": "introverted",
          "first": "initiator",
          "value": true }
      ],
      "effects": [
        { "category": "status",
          "type": "alone",
          "first": "initiator",
          "weight": 5,
          "intentType": true,
          }
      ]
    }
  ]
}
```

# Volition Rule Editor

Strongly recommend writing rules in the authoring tool, rather than JSON directly

Source File: volitionRules.json

**Volition Rule** Name: Introverts like to be alone

**When this is true:**

initiator is introverted .

**These character volitions are altered:**

initiator has more volition ( +5 ) to become alone .

The screenshot shows a user interface for creating a 'Volition Rule'. At the top, there's a 'Source File' dropdown set to 'volitionRules.json', followed by buttons for 'Update Rule', 'Delete', 'Test Rule', 'Redo', and 'Undo'. Below this is a section titled 'Volition Rule' with the name 'Introverts like to be alone'. A yellow box labeled 'When this is true:' contains a condition: 'initiator is introverted'. A green box labeled 'These character volitions are altered:' contains an effect: 'initiator has more volition ( +5 ) to become alone'. There are 'Add Condition' and 'Add Effect' buttons.

# Volition Rule Editor

Source File: volitionRules.json     Redo Undo

**Volition Rule** Name: Introverts like to be alone

**When this is true:**

 initiator  introverted  . 

**Add Condition**

These character volitions are altered:

 initiator has more volition (  +5 ) to  

alone  .

**Add Effect**

# JSON

```
{  
  "name":  
    "Introverts like to be alone",  
  "conditions": [  
    { "category": "trait",  
      "type": "introverted",  
      "first": "initiator",  
      "value": true }  
  ],  
  "effects": [  
    { "category": "status",  
      "type": "alone",  
      "first": "initiator",  
      "weight": 5,  
      "intentType": true, }  
  ]  
}
```

@meldckn

# Volition Rule Editor

Source File: volitionRules.json Update Rule Delete Test Rule Redo Undo

Volition Rule Name: Introverts like to be alone

**When this is true:**

initiator is introverted .

Add Condition

These character volitions are altered:

initiator has more volition ( +5 ) to become

alone .

Add Effect

# JSON

```
{  
  "name":  
    "Introverts like to be alone",  
  "conditions": [  
    { "category": "trait",  
      "type": "introverted",  
      "first": "initiator",  
      "value": true }  
  ],  
  "effects": [  
    { "category": "status",  
      "type": "alone",  
      "first": "initiator",  
      "weight": 5,  
      "intentType": true, }  
  ]  
}
```

@meldckn

# Calculating Volitions from Volition Rules

## **When you ask Ensemble to calculate volitions**

- It binds the roles (“first” and “second”) that appear in each role condition array with every possible character (or pair of characters).

# Calculating Volitions from Volition Rules

## **When you ask Ensemble to calculate volitions**

- It binds the roles (“first” and “second”) that appear in each role condition array with every possible character (or pair of characters).
- If all condition predicates in a rule are met, the rule fires, and its effect predicates are applied (incrementing or decrementing volition scores).

# Calculating Volitions from Volition Rules

## **When you ask Ensemble to calculate volitions**

- It binds the roles (“first” and “second”) that appear in each role condition array with every possible character (or pair of characters).
- If all condition predicates in a rule are met, the rule fires, and its effect predicates are applied (incrementing or decrementing volition scores).
- Once all volition rules are applied, you’ll end up with total accumulated scores for each volition for each character.

# Calculating Volitions from Volition Rules

## **When you ask Ensemble to calculate volitions**

- It binds the roles (“first” and “second”) that appear in each role condition array with every possible character (or pair of characters).
- If all condition predicates in a rule are met, the rule fires, and its effect predicates are applied (incrementing or decrementing volition scores).
- Once all volition rules are applied, you’ll end up with total accumulated scores for each volition for each character.
- Finally, you have a rated list of what high-level thing each character wants to do with every other character.

# Calculating Volitions from Volition Rules

From this:

**If you're my friend & you're having a hard time**  
→ I really want to help you (+10)

**If I'm nice & you're having a hard time**  
→ I want to help you (+5)

**If you recently helped me**  
→ I want to want to return the favor (+5)

...

All the volition rules that say when someone wants  
to help someone else

# Calculating Volitions from Volition Rules

From this:

If you're my friend & you're having a **hard time**  
→ I really want to help you (+10)

If I'm nice & you're having a **hard time**  
→ I want to help you (+5)

If you recently helped me  
→ I want to want to return the favor (+5)

...

All the volition rules that say when someone wants  
to help someone else

To this:

**Steven** has a volition score  
of **20** to **help Onion**

A score indicating how  
much a particular character  
wants to help a particular  
other character

# Authoring Volition Rules

Volition Rule Name: Mischief-makers like to make people laugh at another's expense

*When this is true:*

joker is mischievous . x

audience is anyone . x

Add Condition

These character volitions are altered:

joker has more volition (+5) to increase makes-laugh for audience .

Add Effect

The screenshot shows a user interface for creating a 'Volition Rule'. At the top, it says 'Name: Mischief-makers like to make people laugh at another's expense'. Below this, under 'When this is true:', there are two condition blocks. Each block consists of a subject (joker or audience), a verb ('is'), and a predicate ('mischievous' or 'anyone'). There are also small icons for editing and deleting each block. A green 'Add Condition' button is located below the first block. Below this section, a large green arrow points right, labeled 'These character volitions are altered:' in bold. Underneath the arrow, there are two effect blocks. Each block starts with a subject (joker or audience), followed by 'has more volition', a value input ('+5'), 'to increase', a predicate ('makes-laugh'), and 'for' followed by the other character. A green 'Add Effect' button is located below the second effect block.

## Conditions

Can have any number of condition predicates

# Authoring Volition Rules

Volition Rule Name: Mischief-makers like to make people laugh at another's expense

**When this is true:**

- joker is mischievous . x
- audience is anyone . x

Add Condition

These character volitions are altered:

- joker has more volition ( +5 ) to increase makes-laugh for
- audience .

Add Effect

## Conditions

A rule's condition predicates are ANDed together

i.e., all predicates must be true (for a particular character binding) for the rule to fire and the effects applied to those characters' volition scores

# Authoring Volition Rules

Volition Rule Name: Mischief-makers like to make people laugh at another's expense

**When this is true:**

- joker is mischievous . X
- audience is anyone . X

Add Condition

**These character volitions are altered:**

- joker has more volition ( +5 ) to increase makes-laugh for
- audience .

Add Effect



## Effects

Can also affect any number of character volitions  
(but usually one)

# Authoring Volition Rules

## Roles

A character role can be any string, and you can have any number of roles in a rule. They're like local variables, that only exist in the scope of a rule.

The screenshot shows a user interface for creating a 'Volition Rule'. At the top, there's a toolbar with buttons for 'Source File' (set to 'volitionRules.json'), 'Update Rule' (green), 'Delete' (red), 'Test Rule' (green), and 'Redo/Undo' (grey). Below the toolbar, the rule is named 'Introverts like to be alone'. The interface is divided into two main sections: 'When this is true:' (yellow background) and 'These character volitions are altered:' (green background).

**When this is true:**

initiator is introverted .

**Add Condition**

**These character volitions are altered:**

initiator has more volition (+5) to become alone .

**Add Effect**

# Authoring Volition Rules

## Roles

But roles are bound to characters in the **condition predicates**, not in **effects**.

So any role that appears in the effects must be “initialized” in a condition.

The screenshot shows a software interface for creating a Volition Rule. At the top, there's a toolbar with buttons for 'Source File' (set to 'volitionRules.json'), 'Update Rule' (green), 'Delete' (red), 'Test Rule' (green), and 'Redo/Undo'. Below the toolbar, the rule is named 'Introverts like to be alone'. The interface is divided into two main sections: 'When this is true:' (yellow background) and 'These character volitions are altered:' (green background). In the 'When this is true:' section, there's a condition: 'initiator is introverted'. A green 'Add Condition' button is below it. In the 'These character volitions are altered:' section, there's an effect: 'initiator has more volition (+5) to become alone'. A green 'Add Effect' button is below it. A large black arrow points from the text 'not in effects.' in the explanatory text to the 'These character volitions are altered:' section.

# Authoring Volition Rules

## Roles

But roles are bound to characters in the **condition predicates**, not in **effects**.  
So any role that appears in the effects must be “initialized” in a condition (or you could think of it as needing to check that a character exists before affecting them).

The screenshot shows a user interface for creating a Volition Rule. At the top, there are buttons for 'Source File: volitionRules.json' (with a dropdown arrow), 'Update Rule' (green), 'Delete' (red), 'Test Rule' (green), 'Redo' (light blue), and 'Undo' (light blue). Below this, the rule is named 'Introverts like to be alone'. The interface is divided into two main sections: 'When this is true:' (yellow background) and 'These character volitions are altered:' (green background). In the 'When this is true:' section, there is a condition: 'initiator is introverted'. A green 'Add Condition' button is below it. In the 'These character volitions are altered:' section, there is an effect: 'initiator has more volition (+5) to become alone'. A green 'Add Effect' button is below it. A large black arrow points from the text 'not in effects.' in the slide content to the 'These character volitions are altered:' section.

# Authoring Volition Rules

## Roles

So this won't work:

(Rule will never fire,  
and will not throw an  
error.)



The screenshot shows a user interface for creating character volition rules. The top section, titled "When this is true:", contains a condition: "joker is mischievous". Below this is a green button labeled "Add Condition". A large black arrow points from the word "These" in the title of the bottom section to the character "joker" in the condition. The bottom section, titled "These character volitions are altered:", contains two entries: "joker has more volition (+5) to increase makes-laugh for audience" and "audience". Below this is a green button labeled "Add Effect".

*When this is true:*

joker is mischievous .

Add Condition

*These character volitions are altered:*

joker has more volition (+5) to increase makes-laugh for audience .

Add Effect

# Authoring Volition Rules

## Roles

Corrected:

**Volition Rule** Name: Mischief-makers like to make people laugh at another's expense

*When this is true:*

joker is mischievous . 🕒 X

audience is anyone . 🕒 X

**Add Condition**

**These character volitions are altered:**

joker has more volition ( +5 ) to increase makes-laugh for audience .

**Add Effect**

