Ensemble | Engine for Social Simulation

Melanie Dickinson

Table of Contents

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

---

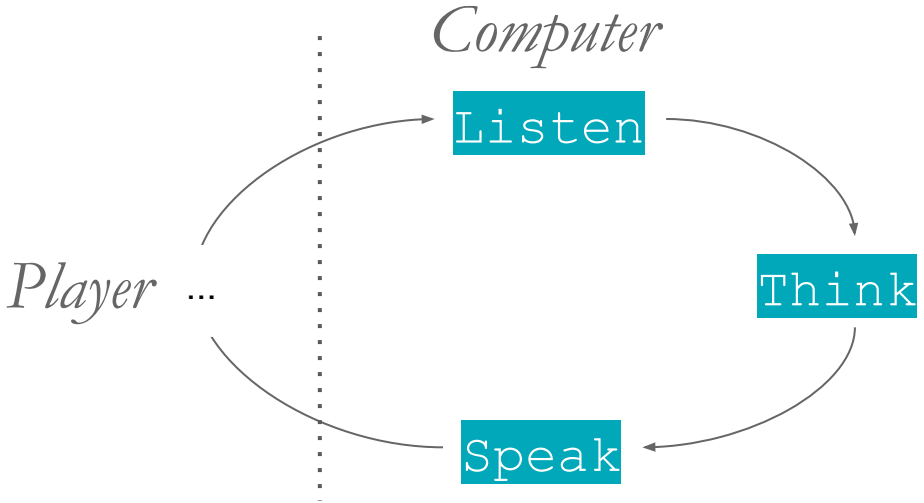
But why?

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

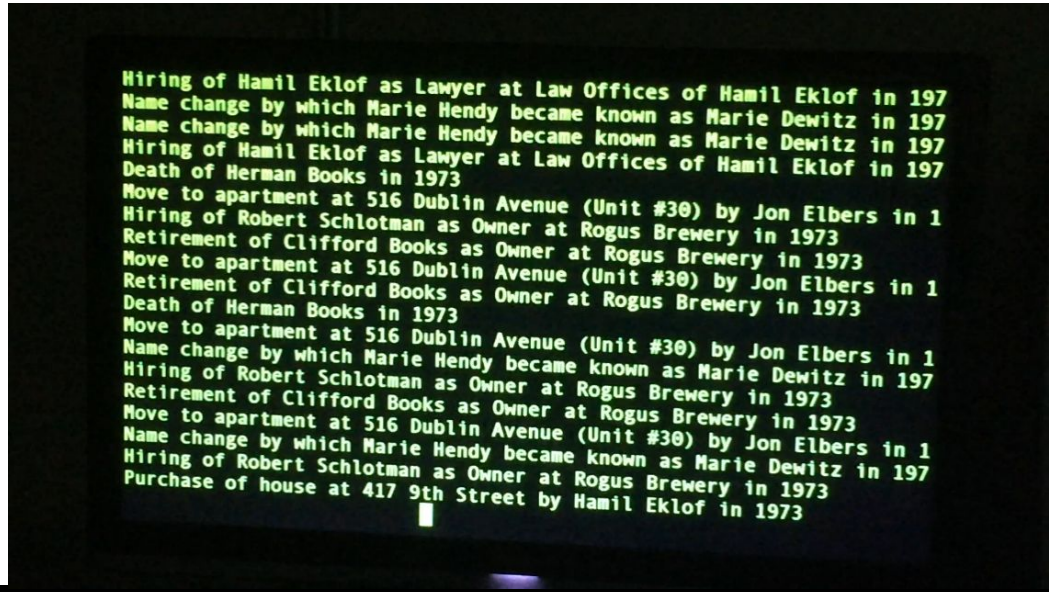
⬇️ *Stardew Valley* — small town chats

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



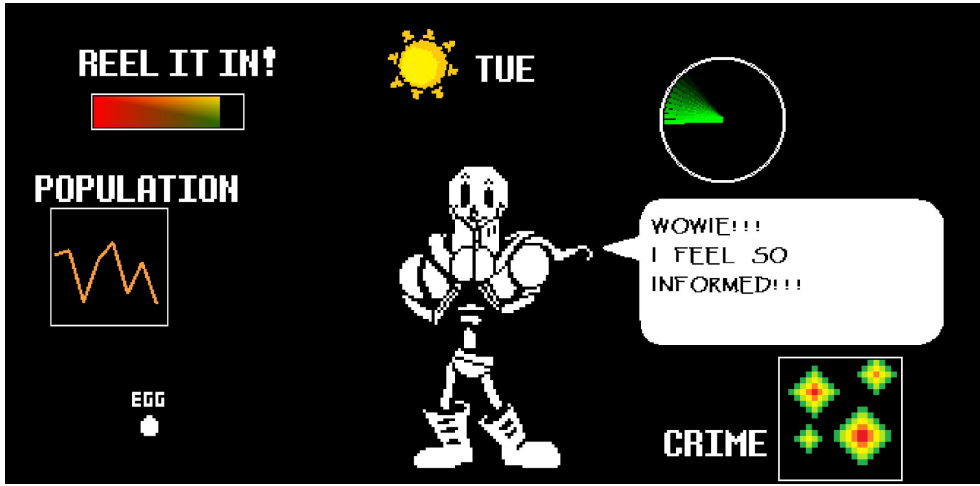
It lets you talk about things that are meaningful to everyday life and activate social pleasures

E.g.,

The feeling of hanging out with good friends

Feeling enmeshed in a community

Interacting with compelling characters



⬆️ *Undertale* — dating minigame

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 ➡️

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

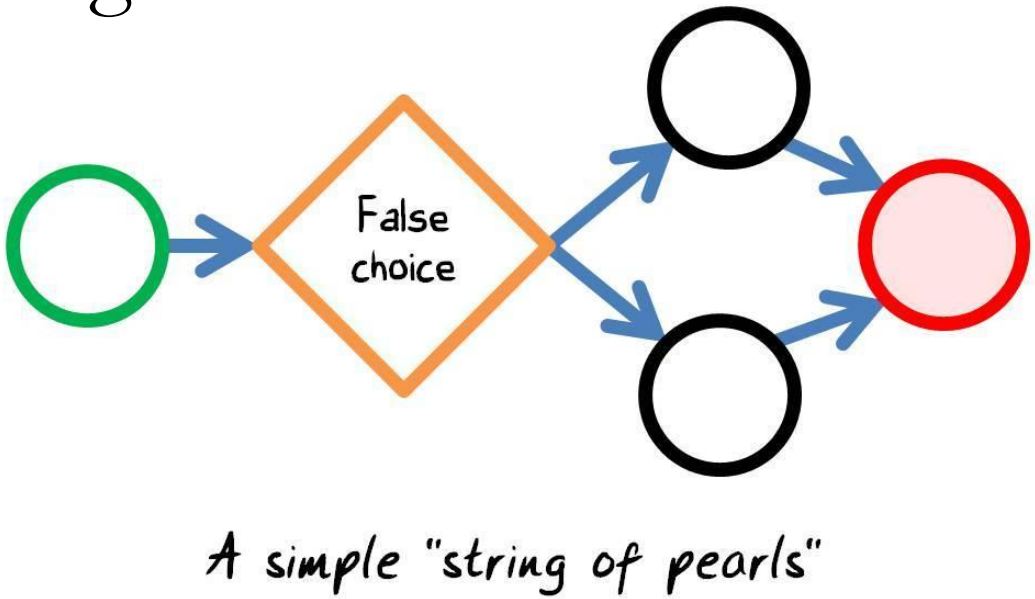
E.g., societal violence & blending in, terrorism, activist organizing, self-segregation ⬆️

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

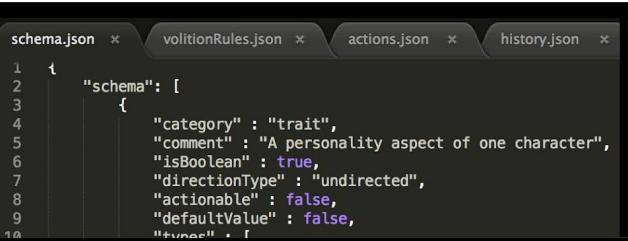
A JSON-embedded language in which you define a social world

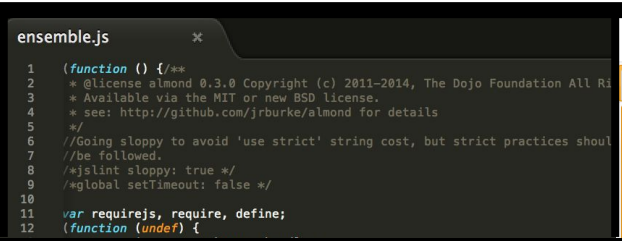
A Javascript library

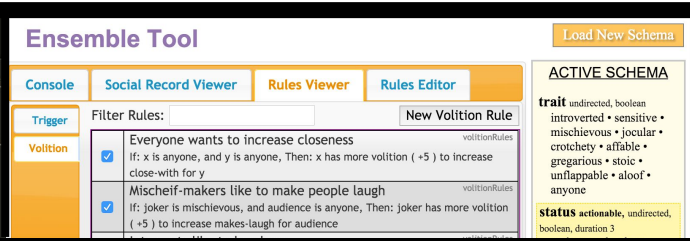
Whose functions you call within the main code of a game or simulation. Calculates character desires, actions, and effects based on your social world

An authoring tool

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



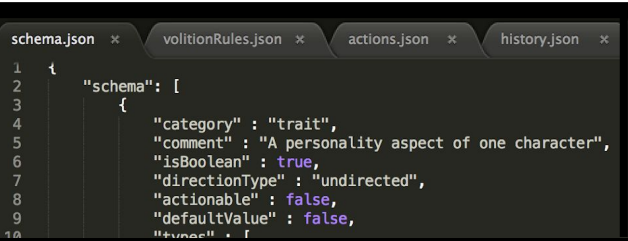


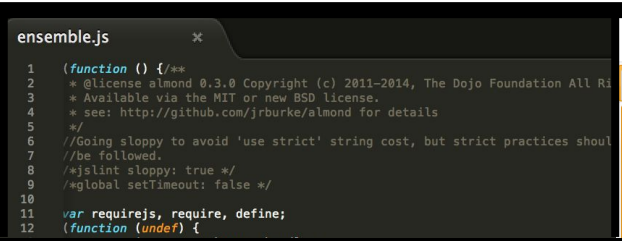


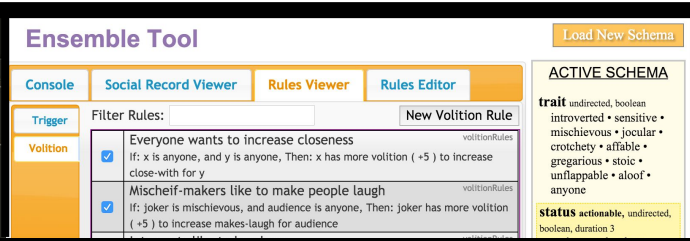
Ensemble can be found on...