```
graph TD; A[joker is mischievous] --- B[audience is anyone]; B --> C[These character volitions are altered: joker has more volition (+5) to increase makes-laugh for audience]; C --> D[audience]
```

# Authoring Volition Rules

## Roles

That's why it's handy to have a schema type like "anyone" that you set as true for every character before play begins.

**Volition Rule** Name: Mischief-makers like to make people laugh at another's expense

*When this is true:*

joker is mischievous . 🕒 X

audience is anyone . 🕒 X

**Add Condition**

*These character volitions are altered:*

joker has more volition ( +5 ) to increase makes-laugh for

audience .

**Add Effect**

@meldckn

# Authoring Volition Rules

Volition Rule Name: People who have had a recent falling-out do not want to connect

When this is true:

someone is falling-out . other

Add Condition

These character volitions are altered:

someone has less volition ( -5 ) to increase close-with for other .

Add Effect

## Referencing History

Select the clock icon next to a condition predicate

# Authoring Volition Rules

Volition Rule Name: People who have had a recent falling-out do not want to connect

When this is true:

someone has been falling-out ↴ other .

↪ sometime between NOW and 5 turns ago

Add Condition

These character volitions are altered:

someone has less volition ( -5 ↴ ) to increase close-with for

other .

Add Effect

## Referencing History

Specify time frame

# Now we'll enable characters to take action based on their desires

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

# Now we'll enable characters to take action based on their desires

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.



Set of condition predicates & influence rules



Set of effects to make on the social state when true

# Now we'll enable characters to take action based on their desires

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

actions.json

```
graph TD; A[actions.json] --> B["Set of condition predicates & influence rules"]; B --> C["Set of effects to make on the social state when true"]
```

Set of condition predicates & influence rules



Set of effects to make on the social state when true

# Actions

Unfortunately, no GUI tool for action authoring yet, so you have to edit JSON directly to add/edit actions

```
{  
  "fileName" : "actions.json",  
  "actions" : [  
    {  
      "name" : "connect",  
      "displayName" : "connect",  
      "influenceRules" : [],  
      "conditions" : [],  
      "intent" :  
        {  
          "category" : "relationship",  
          "type" : "closeness",  
          "intentType" : true,  
          "first" : "initiator",  
          "second" : "responder"  
        },  
      "leadsTo" : ["chat", "makeLaugh", "confideIn"]  
    }  
    ...  
  ]  
}
```

# Action JSON Basics

Array of objects

Each object is an action node

Each action node has a name, an array of influence rules, and an array of conditions (can be empty). All other properties are only sometimes required.

```
{  
  "fileName" : "actions.json",  
  "actions" : [  
    {  
      "name" : "connect",  
      "displayName" : "connect",  
      "influenceRules" : [],  
      "conditions" : [],  
      "intent" :  
        {  
          "category" : "relationship",  
          "type" : "closeness",  
          "intentType" : true,  
          "first" : "initiator",  
          "second" : "responder"  
        },  
      "leadsTo" : ["chat", "makeLaugh", "confideIn"]  
    }  
    ...  
  ]  
}
```

# Action JSON Basics

Array of objects

Each object is an action node

Each action node has a name, an array of influence rules, and an array of conditions (can be empty). All other properties are only sometimes required.

## Action Node Properties

### Always required

```
"name"  
"influenceRules"  
"conditions"
```

### Sometimes required,

Depending on where the node is in an action graph

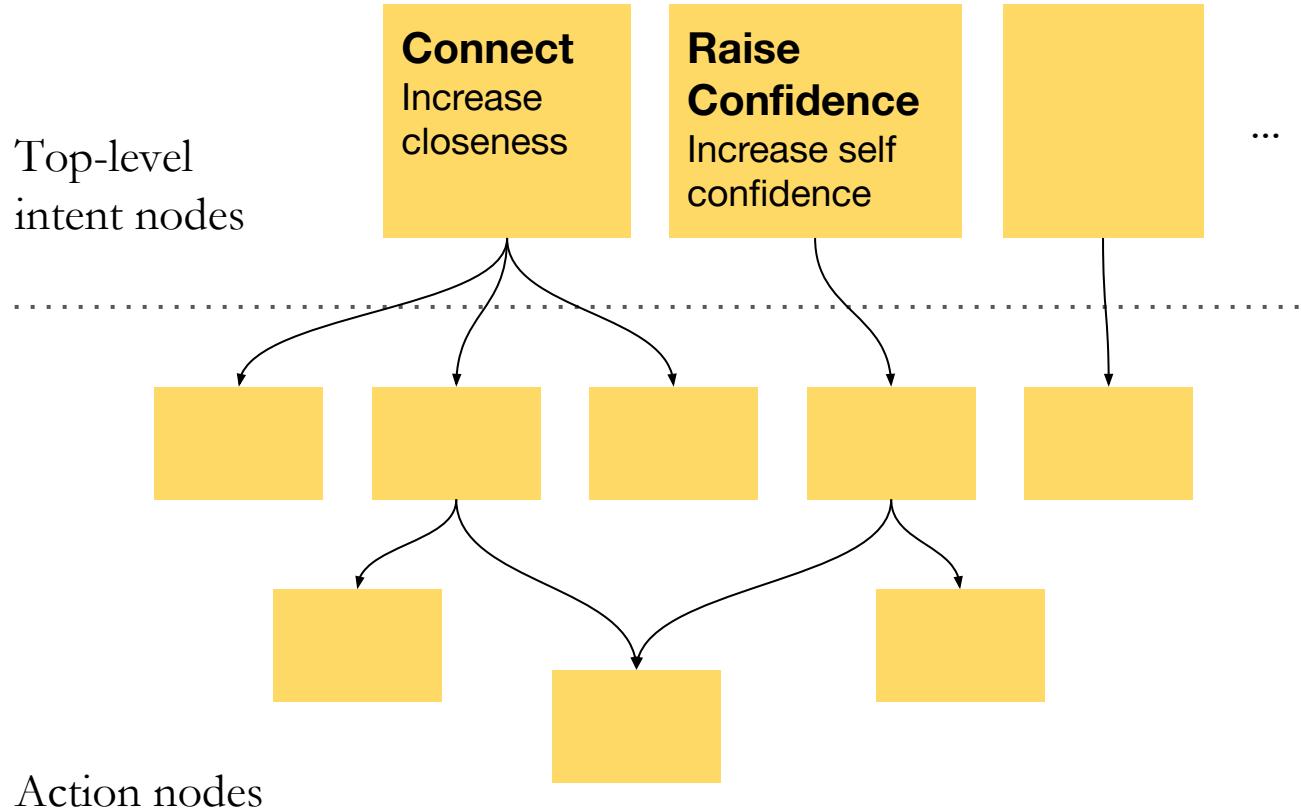
```
"intent"  
"category"  
"type"  
"intentType"  
"first"  
("second")  
"leadsTo"  
"effects"  
"isAccept"
```

# Actions

Ensemble actions have a graph structure.

Start from high-level intents/volitions, which lead to more and more specific actions to fulfill that intent.

Top-level intent nodes

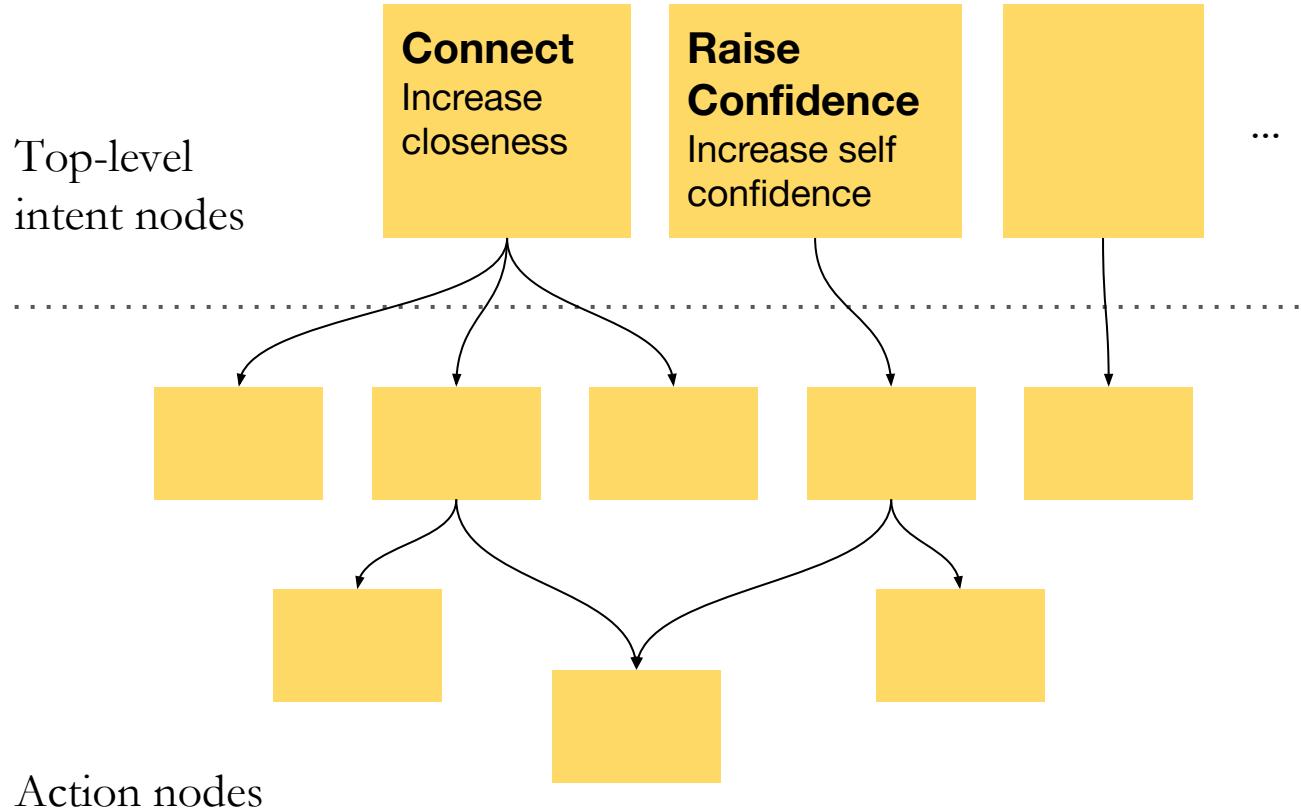


# Actions

Ensemble actions have a graph structure.

- Directed, but not strictly hierarchical
- Many root nodes, corresponding to each volition that appears in volition rules

Top-level intent nodes



# Top-Level Intent Nodes

---

The roots of action graphs are special. They specify what volition the attached nodes are attempting to satisfy.

Top-level intent nodes

**Connect**  
Increase closeness

**Raise Confidence**  
Increase self confidence

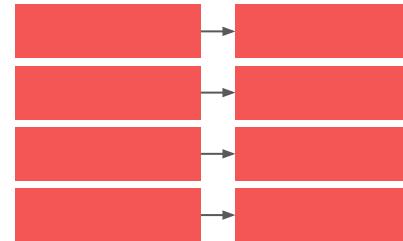
...

# Top-Level Intent Nodes

The roots of action graphs are special. They specify what volition the attached nodes are attempting to satisfy.

They're what connect volition rules to actions.

Volition rules



Top-level intent nodes

**Connect**  
Increase closeness

**Raise Confidence**  
Increase self confidence

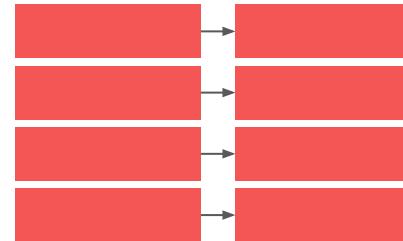
...

# Top-Level Intent Nodes

The roots of action graphs are special. They specify what volition the attached nodes are attempting to satisfy.

They're what connect volition rules to actions.

Volition  
rules



Top-level  
intent  
nodes

**Connect**  
Increase closeness

**Raise Confidence**  
Increase self confidence

...

# Top-Level Intent Nodes

Because of this, they have a special JSON property, that other action nodes do not:

"intent"

**Connect**

Increase closeness