github.com/ensemble-engine/ensemble

+ See the releases page for downloadables







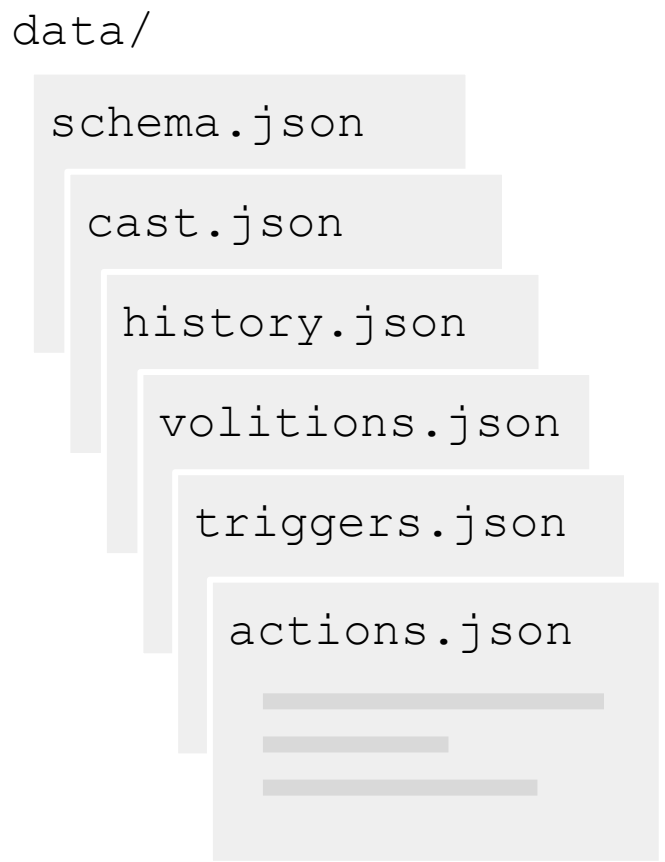
HOW TO USE ENSEMBLE

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`



HOW TO USE IT

2. Call Ensemble functions

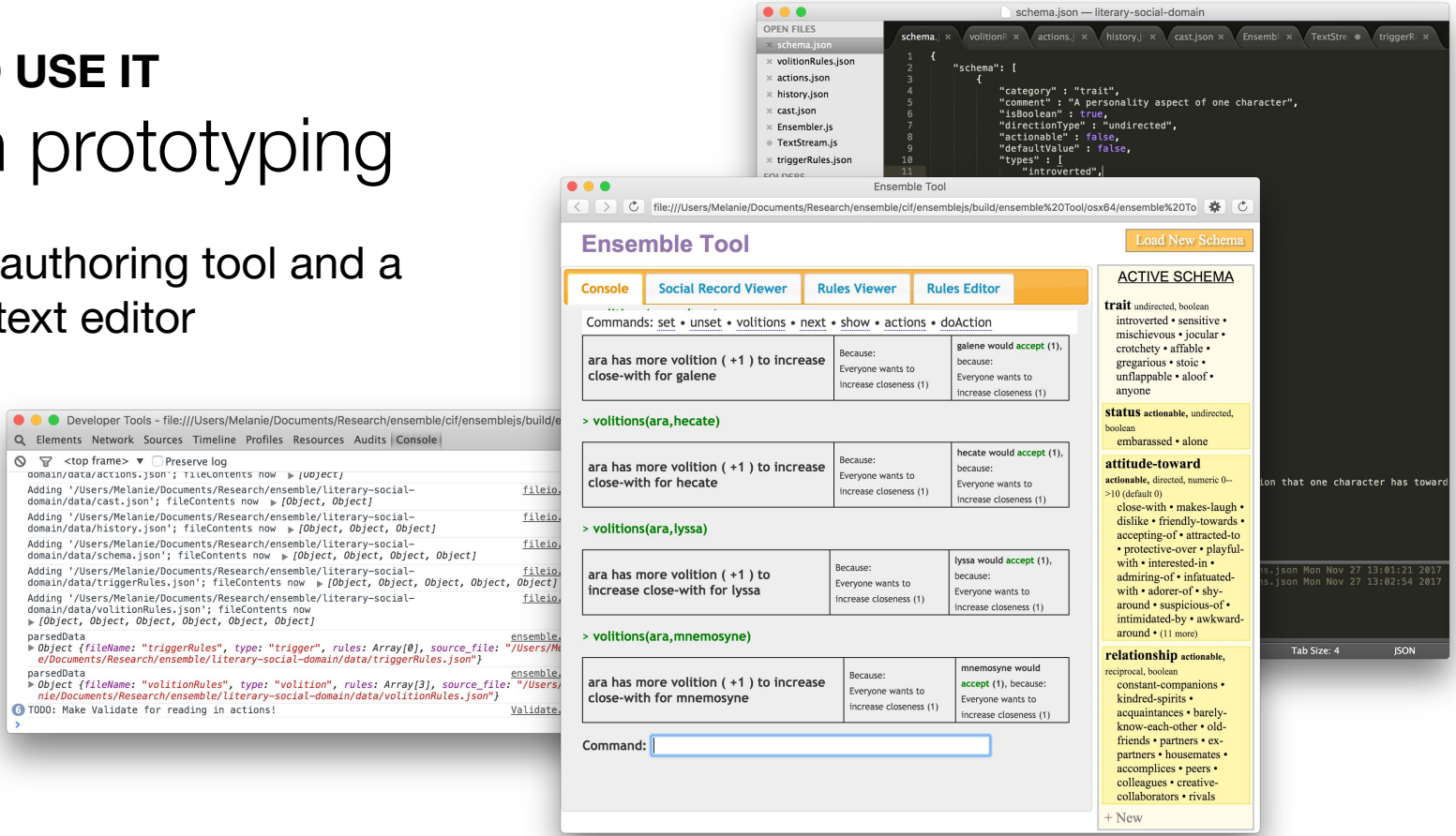
`calculateVolitions`	:	see what volitions a character has
`getActions`	:	see what actions a character wants to do
`set`	:	change social state
`get`	:	get social state
`runTriggerRules`	:	apply rules that conditionally change social state
`dumpSocialRecord`	:	print all history to the console
`setNextTimeStep`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

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

HOW TO USE IT

When prototyping

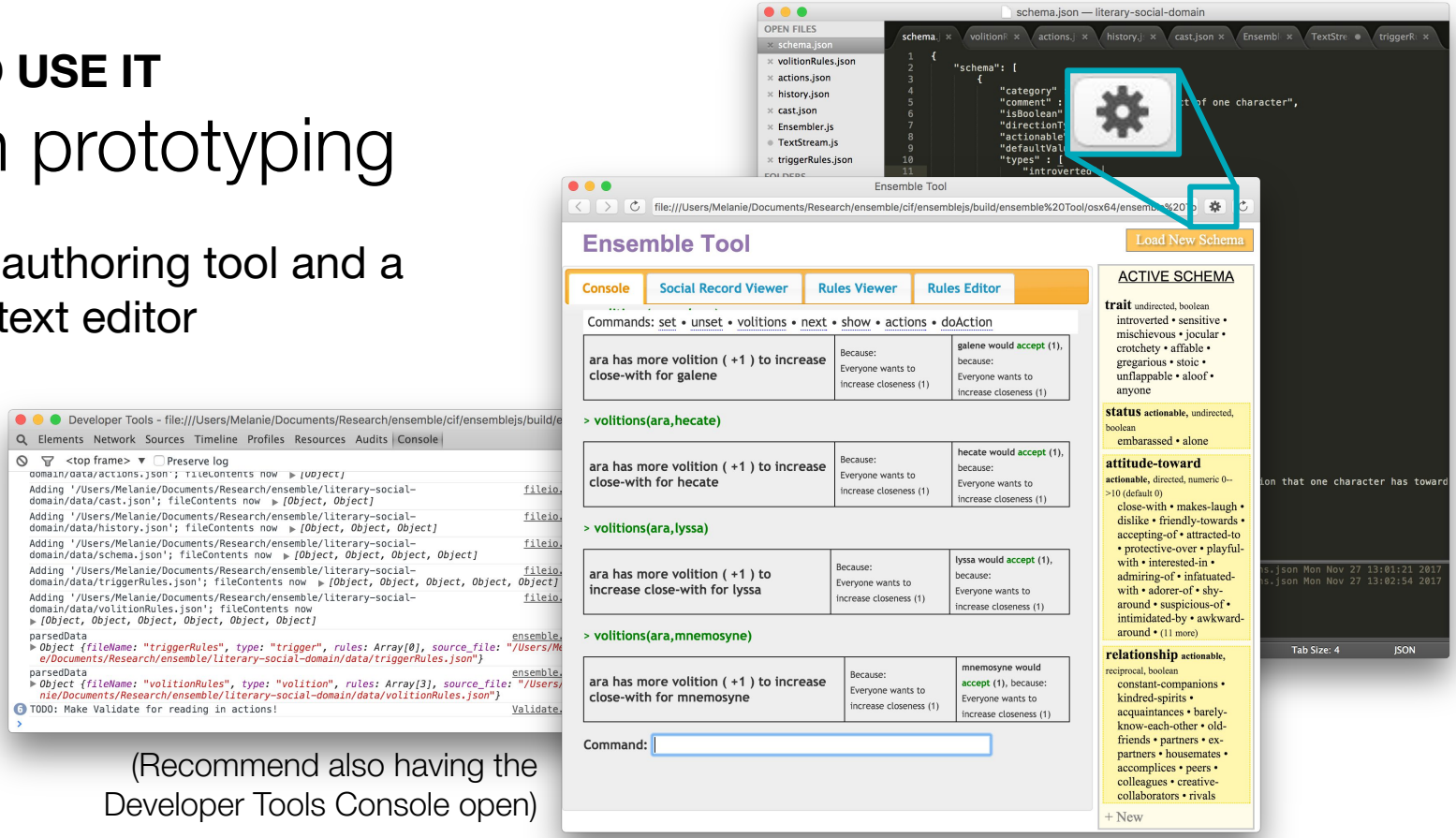
Use the authoring tool and a favorite text editor



HOW TO USE IT

When prototyping

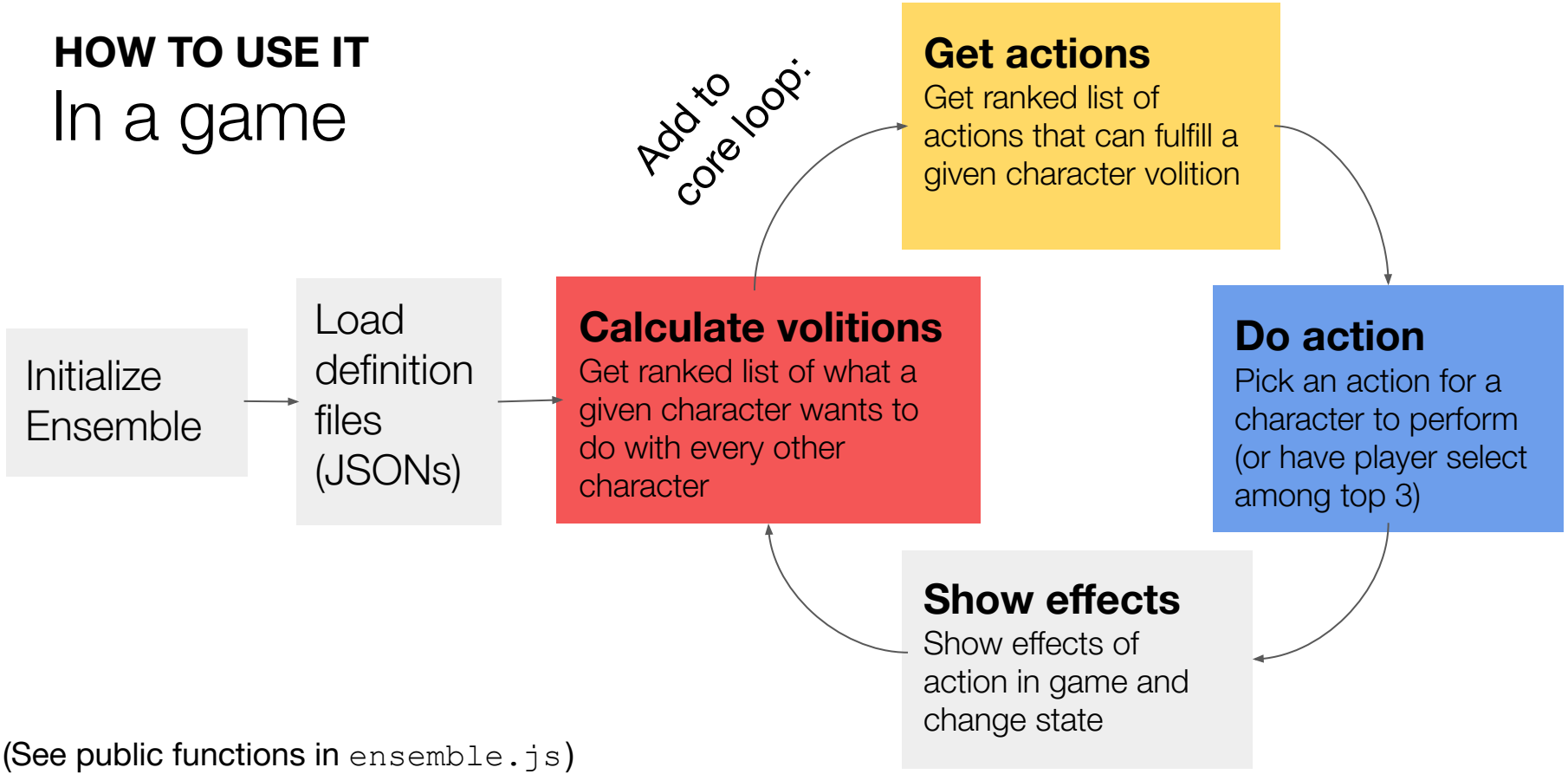
Use the authoring tool and a favorite text editor



(Recommend also having the Developer Tools Console open)

HOW TO USE IT

In a game



(See public functions in `ensemble.js`)

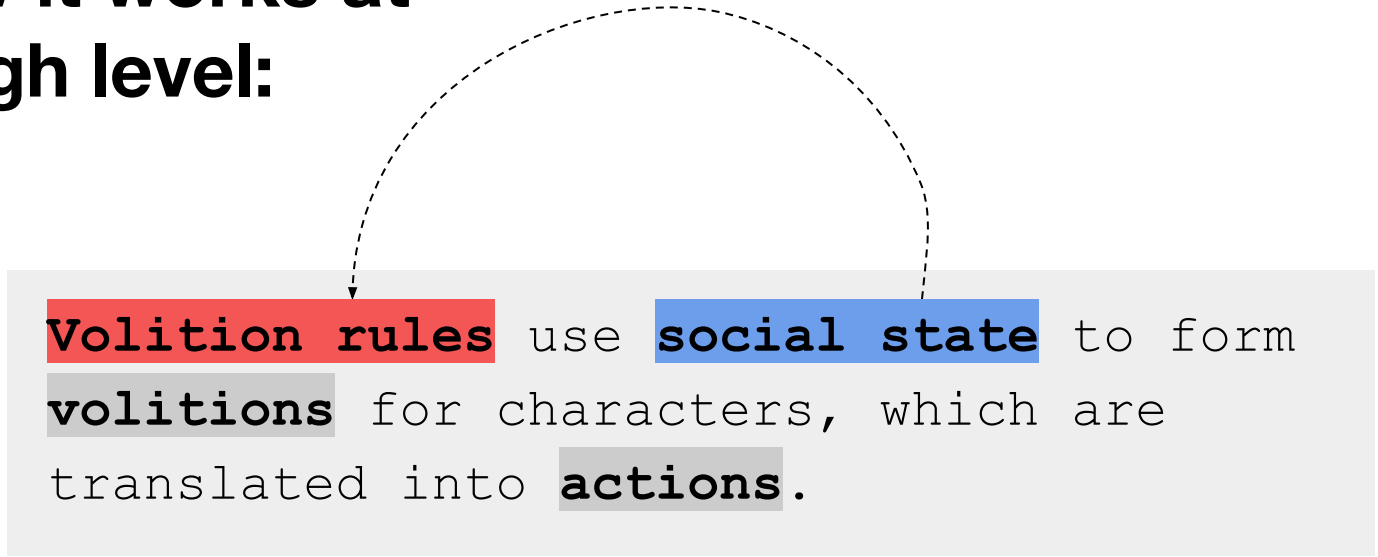
How Ensemble works at a high level:

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

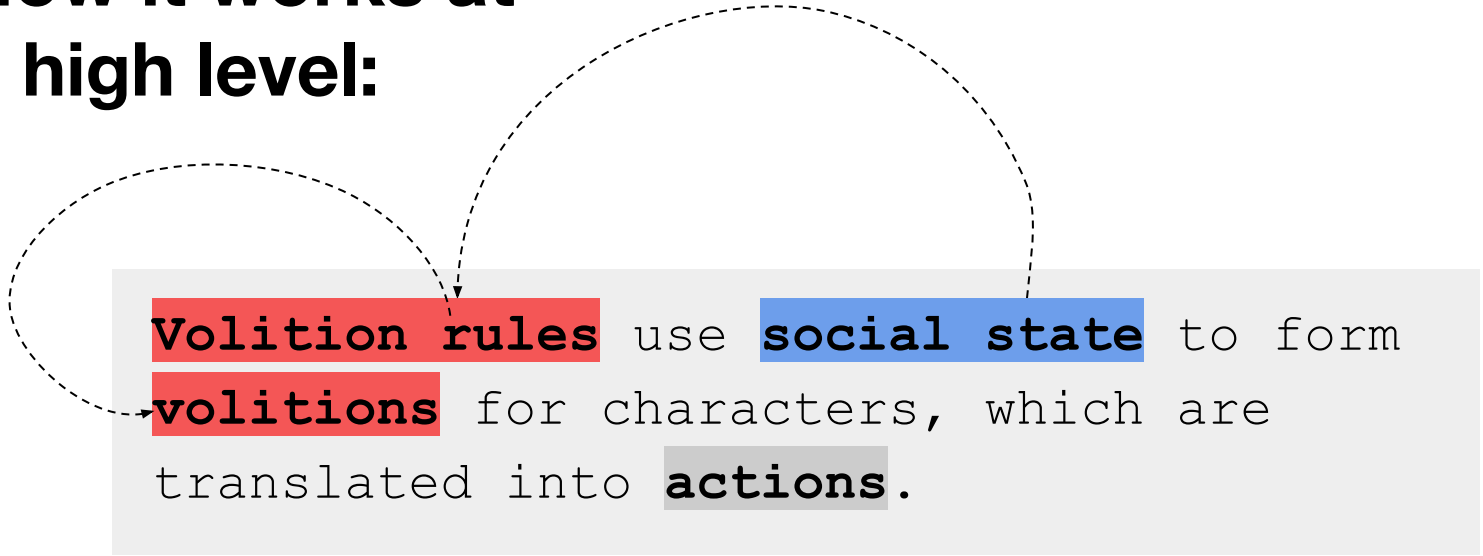
How it works at a high level:

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

How it works at a high level:

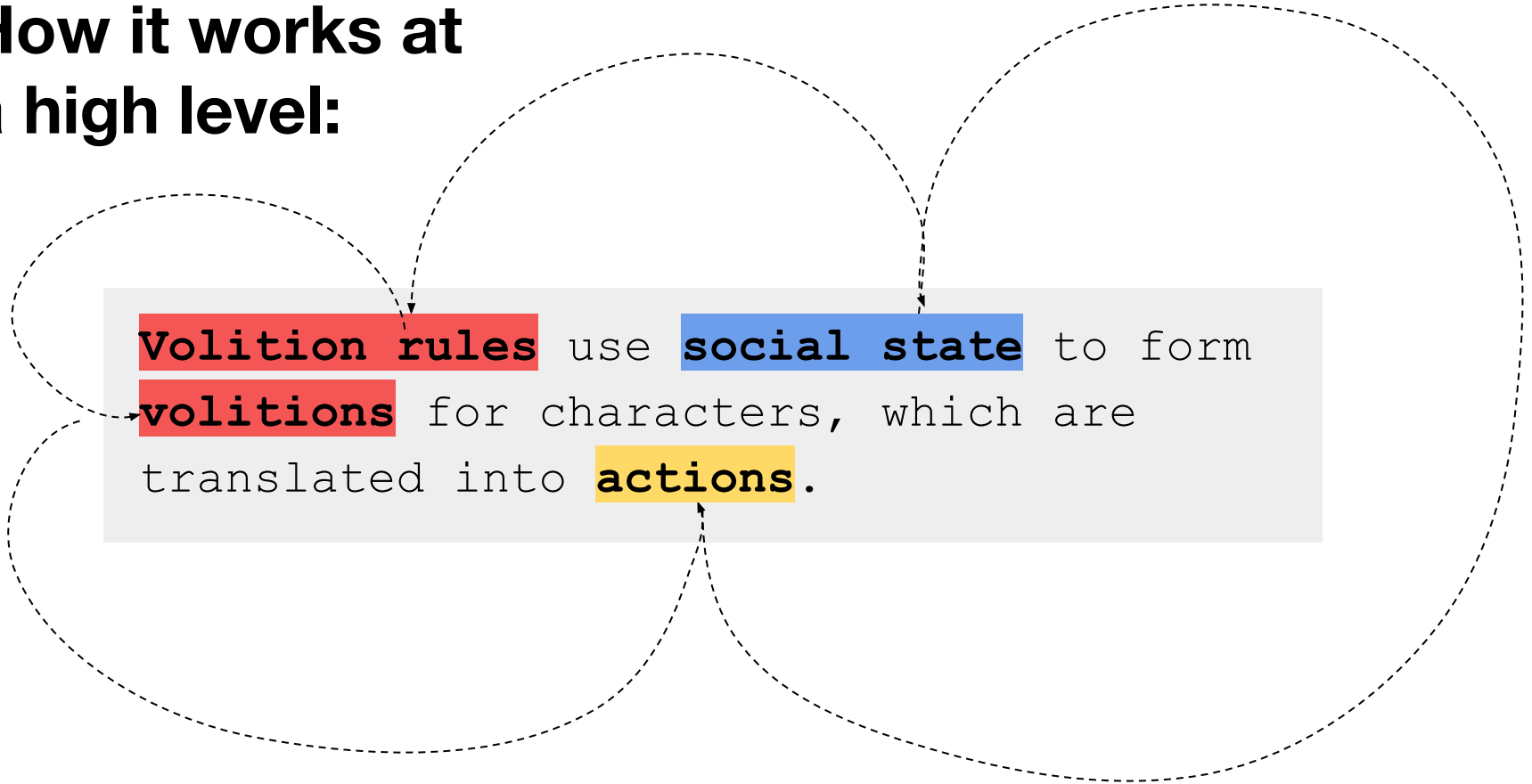


How it works at a high level:

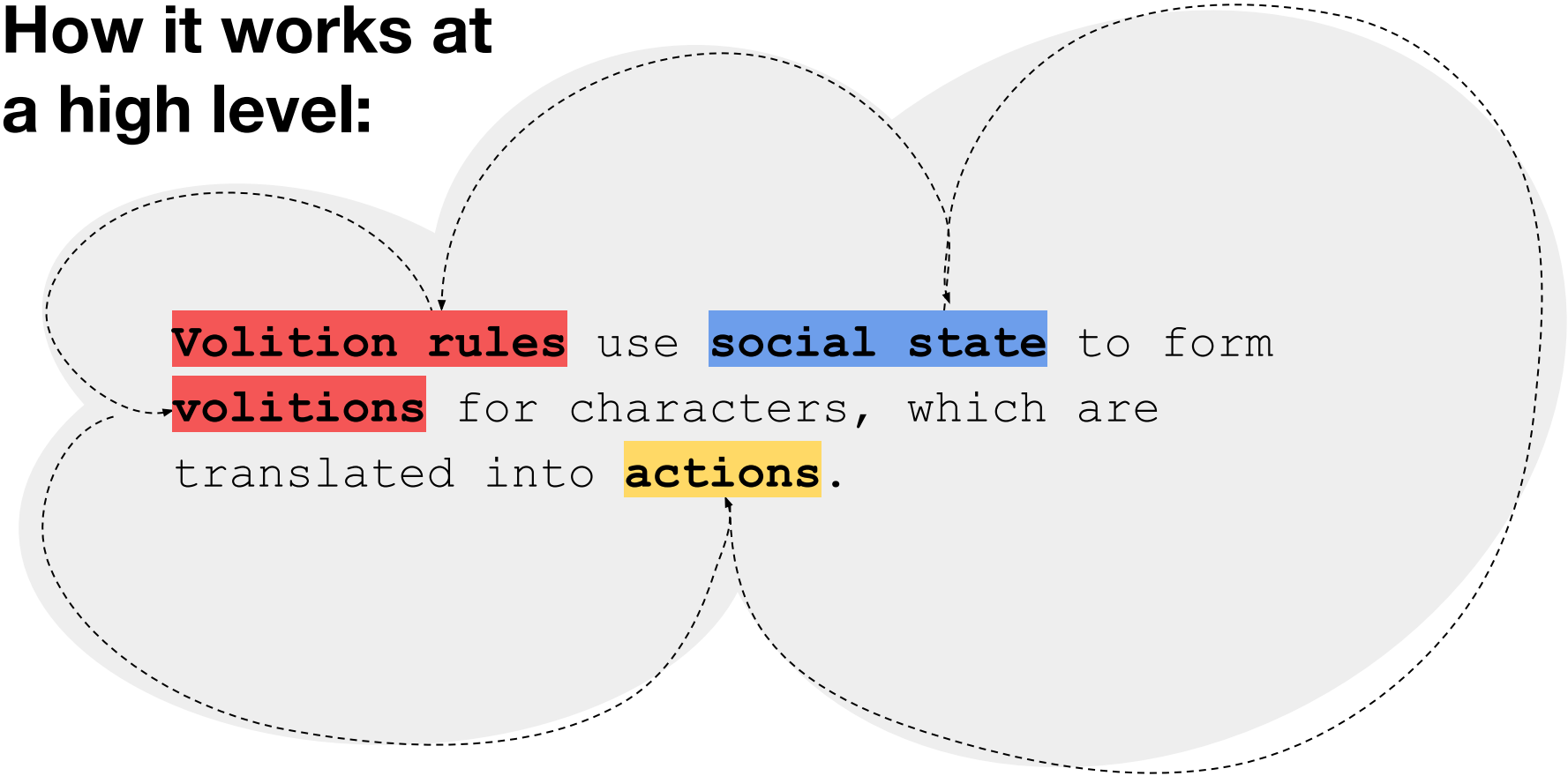


How it works at a high level:

How it works at a high level:



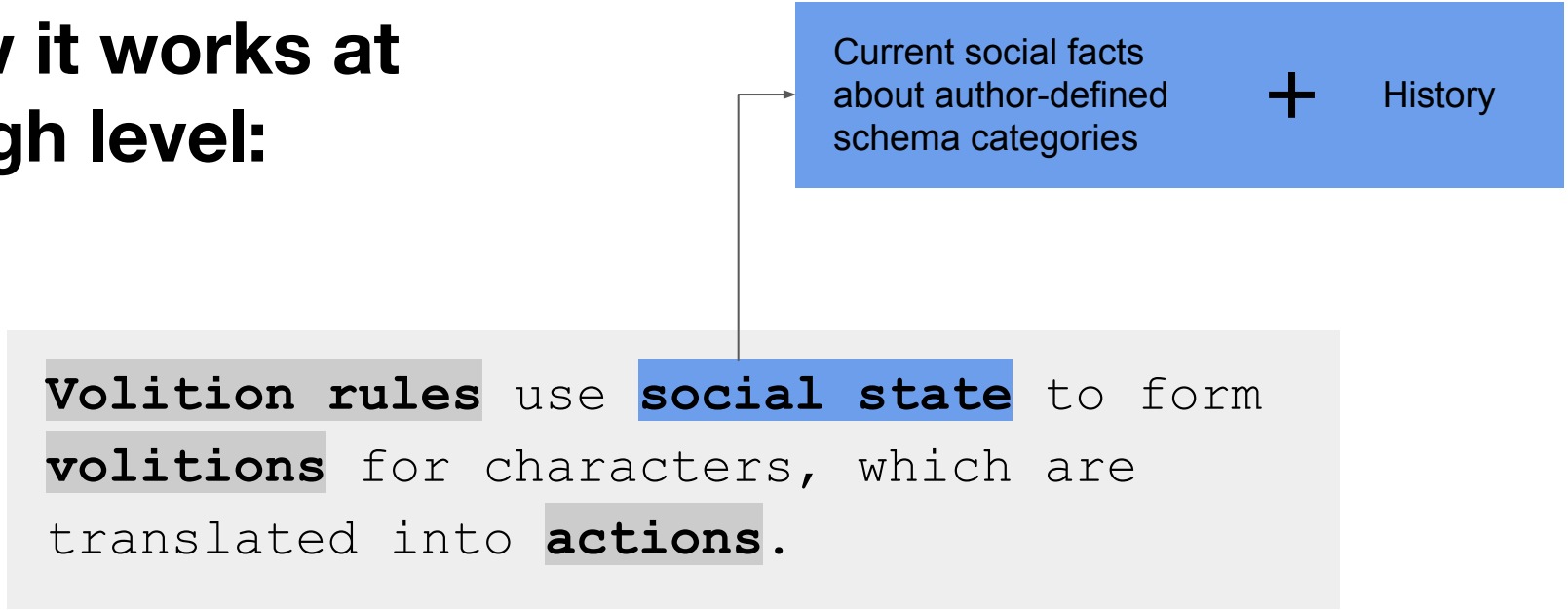
How it works at a high level:



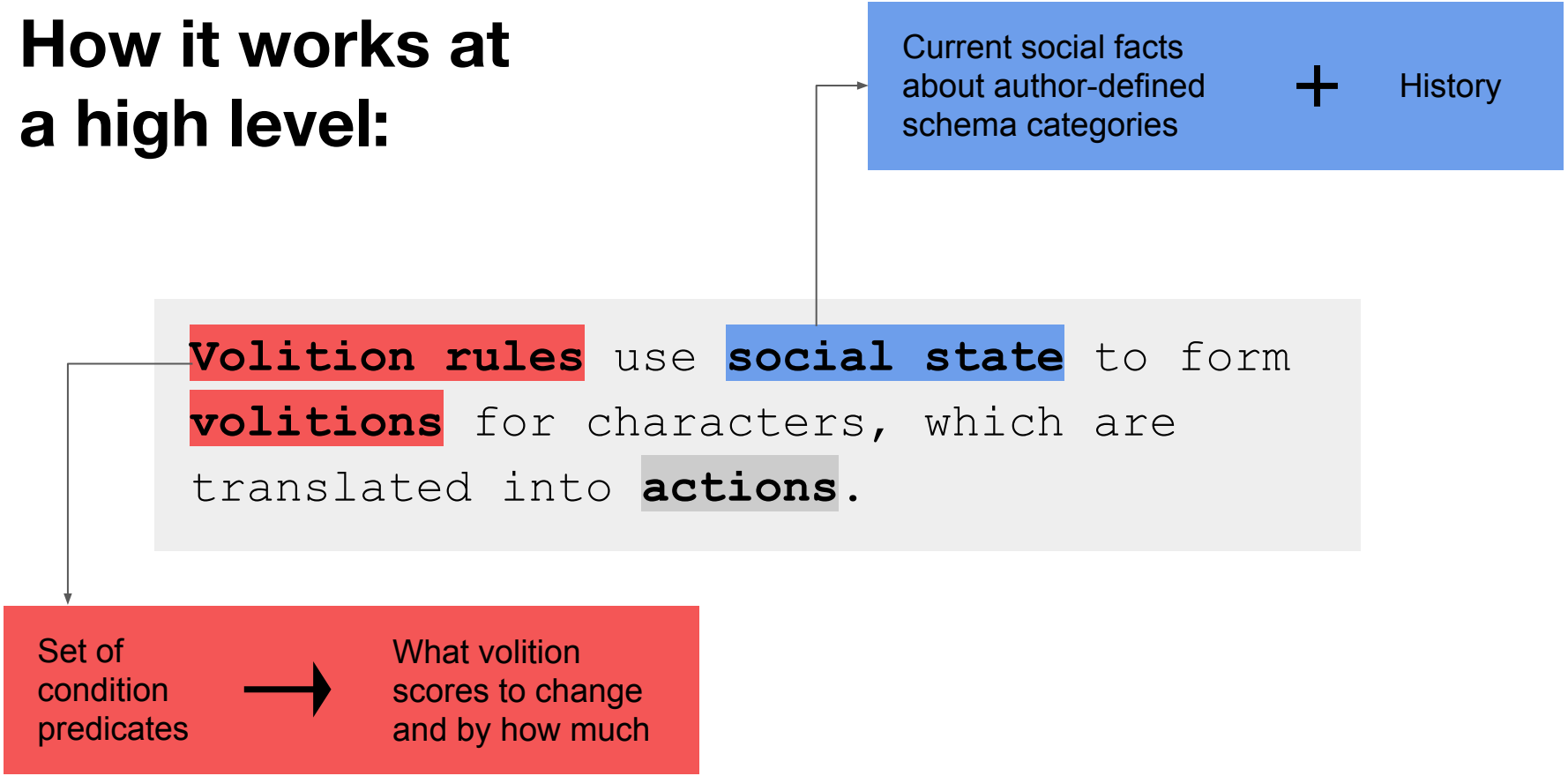
How it works at a high level:

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

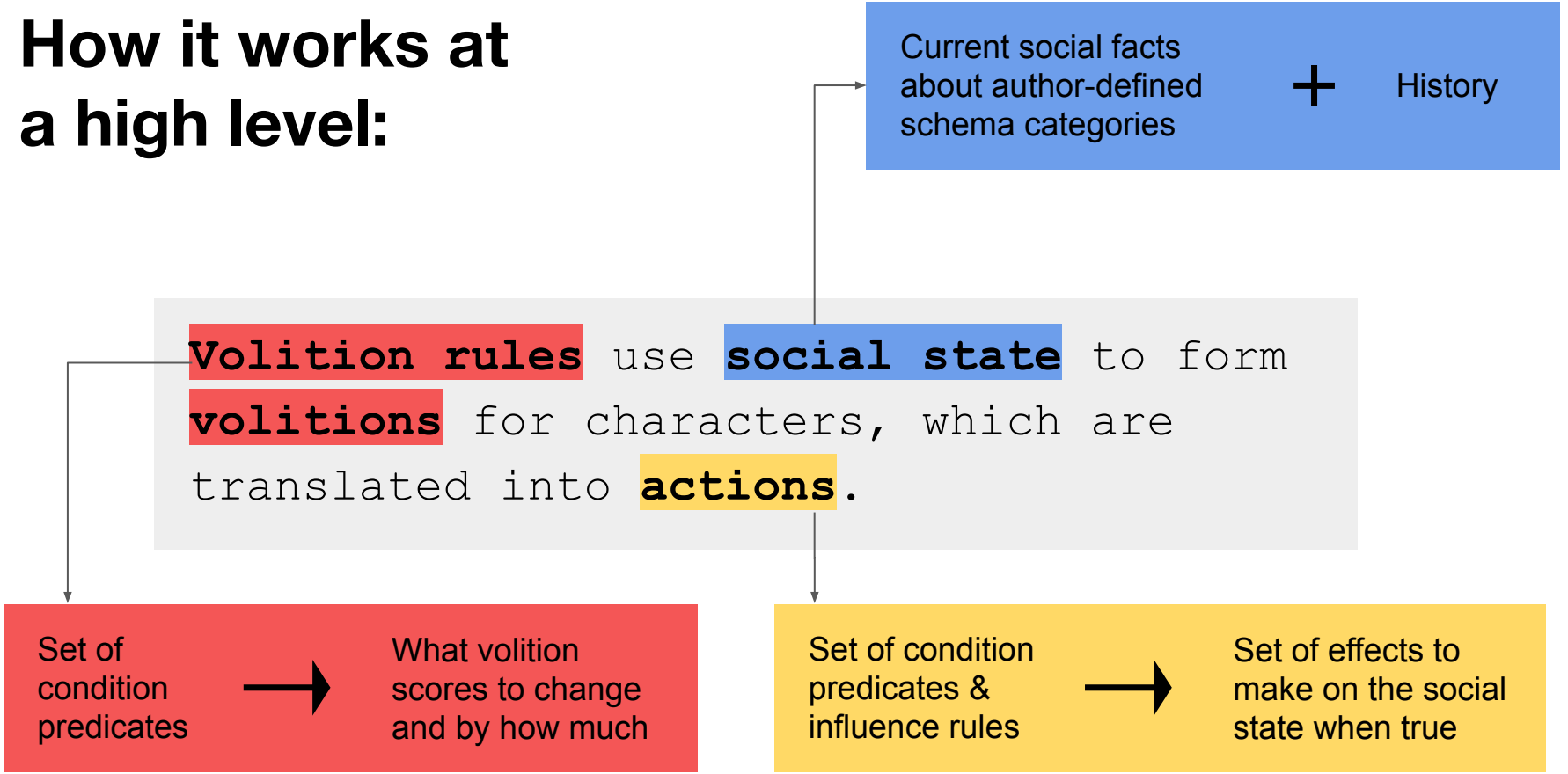
How it works at a high level:



How it works at a high level:



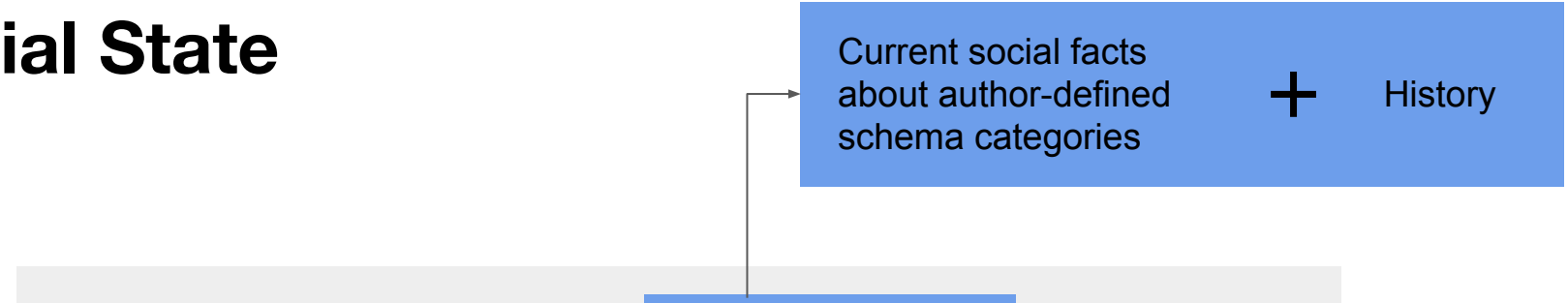
How it works at a high level:



How it works at a high level:

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

Social State

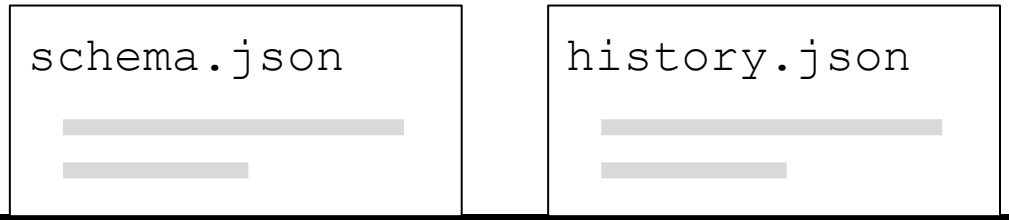


Volition rules use **social state** to form **volitions** for characters, which are translated into **actions.**

Social State

Current social facts about author-defined schema categories + History

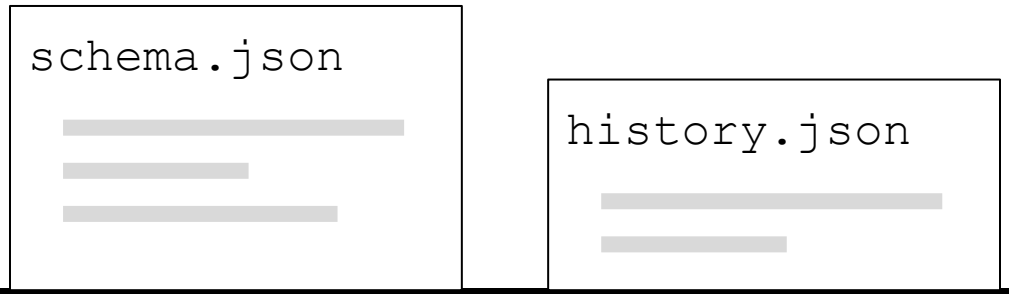
Volition rules use **social state** to form **volitions** for characters, which are translated into **actions.**



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

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

```
{
    "schema": [
        {
            "category" : "socialRecordLabel",
            "types" : [ broke-up-with ],
            "isBoolean" : true,
            "directionType" : "directed",
            "actionable" : false,
            "defaultValue" : false,
            "comment" :
                "A socially-relevant event"
        }
}
```

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"
          }
}
```

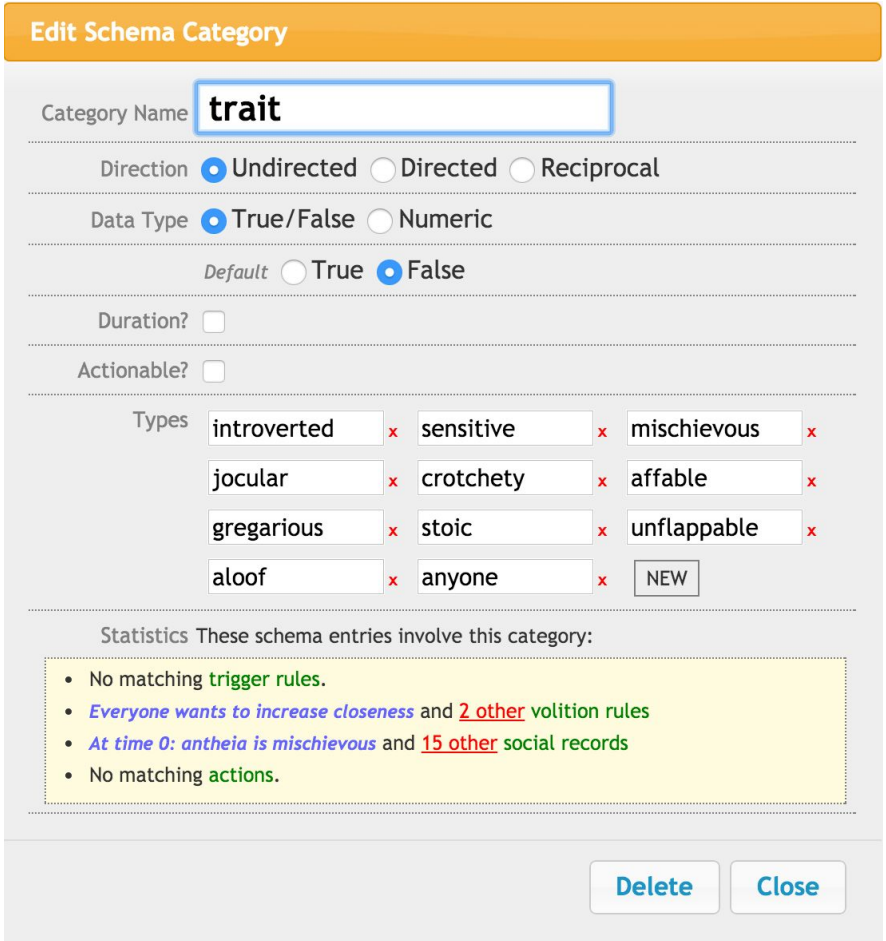
Schema.json Fields In Brief

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
`minValue`	*Number (only needed if `isBoolean`==false)*
`maxValue`	*Number (“”)*
`duration`	*Number*

Sometimes

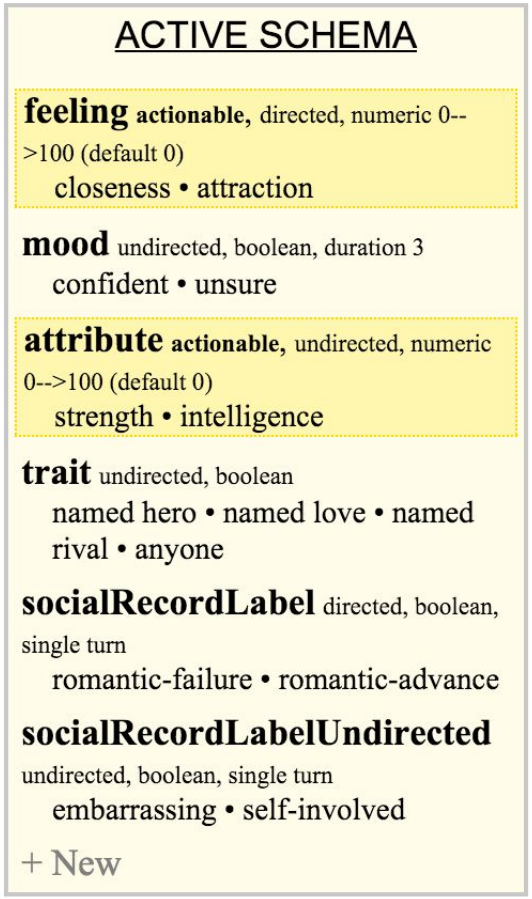
Schema through the lens of the Authoring Tool

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



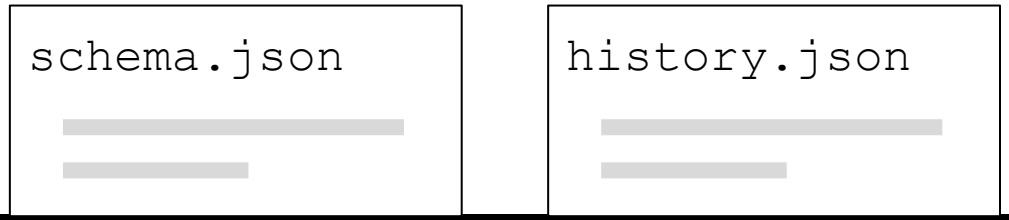
Schema through the lens of the Authoring Tool

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



More Social State

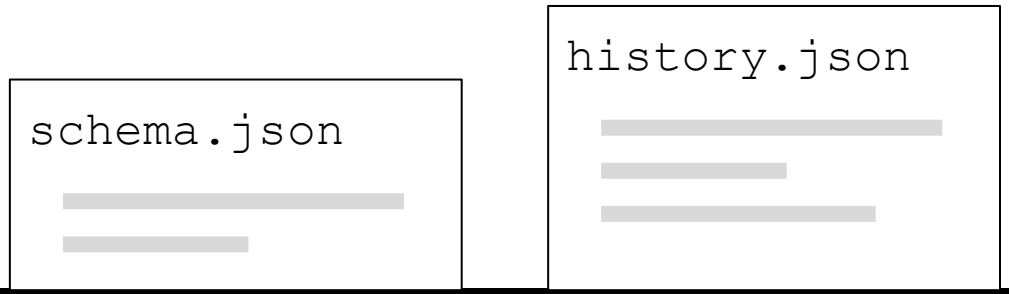
Volition rules use **social state** to form **volitions** for characters, which are translated into **actions.**



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.**



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" : "mischievous",
                    "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" : "mischievous",
                    "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" : "mischievous",
                    "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" : "mischievous",
					"first" : "ara",
					"value" : true
				},{
					"category" : "relationship",
					"type" : "best-buds",
					"first" : "calypso",
					"second" : "ara",
					"value" : true
				}
			]
		}
	]
}
```