```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

Because of this, they have a special JSON property, that other action nodes do not:

"intent"

Btw, you can't have the same intent in more than one action

**Connect**  
Increase closeness

```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

"intent" object properties

## category

What schema category this action and all its subactions affect

## type

What type within that category that you're referencing

```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

"intent" object properties

## intentType

Whether this action is about attempting to make the schema value increase or decrease (or become true or false, if the category is boolean)

true : increase or make true  
false : decrease or make false



```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

"intent" object properties

## first

Role name of the character who is initiating the action; who has this intent; whose state will change if they perform the action successfully

## second

Role name of the other character involved in the action, if the schema category you're referencing is between two people

```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

What about these?

Required properties on all action nodes, but values can be empty arrays (and probably should be for root actions)

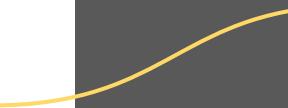
```
{  
  "name" : "connect",  
  "displayName" : "connect",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "intentType" : true,  
      "first" : "initiator",  
      "second" : "responder"  
    },  
  "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

## influenceRules

Array of rules that help determine which action a character might be **more inclined** to perform. Soft condition.

This selection process (with respect to top-level intent actions) is the whole point of volition rules, so probably shouldn't have any influence rules or conditions in intent-level nodes.



```
{  
    "name" : "connect",  
    "displayName" : "connect",  
    "influenceRules" : [],  
    "conditions" : [],  
    "intent" :  
        {  
            "category" : "relationship",  
            "type" : "closeness",  
            "intentType" : true,  
            "first" : "initiator",  
            "second" : "responder"  
        },  
    "leadsTo" :  
        ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

## conditions

Array of hard preconditions that must hold true for this action to be considered.

Again, usually no conditions for top-level action nodes.



```
{  
    "name" : "connect",  
    "displayName" : "connect",  
    "influenceRules" : [],  
    "conditions" : [],  
    "intent" :  
    {  
        "category" : "relationship",  
        "type" : "closeness",  
        "intentType" : true,  
        "first" : "initiator",  
        "second" : "responder"  
    },  
    "leadsTo" :  
    ["chat", "makeLaugh", "confideIn"]  
}
```

# Top-Level Intent Nodes

Intent level action nodes can't be the only nodes in your action graph.

Top-level intent nodes

**Connect**

Increase closeness

**Raise Confidence**

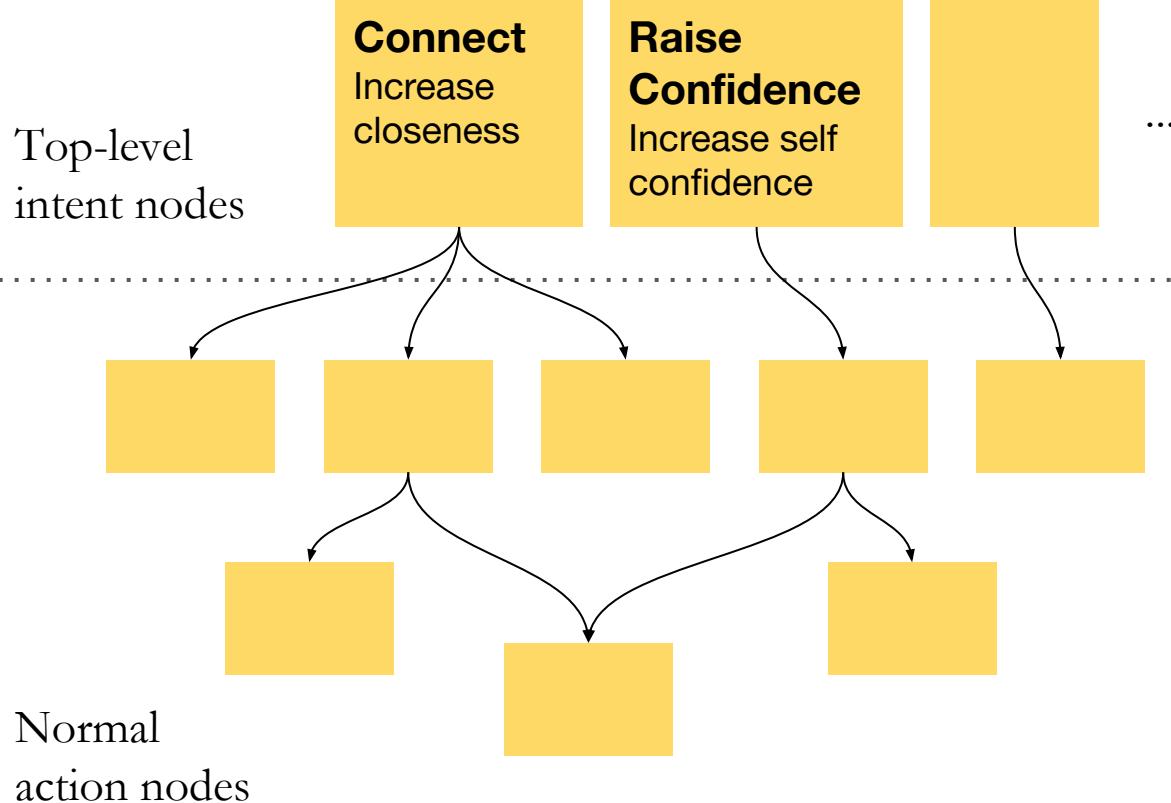
Increase self confidence

...

# Top-Level Intent Nodes

Intent level action nodes can't be the only nodes in your action graph.

Need to lead to other nodes, which actually affect the world.



# Top-Level Intent Node Example

So this action node is saying the "connect" action (and its three subactions) is something a character can do if they have high volition to increase the closeness between them and another character.

It also says that `chat`, `makeLaugh`, and `confideIn` are all more specific actions that fulfill this connect intent.

The diagram shows a JSON object structure. A yellow box highlights the `"leadsTo"` field, which contains an array of three actions: `["chat", "makeLaugh", "confideIn"]`. Two yellow arrows point from the text above to the yellow box: one points to the `"leadsTo"` field, and another points to the closing brace of the JSON object.

```
{
  "name" : "connect",
  "displayName" : "connect",
  "influenceRules" : [],
  "conditions" : [],
  "intent" :
  {
    "category" : "relationship",
    "type" : "closeness",
    "intentType" : true,
    "first" : "initiator",
    "second" : "responder"
  },
  "leadsTo" :
  [
    "chat",
    "makeLaugh",
    "confideIn"
  ]
}
```

# Top-Level Intent Node Example

This one is saying the "escape" action (and its one subaction) is something a character can do if they have high volition to make their own alone status true.

In this social world, there is only one specific action this intent leads to.

```
{  
  "name" : "escape",  
  "displayName" : "escape",  
  "influenceRules" : [],  
  "conditions" : [],  
  "intent" :  
    {  
      "category" : "status",  
      "type" : "alone",  
      "intentType" : true,  
      "first" : "initiator"  
    },  
  "leadsTo" : ["runAway"]  
}
```

# Action Node Properties

## Always required

- ✓ "name"
- ✓ "influenceRules"
- ✓ "conditions"

## Sometimes required,

Depending on where the node  
is in an action graph

- ✓ "intent"  
"category"  
"type"  
"intentType"  
"first"  
("second")
- ✓ "leadsTo"  
"effects"  
"isAccept"

# Action Node Properties

## Always required

- ✓ "name"
- ✓ "influenceRules"
- ✓ "conditions"

## Sometimes required,

Depending on where the node  
is in an action graph

- ✓ "intent"
- "category"
- "type"
- "intentType"
- "first"
- ("second")
- ✓ "leadsTo"
- "effects"
- "isAccept"

# Actions

Connecting intent-level nodes to other, more concrete action nodes

Top-level intent node

Action nodes

**Connect**  
Increase closeness

**Chat**  
If you don't know someone that well



# Non-root action node

With an influence rule  
and one effect

```
{  
    "name" : "chat",  
    "conditions" : [],  
    "influenceRules" : [  
        {  
            "name": "Chatting is a safe thing to do  
                    with people you don't know too well.",  
            "conditions": [  
                {  
                    "category": "relationship",  
                    "type": "closeness",  
                    "first": "initiator",  
                    "second": "responder",  
                    "operator": "<",  
                    "value": 5  
                }],  
                "weight": 3  
            ]],  
            "effects" : [  
                {  
                    "category" : "relationship",  
                    "type" : "closeness",  
                    "first" : "initiator",  
                    "second" : "responder",  
                    "operator" : "+",  
                    "value" : 1  
                }]  
    }  
}
```

# Non-root action node

## influenceRules

Array of rules that help determine which action a character might be **more inclined** to perform. Soft condition.

Optional but should include either influenceRule or condition if this node has siblings in the action graph

```
{  
  "name" : "chat",  
  "conditions" : [],  
  "influenceRules" : [{  
    "name": "Chatting is a safe thing to do  
           with people you don't know too well.",  
    "conditions": [  
      {"category": "relationship",  
       "type": "closeness",  
       "first": "initiator",  
       "second": "responder",  
       "operator": "<",  
       "value": 5  
    ],  
    "weight": 3  
  ],  
  "effects" : [  
    {"category" : "relationship",  
     "type" : "closeness",  
     "first" : "initiator",  
     "second" : "responder",  
     "operator" : "+",  
     "value" : 1  
  ]  
}
```

# Non-root action node

## effects

Array of predicates specifying how carrying out the action should affect the state of the world.

```
{  
    "name" : "chat",  
    "conditions" : [],  
    "influenceRules" : [  
        {  
            "name": "Chatting is a safe thing to do  
                    with people you don't know too well.",  
            "conditions": [  
                {  
                    "category": "relationship",  
                    "type": "closeness",  
                    "first": "initiator",  
                    "second": "responder",  
                    "operator": "<",  
                    "value": 5  
                }],  
                "weight": 3  
            ],  
            "effects" : [  
                {  
                    "category" : "relationship",  
                    "type" : "closeness",  
                    "first" : "initiator",  
                    "second" : "responder",  
                    "operator" : "+",  
                    "value" : 1  
                }]  
        }]
```