History.json

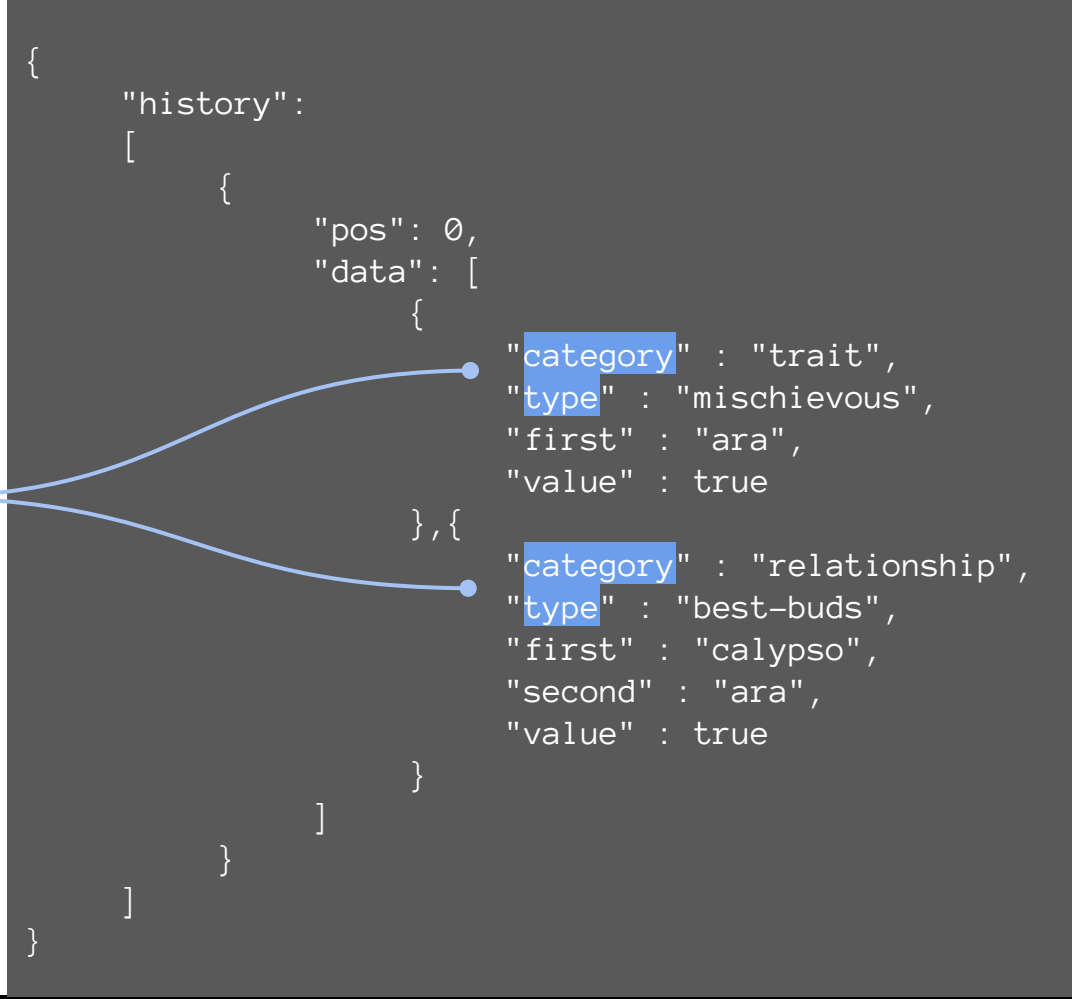
Fields for a state object within the `data` array:

`category`

What schema category you're referencing

`type`

What type within that category that you're referencing



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"],
          ...
}}
```

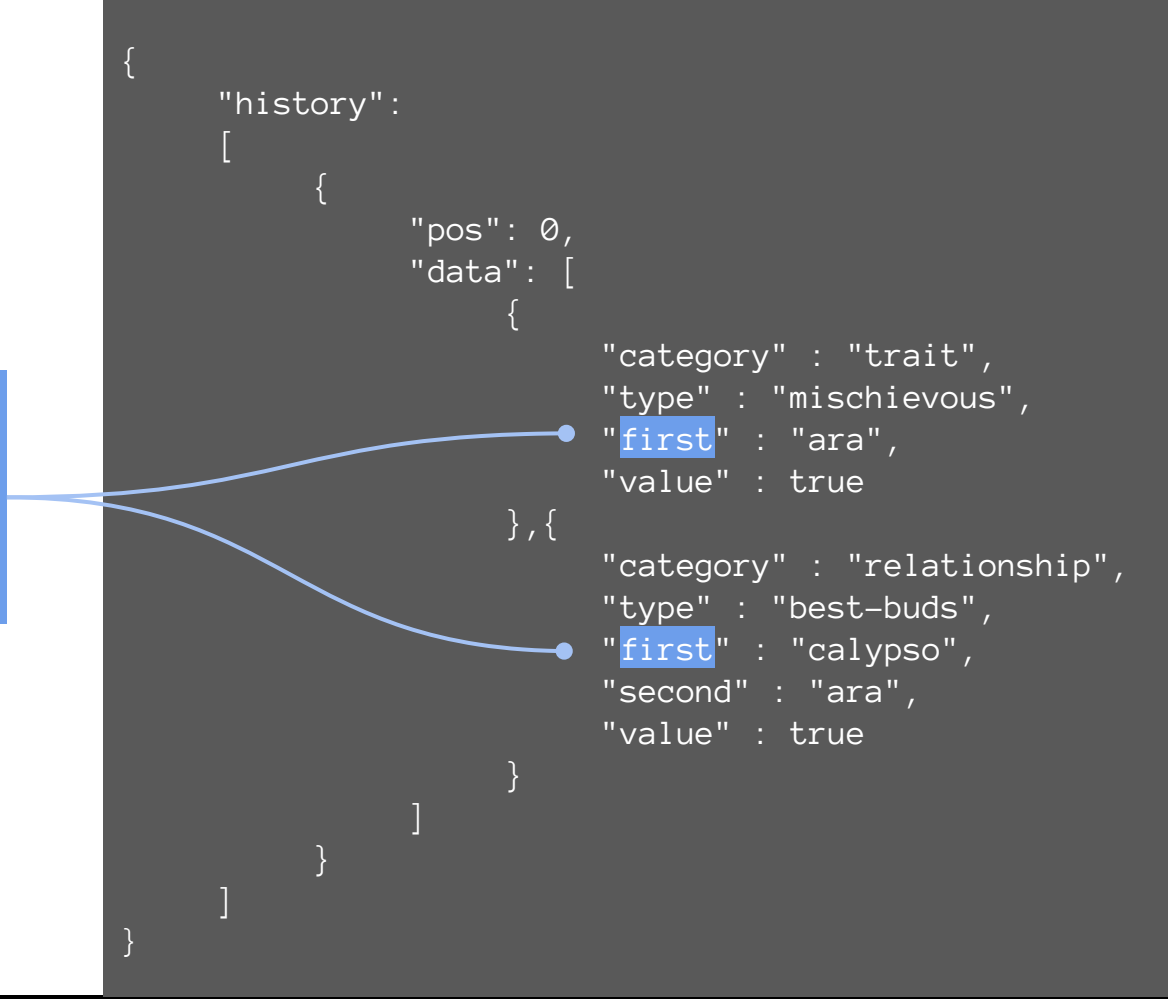
```
{
     "history":
     [
          {
               "pos": 0,
               "data": [
                    {
                         "category" : "trait",
                         "type" : "mischievous",
                         "first" : "ara",
                         "value" : true
                    },{
                         "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.json

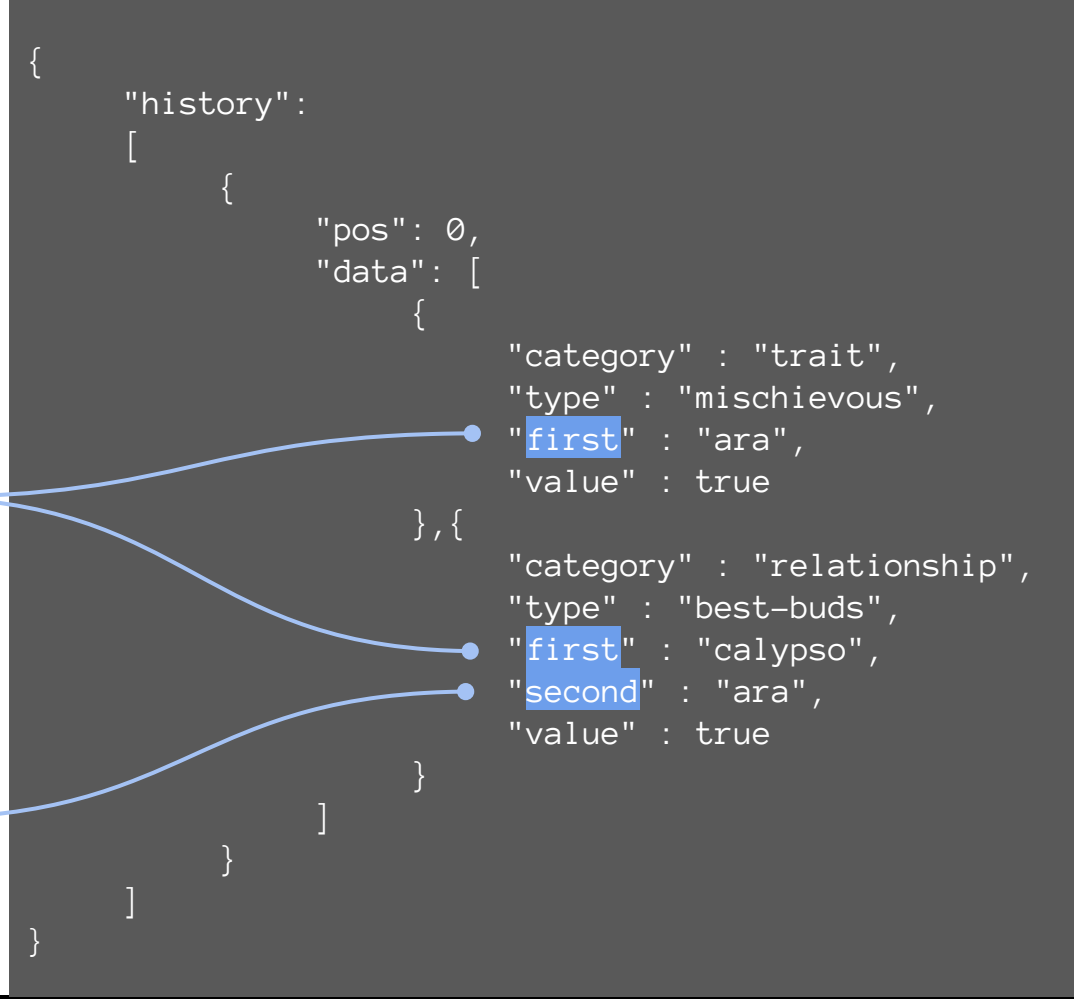
Fields for a state object within the `data` array:

`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

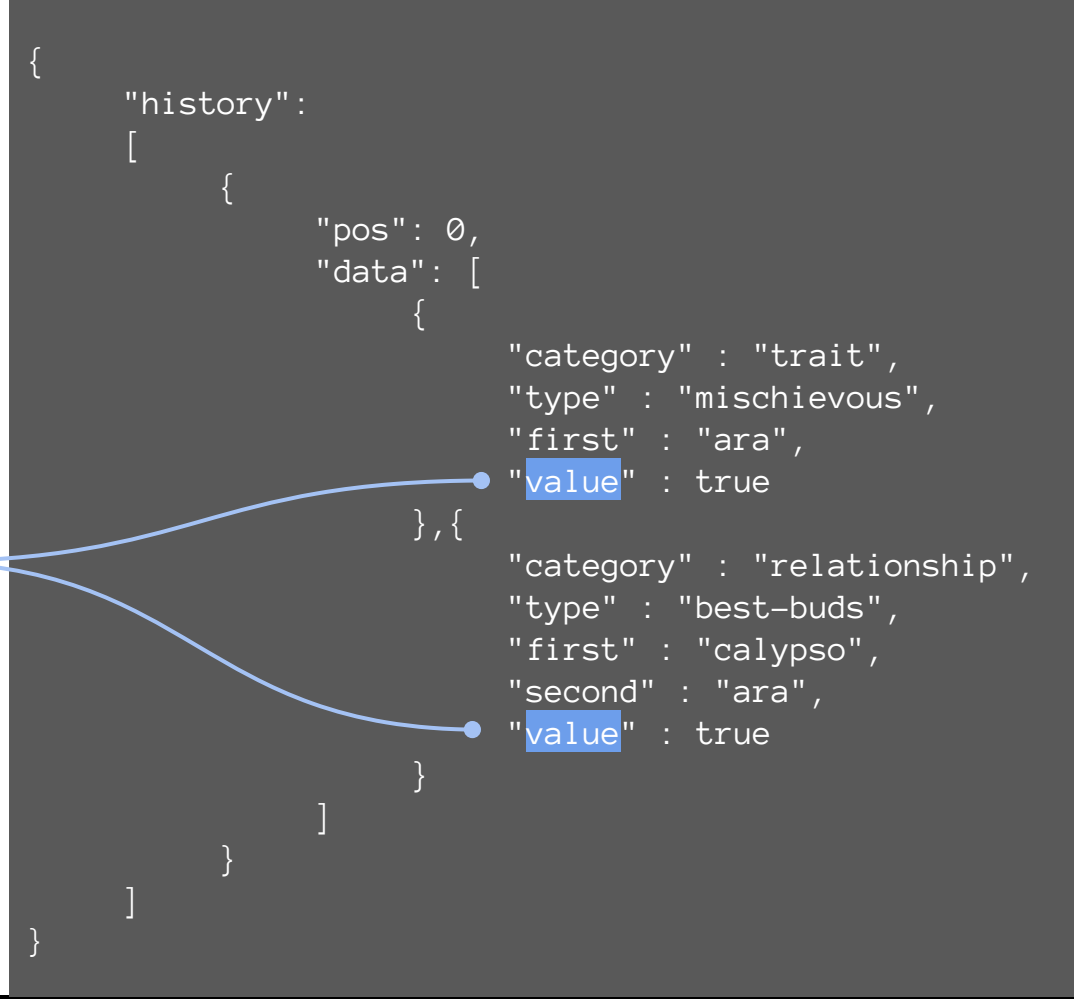


History.json

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`



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

By connecting our social data types…

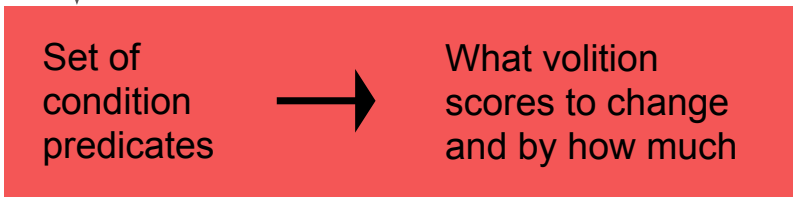
Volition rules use **social state** to form **volitions** for characters, which are translated into **actions.**

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.**



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`

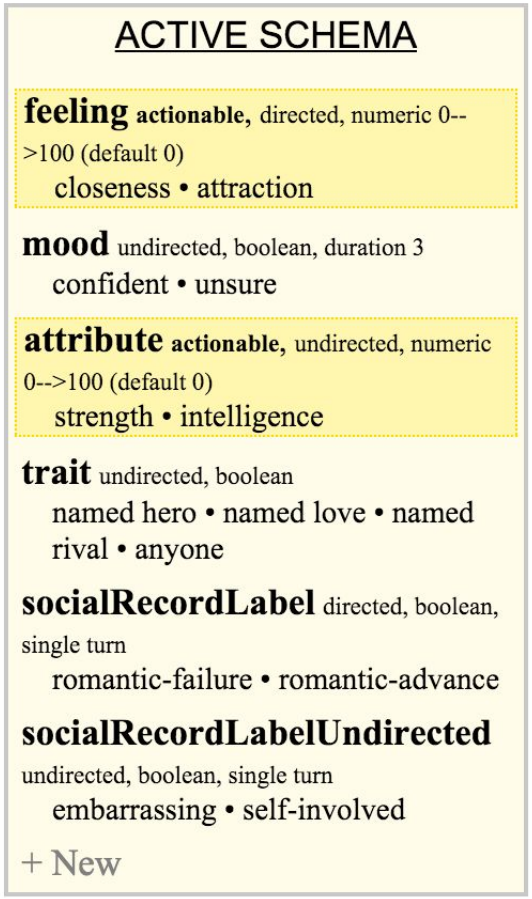
An array of action node names which are subactions of this one

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

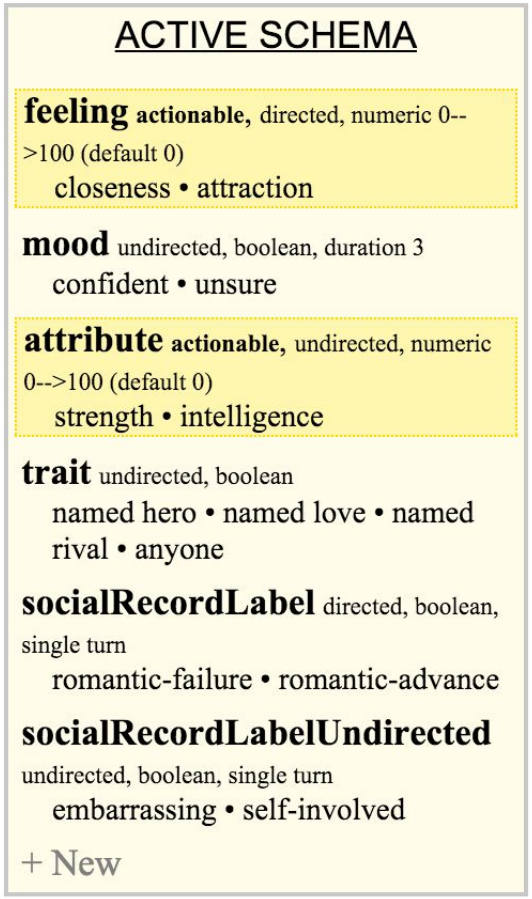
Volitions (or character desires) in Ensemble are about changes to a particular social state value, so they are deeply connected to your **schema**.



Volitions

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

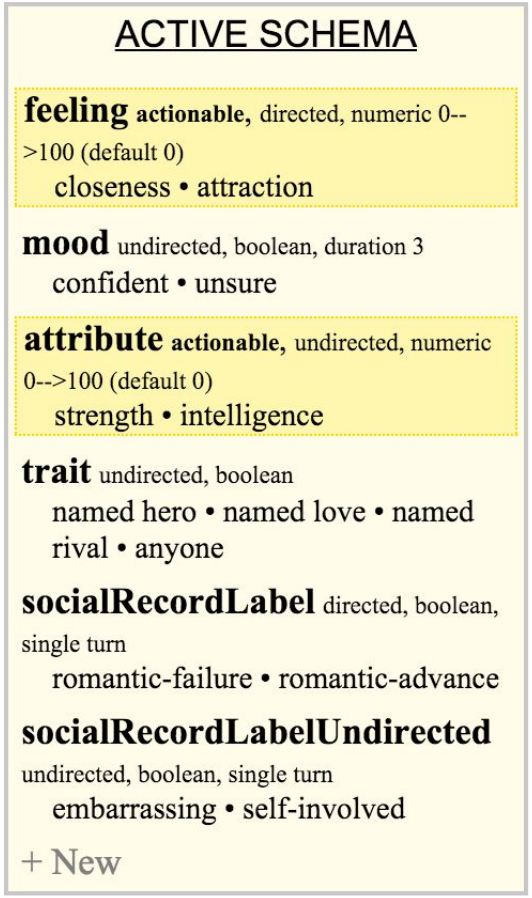
E.g., you may want characters to form a desire to increase their closeness to another character, or to increase their own confidence



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

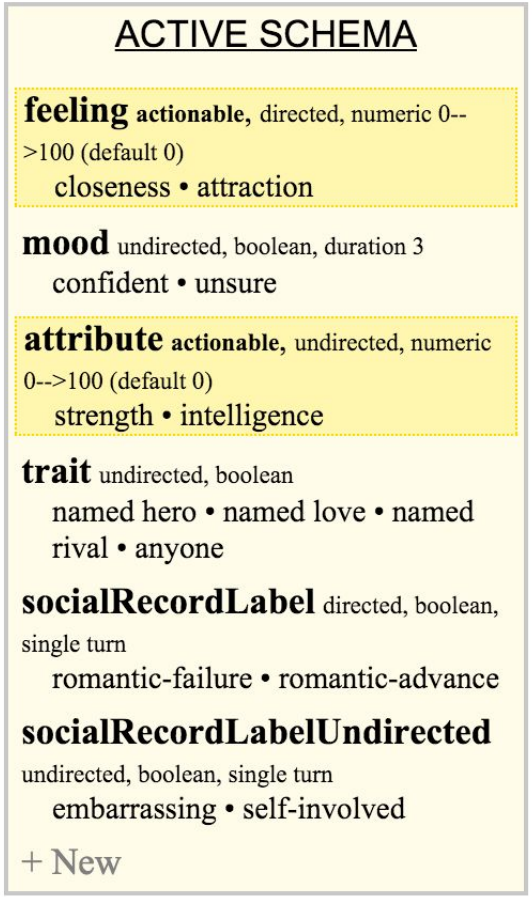


...But you probably won't want characters to form volitions about *every* schema concept

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



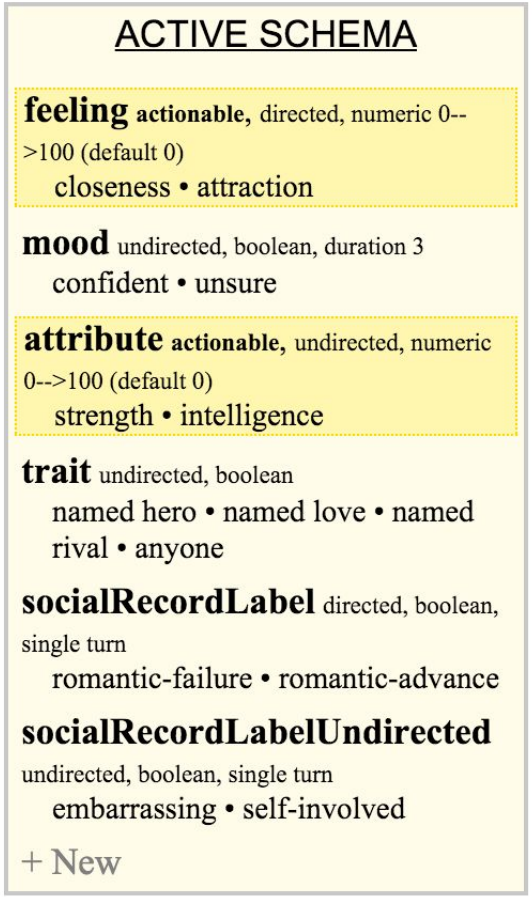
...But you probably won't want characters to form volitions about *every* schema concept

- 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



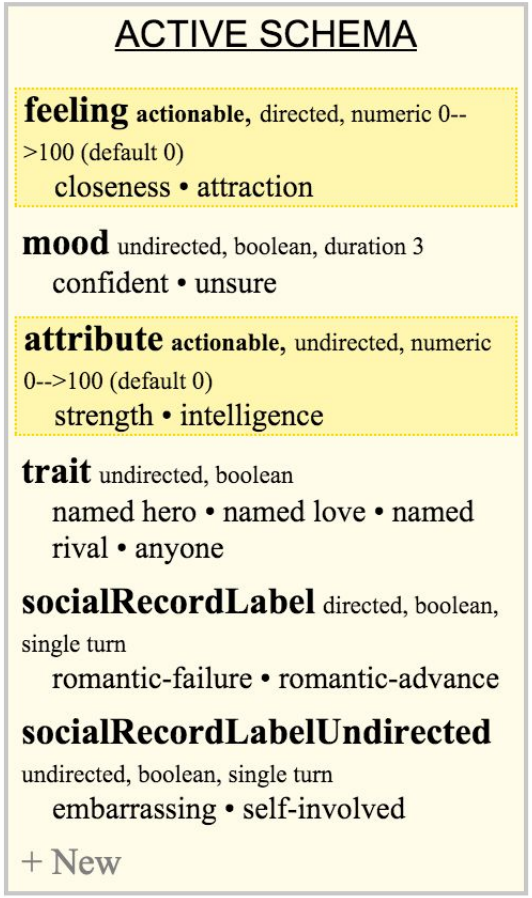
...But you probably won't want characters to form volitions about *every* schema concept

- 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



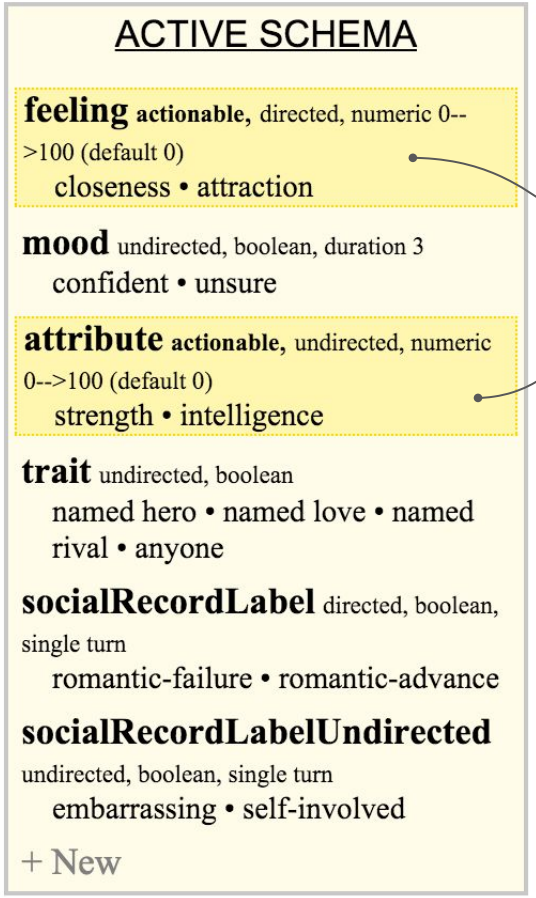
...But you probably won't want characters to form volitions about *every* schema concept