# Non-root action node

Only difference between action effects and volition effects:

```
{  
  "category": "level",  
  "type": "energy",  
  "first": "x",  
  "weight": 3,  
  "intentType": true,  
}
```

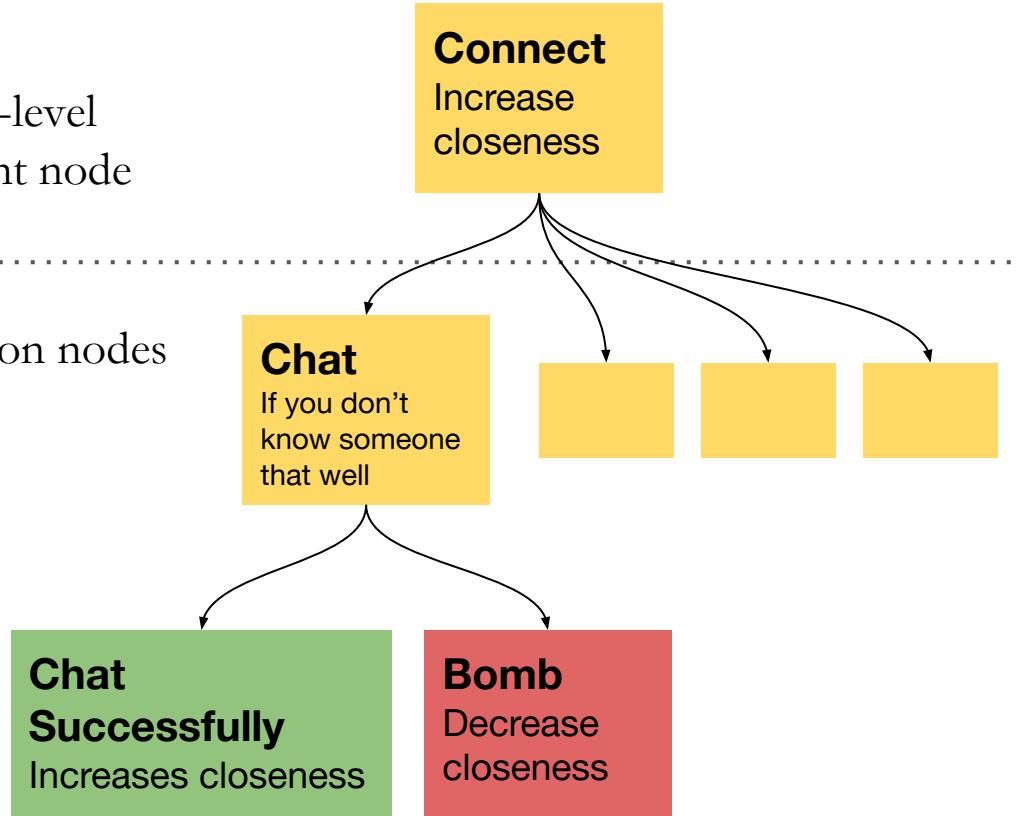
```
{  
  "name" : "chat",  
  "conditions" : [],  
  "influenceRules" : [  
    {  
      "name": "Chatting is a safe thing to do  
              with people you don't know too well.",  
      "conditions": [  
        {  
          "category": "relationship",  
          "type": "closeness",  
          "first": "initiator",  
          "second": "responder",  
          "operator": "<",  
          "value": 5  
        }],  
      "weight": 3  
    }],  
  "effects" : [  
    {  
      "category" : "relationship",  
      "type" : "closeness",  
      "first" : "initiator",  
      "second" : "responder",  
      "operator" : "+",  
      "value" : 1  
    }]  
}
```

# Acceptance and Rejection

In Ensemble, actions are often directed toward another character. So there's a notion of the responding character either responding positively or negatively.

Top-level intent node

Action nodes



# isAccept

## isAccept

Boolean. Specifies whether this is an action that should be played when the responder accepts (true) or rejects (false). True by default.

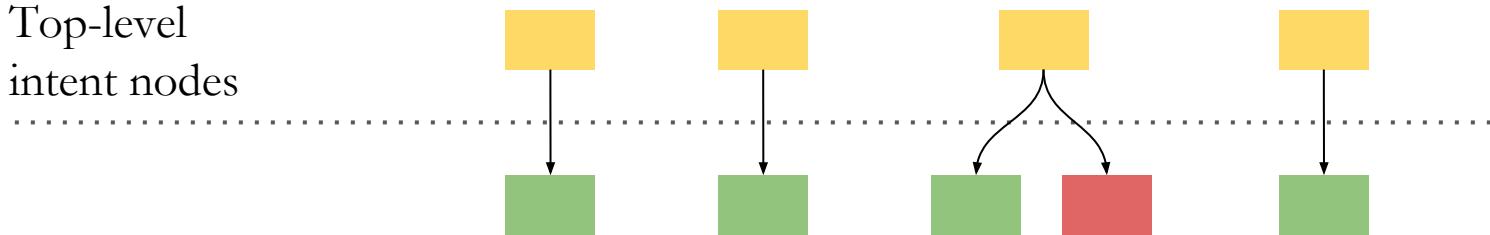
If the responding character would have positive volition to do the same action if the roles were reversed, then they will “accept”, otherwise “reject”

```
{  
    "name" : "chat",  
    "conditions" : [],  
    "influenceRules" : [  
        {"name": "Chatting is a safe thing to do  
            with people you don't know too well.",  
        "conditions": [{  
            "category": "relationship",  
            "type": "closeness",  
            "first": "initiator",  
            "second": "responder",  
            "operator": "<",  
            "value": 5  
        }],  
        "weight": 3  
    ],  
    "effects" : [  
        {"category" : "relationship",  
        "type" : "closeness",  
        "first" : "initiator",  
        "second" : "responder",  
        "operator" : "+",  
        "value" : 1  
    ]],  
    "isAccept": true  
}
```

Many different flavors or ways of organizing your action graph(s)

## Simplest Action Structure

Top-level intent nodes



- Only one level of `leadsTo`
- Mostly `isAccept` true (default), but maybe some rejections sprinkled in

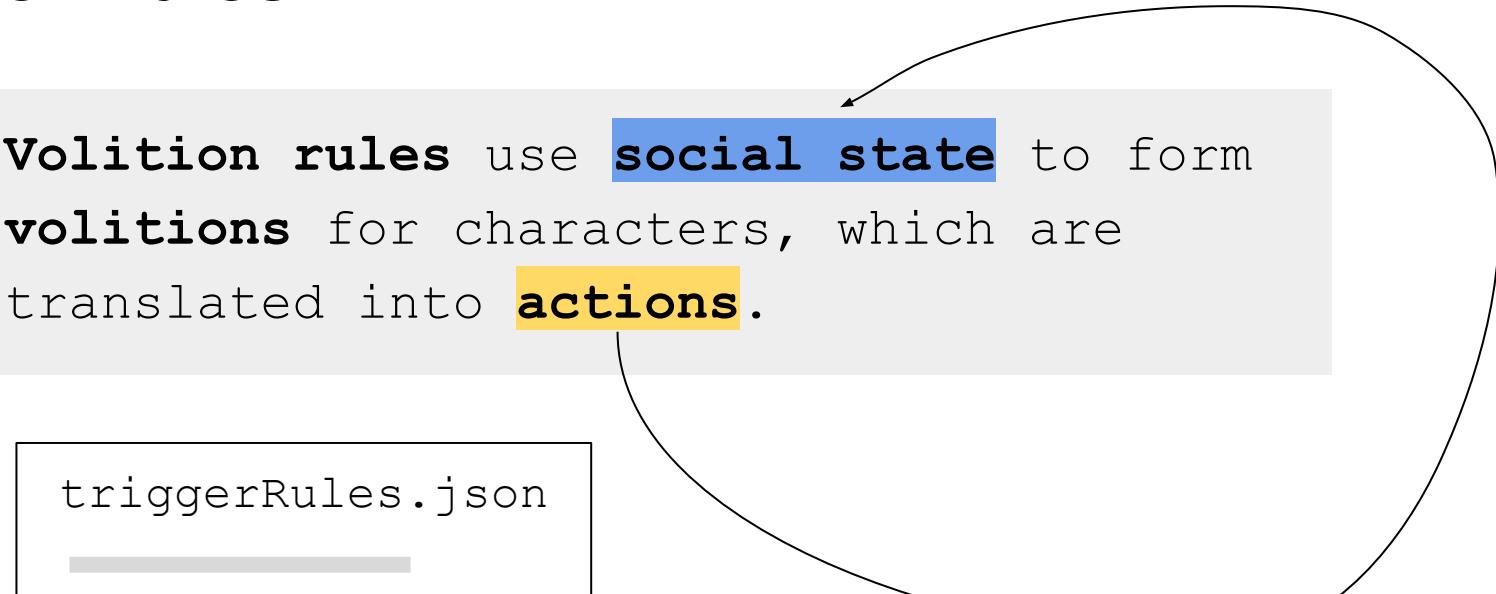
# Now we'll create cascading effects from characters taking action in the world

Volition rules use **social state** to form **volitions** for characters, which are translated into **actions**.

# Trigger Rules

**Volition rules** use **social state** to form **volitions** for characters, which are translated into **actions**.

triggerRules.json



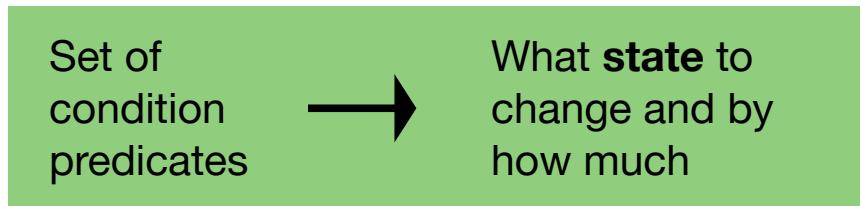
```
[{"id": 1, "text": "Volition rules use social state to form volitions for characters, which are translated into actions."}, {"id": 2, "text": "triggerRules.json"}]
```

# Trigger Rules

Similar to volition rules:



But instead of affecting internal character desires and going through actions to change the state of the world...



...directly affect the state of the world

# TriggerRules.json

Rules that check conditions against current state and then immediately modify that state

**Same kind of conditions and effects as actions**

```
{"name": "Introverts constantly lose energy when they are with people",
"conditions": [
    {
        "category": "trait",
        "type": "introverted",
        "first": "introvert",
        "value": true
    },
    {
        "category": "status",
        "type": "alone",
        "first": "introvert",
        "value": false
    }
],
"effects": [
    {
        "category": "level",
        "type": "energy",
        "first": "introvert",
        "value": 1,
        "operator": "-"
    }
]
}
```

# Trigger Rules

**Trigger Rule** Name: Introverts constantly lose energy when they are with people

*When this is true:*

[introvert] is introverted . (L) X

[introvert] is not alone . (L) X

**Add Condition**

→ *Make this true:*

[introvert] has 1 less energy .

**Add Effect**

**That covers the major concepts**

Social State

History

Character Desires

Actions

Trigger Rules

# Ensemble Engine

[github.com/ensemble-engine/ensemble](https://github.com/ensemble-engine/ensemble)

- + See the [releases page](#) for the latest downloadables
- + The [issues](#) page for known bugs
- + The [README](#) for how to build the standalone JS library and the tool
- + [Eventually] the [wiki](#) for more documentation and learning materials