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

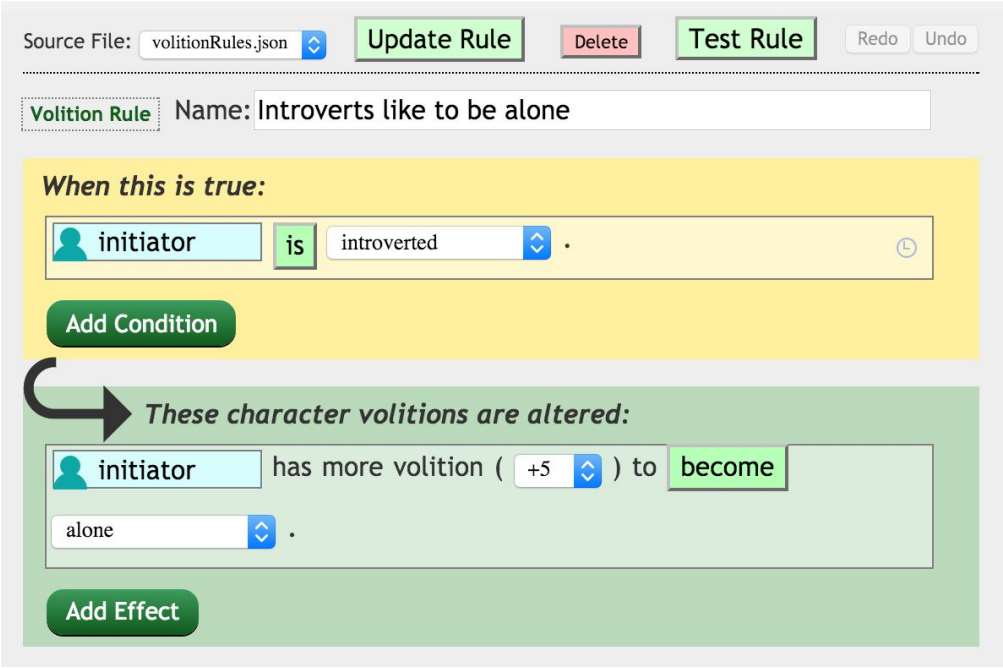
One rule :=

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

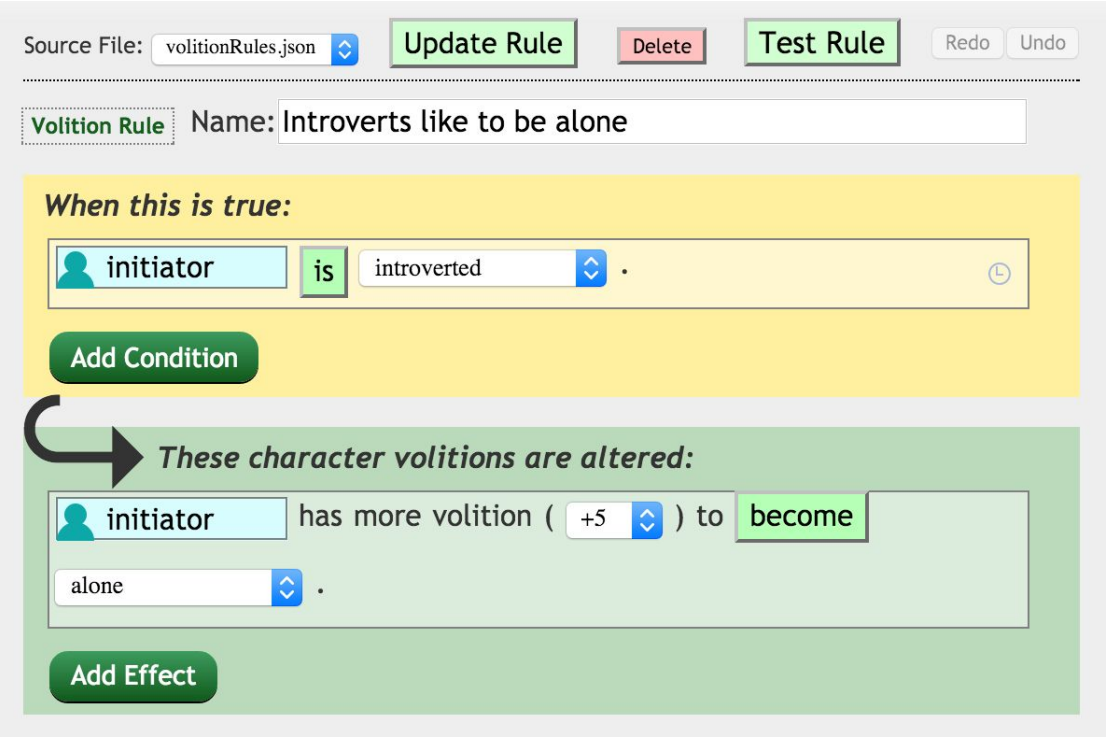
```
{     "fileName": "volitionRules",
      "type": "volition",
      "rules": [
            {
                  "name": "Introverts like to be alone",
                  "conditions": [
                        { "category": "trait",
                          "type": "introverted",
                          "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



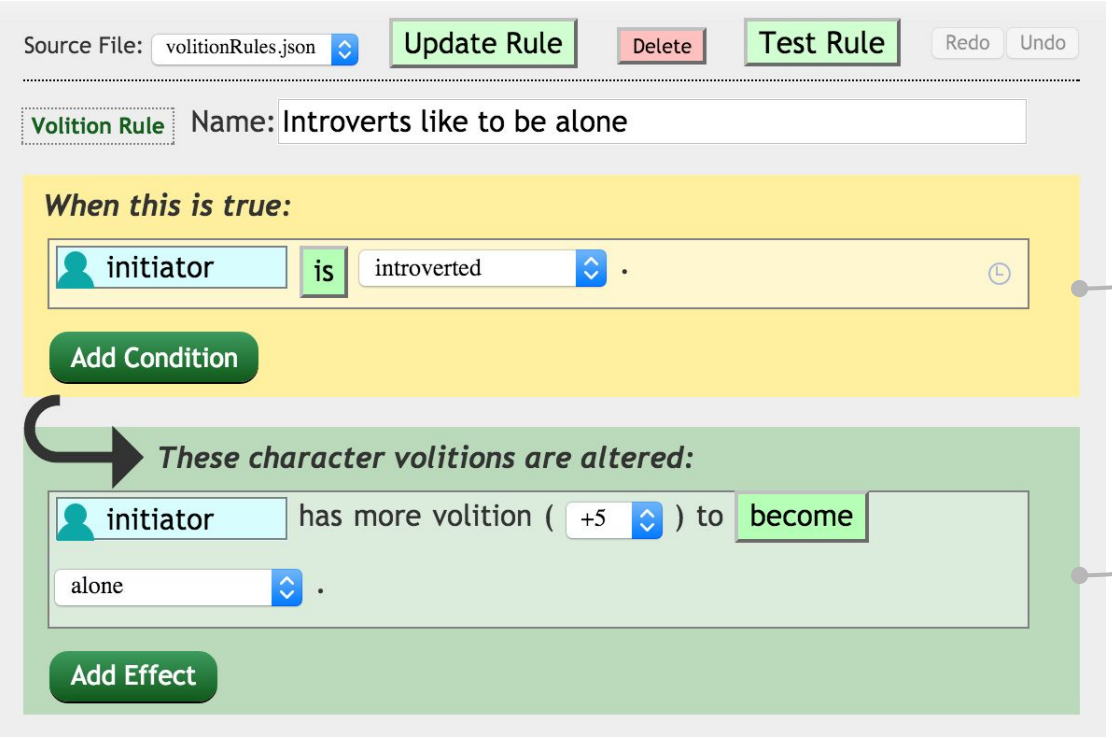
Volition Rule Editor



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,
            }
      ]
}
```