# See also this Ensemble Data Objects Quick Reference

Ensemble Objects Quick Reference															
Hand-Authored Definitions															
JSON Object	Cast	Schema	VolitionRules		TriggerRules		Actions		History		System-Generated Data				
JSON Properties and expected values or types	name	String	category	String	name	String	name	String	pos	number	category	String (from Schema)	name	String (from Actions)	
	types	[String]	conditions	[condition] ↓	conditions	[condition] ↓	intent	intent ↓	data	[predicate] ↓	type	String (from Schema)	filename	String	
	directionType	"directed", "undirected", "reciprocal"	effects	[effect] ↓	effects	[effect] ↓	conditions	[condition] ↓			intentType	boolean	conditions	[condition] ↓	
<b>Legend:</b>	isBoolean	boolean									first	String (from Cast)	influenceRules	[influenceRule] ↓	
Required property	actionable	boolean									second	String (from Cast)	effects	[effect] ↓	
Optional property	defaultValue	boolean, number									englishInfluences	[englishInfluence] ↓	isAccept	boolean	
↓ = Def'd below	minValue	number									weight	number	lineage	String	
, = or	maxValue	number									goodBindings	[binding] ↓	weight	number	
	duration	number									salience	number			
<b>Sub-Object</b>			Condition		Condition		Condition		Predicate		Calculated Influence				
Properties			category	String (from Schema)	category	String (from Schema)	category	String (from Schema)	category	String (from Schema)	englishRule	String	category	String (from Schema)	
			type	String (from Schema)	type	String (from Schema)	type	String (from Schema)	type	String (from Schema)	origin	String	type	String (from Schema)	
			first	String (from Cast)	first	String (from Cast)	first	String (from Cast)	first	String (from Cast)	ruleName	String (from Volitions)	first	String (from Cast)	
			second	String (from Cast)	second	String (from Cast)	second	String (from Cast)	second	String (from Cast)	second	String (from Cast)	second	String (from Cast)	
			operator	>, <, >=, <=, ==	operator	>, <, >=, <=, ==	operator	>, <, >=, <=, ==	value	boolean, number	operator	>, <, >=, <=, ==	value	boolean, number	
			value	boolean, number	value	boolean, number	value	boolean, number			value	boolean, number	operator	>, <, >=, <=, ==	
			<b>Effect</b>	<b>Effect</b>	<b>Effect</b>	<b>Effect</b>	<b>Effect</b>	<b>Effect</b>					<b>Effect</b>		
			category	String (from Schema)	category	String (from Schema)	category	String (from Schema)	category	String (from Schema)	category	String (from Schema)	type	String (from Schema)	
			type	String (from Schema)	type	String (from Schema)	type	String (from Schema)	type	String (from Schema)	intentType	boolean	intentType	boolean	
			intentType	boolean	intentType	boolean	intentType	boolean	first	String (from Cast)	first	String (from Cast)	first	String (from Cast)	
			first	String (from Cast)	first	String (from Cast)	first	String (from Cast)	value	boolean, number	value	boolean, number	value	boolean, number	
			second	String (from Cast)	second	String (from Cast)	second	String (from Cast)	operator	"+", "-"	operator	"+", "-"	operator	"+", "-"	
			weight	number	value	boolean, number	operator	"+", "-"							
FYI, the authoring tool ignores some fields and values for file I/O (e.g., origin, id). Those are safe to ignore.															

@meldckn

# Caveats & Common Issues

Your JSON isn't valid JSON. (Project won't load in editor)

- Edit in a JSON-highlighting text editor
- Copy-paste your whole file into something like [jsonlint.com](http://jsonlint.com)

Unexpected behavior from either the tool or Ensemble core

- Open the dev console in the tool, look for error messages
- Maybe search the repo's [issues](#) to see if it's a known bug (the issue may describe a workaround)

You don't know what the authoring tool console commands do

- There are ample tooltips (hover over things)

Open a new [issue](#) (perhaps using 'bug' or 'question' tags)  
or email me at [meldckn@gmail.com](mailto:meldckn@gmail.com)!

More info, but in academic paper form:

[aaronareed.net/2015/Ensemble-Engine-FDG2015.pdf](http://aaronareed.net/2015/Ensemble-Engine-FDG2015.pdf)

## The Ensemble Engine: Next-Generation Social Physics

Ben Samuel, Aaron A. Reed, Paul Maddaloni, Michael Matasa, Noah Wardrip-Fruin  
University of California Santa Cruz, Emerging Intelligence Studio  
(bsamuel, aaronr, 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, the first public version of the Ensemble physics engine. The Ensemble Engine is inspired by the lessons learned from more than five years building the *Conway's Game of Life* [1] and *Primed* [2] engines, and from the game *Prism* [3] (including *From World*). The Ensemble Engine retains the most successful aspects of those engines while making major improvements in areas such as the flexibility of rules, the expressiveness and expressivity of its rules. The system is authored in an open-source, modular, and extensible way, designed to facilitate reuse and increase accessibility for game researchers and creators. Through these improvements and a distributed strategy, the Ensemble Engine represents an opportunity for a fundamental shift in social physics to become much more broadly explored.

**Categories and Subject Descriptors** D.2.10 [Artificial Intelligence]: Knowledge Representation—Formation and Methods—Representations (procedural and rule-based).

### General Terms

### Design

### Keywords

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

### 1. INTRODUCTION

Social dynamics 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 Witcher 3* [1]), puzzle games (e.g., *Angry Birds* [2]), first-person shooters (e.g., *Call of Duty* [20]), and even survival games (e.g., *State of Decay* [24]). Unfortunately, players can only experience these dynamics through cutscenes, which usually change at fixed, predetermined progression points in the game, or through other systems within the game such as combat, allowing the player to directly act and agency over them.

Some games have provided somewhat dynamic social relationships. For example, some role-playing games let the player change their gender and race, and some RPGs allow the player to have sex with non-player characters, giving the possibility of character-specific

concepts such as quests, battles, and romances [1]. But these systems are simple and limited. They are often static, such that players are not engaged in the same series, for example, they are with same game's combat system. In contrast, the Ensemble Engine is a sample of playable social interaction in *The Sims* [22], which is a very successful series, though rarely effectively imitated. But *The Sims* is also quite limited in the potential to have meaningful social interactions. It is limited by the language of the game, the language of the game's interface, the language of the game's user interface, the meaningfulness of its actions, and so on. As popular as *The Sims* is, it is not a game that is widely played. *Prism* [3] and *From World* [20] present the illusion that characters will have richer memories and more complex social interactions. But the game *Prism* [3] shows that these dynamics ultimately never strengthen because the game's physics are too rigid.

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 by computational representations of social systems [13, 14]. This field makes dynamic social interactions to become the centerpiece of a game, and for the player to have agency over them. These systems are often called "socially represented". The two primary examples of social physics engines as far as we are concerned are the Ensemble Engine [1] and the *Prism* system [3]. The Ensemble Engine is designed to be used to create *From World* [14], *Prism* [3], and *Twine* [21], a game for the *GREEN* conflict resolution project [27], among other playable stories. The Ensemble Engine is a general system for creating social comedy of manners, a set of modern office comedy stories, and ambient media, and *Twine* [21].

Up until now, these systems have only been available to game 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 and to prevent social physics gameplay from being combined with other types of game mechanics. The Ensemble Engine is a freely-available social physics engine<sup>1</sup>. The design of the system is intended to be accessible to anyone who has had less than five years of building the *CGP* [18] or *Prism* [3]. The Ensemble Engine is designed to be used to create a wide variety of games. Put simply, a social physics system looks at all of the social interactions in a game and uses them to determine what happens and determines how the characters might best react to the current social state to save their stories, as well as directly changing the social state itself (as appropriate). *Prism*'s notable feature—in social physics—is that it is procedural.

<sup>1</sup>The Ensemble Engine is based on the *Prism* engine at the time of this writing. The source code of the engine can be found at <https://github.com/soe-ucsc/ensemble-engine>.