Volition Rule Editor



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,
            }
      ]
}
```

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)

...

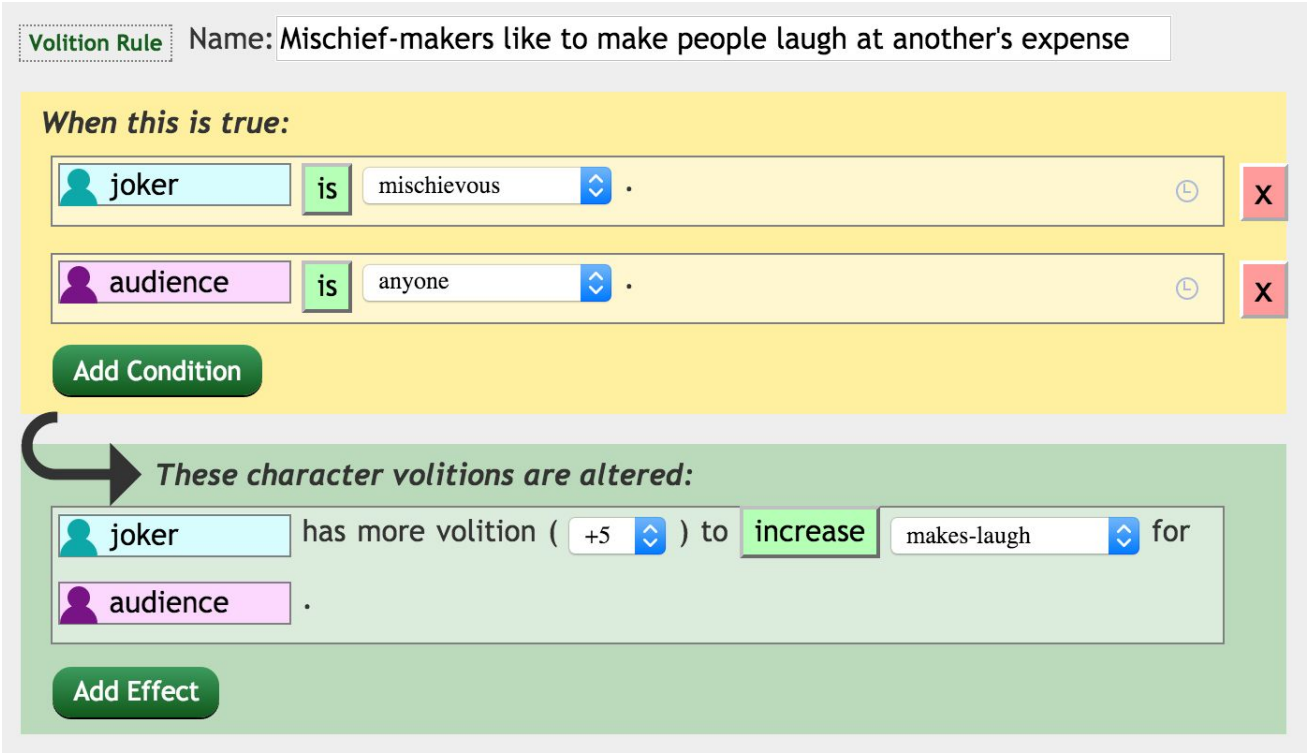
All the volition rules that say when someone wants to help someone else

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

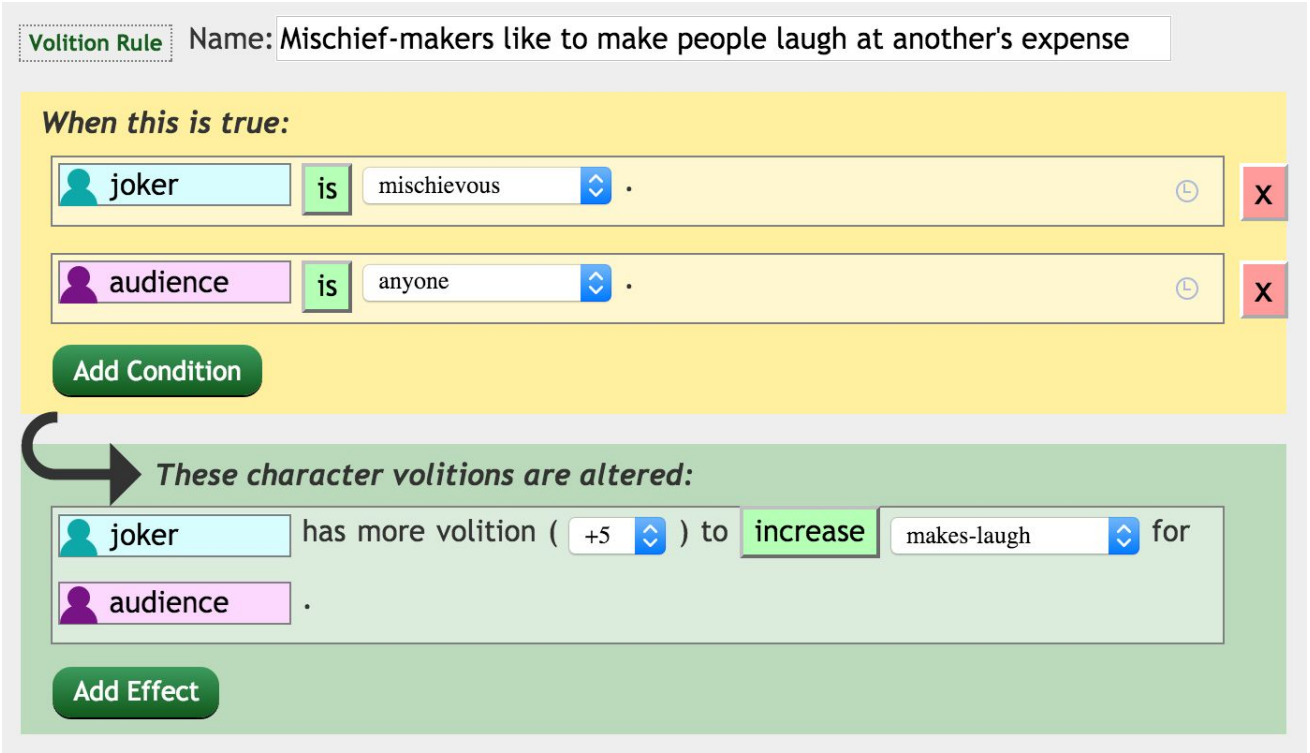
Authoring Volition Rules



Conditions

Can have any number of condition predicates

Authoring Volition Rules

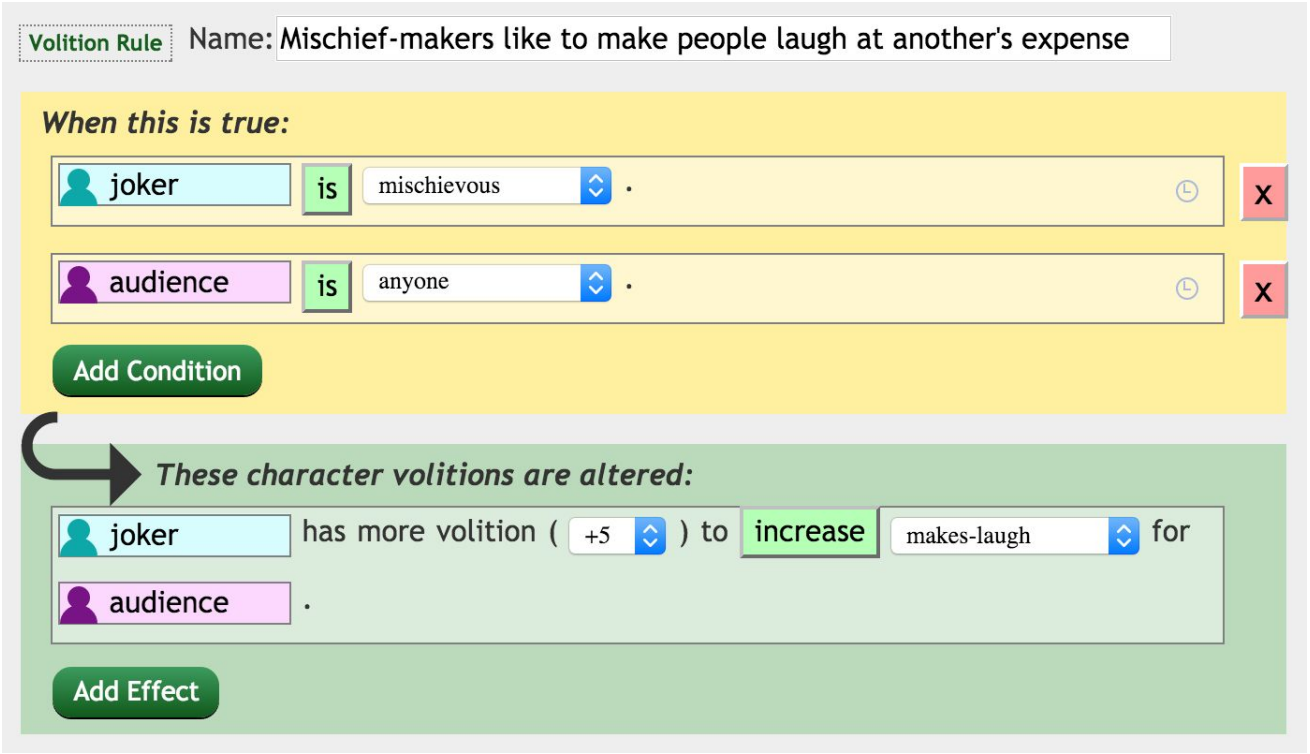


Conditions

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



Effects

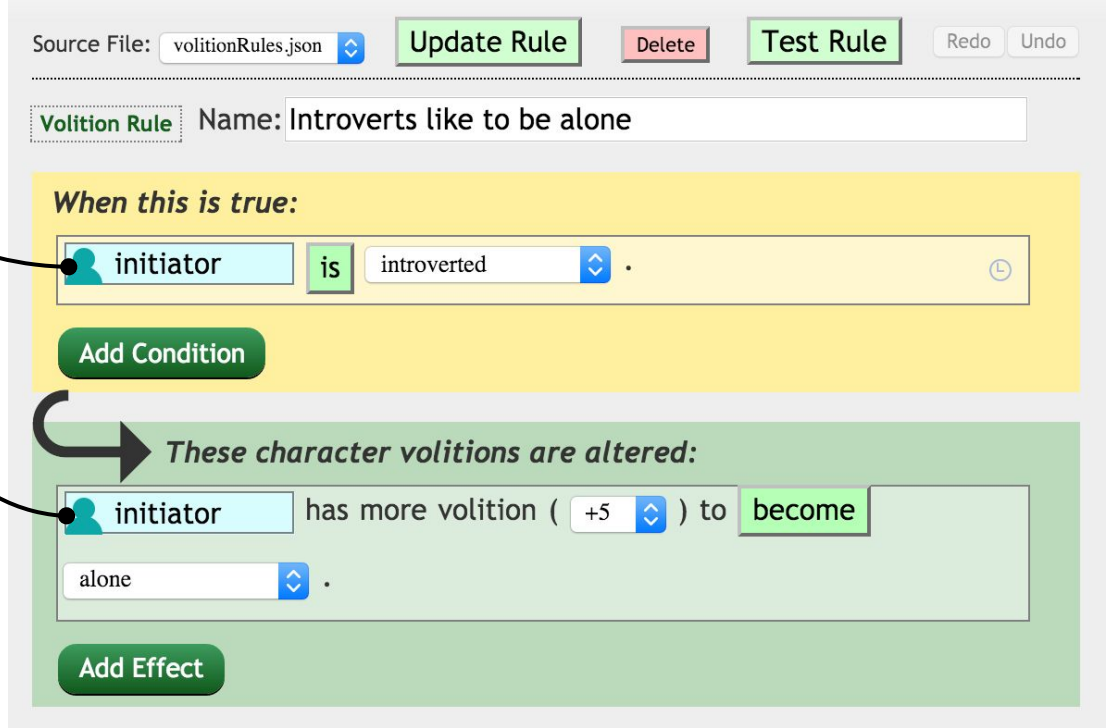
Can also affect any number of character volitions

(but usually one)

Authoring Volition Rules

Roles

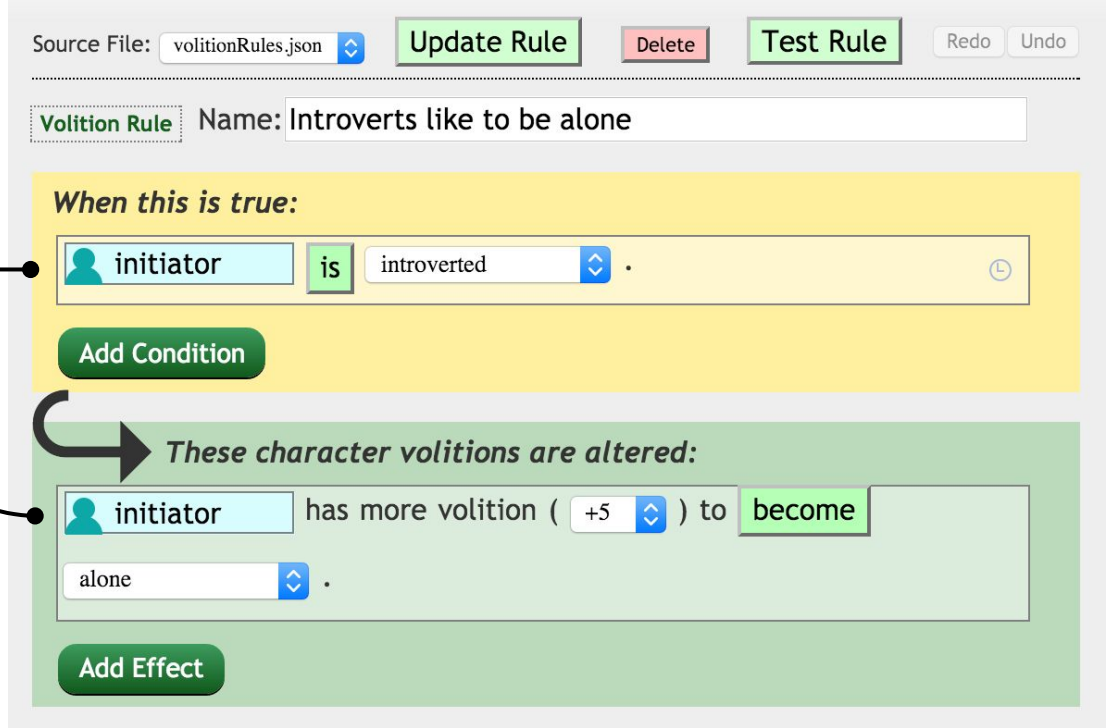
A character role can be any string, and you can have any number of roles in a rule. They're like local variables, that only exist in the scope of a rule.



Authoring Volition Rules

Roles

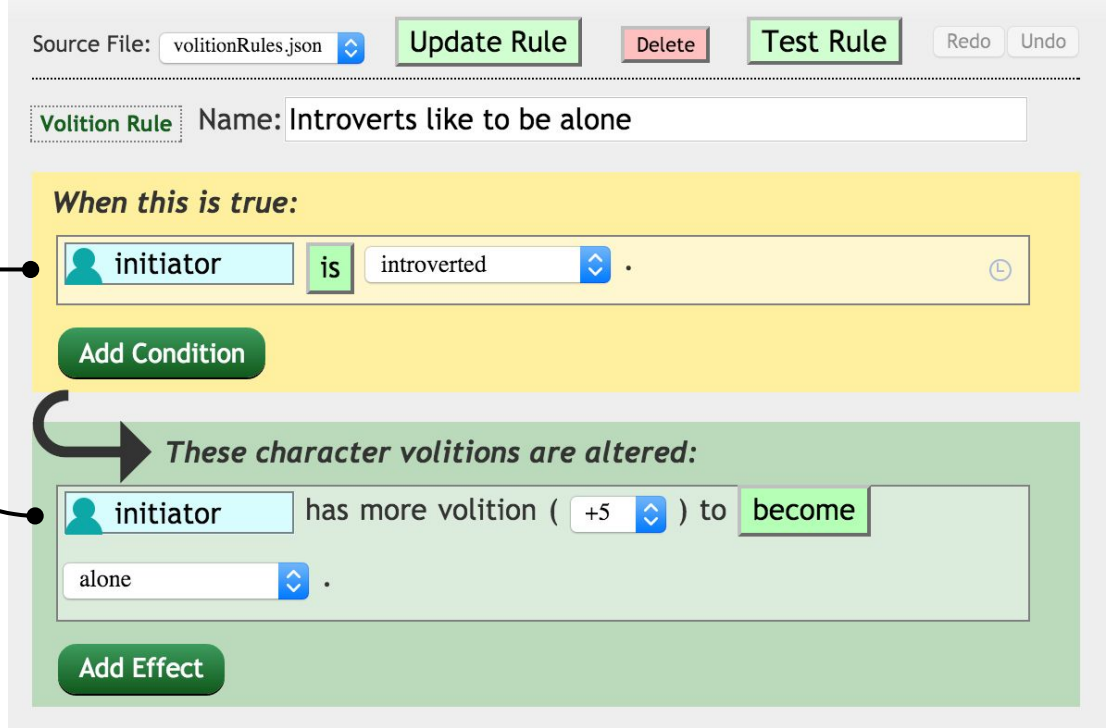
But roles are bound to characters in the **condition predicates**, not in **effects**. So any role that appears in the effects must be “initialized” in a condition.



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).

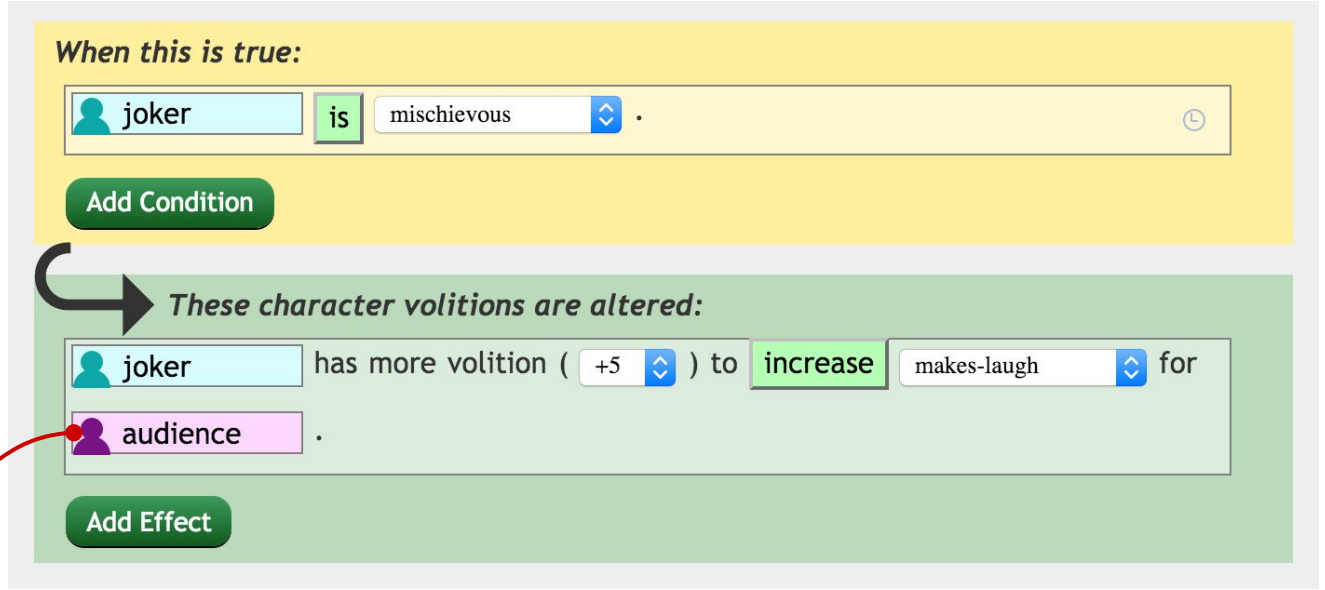


Authoring Volition Rules

Roles

So this won't work:

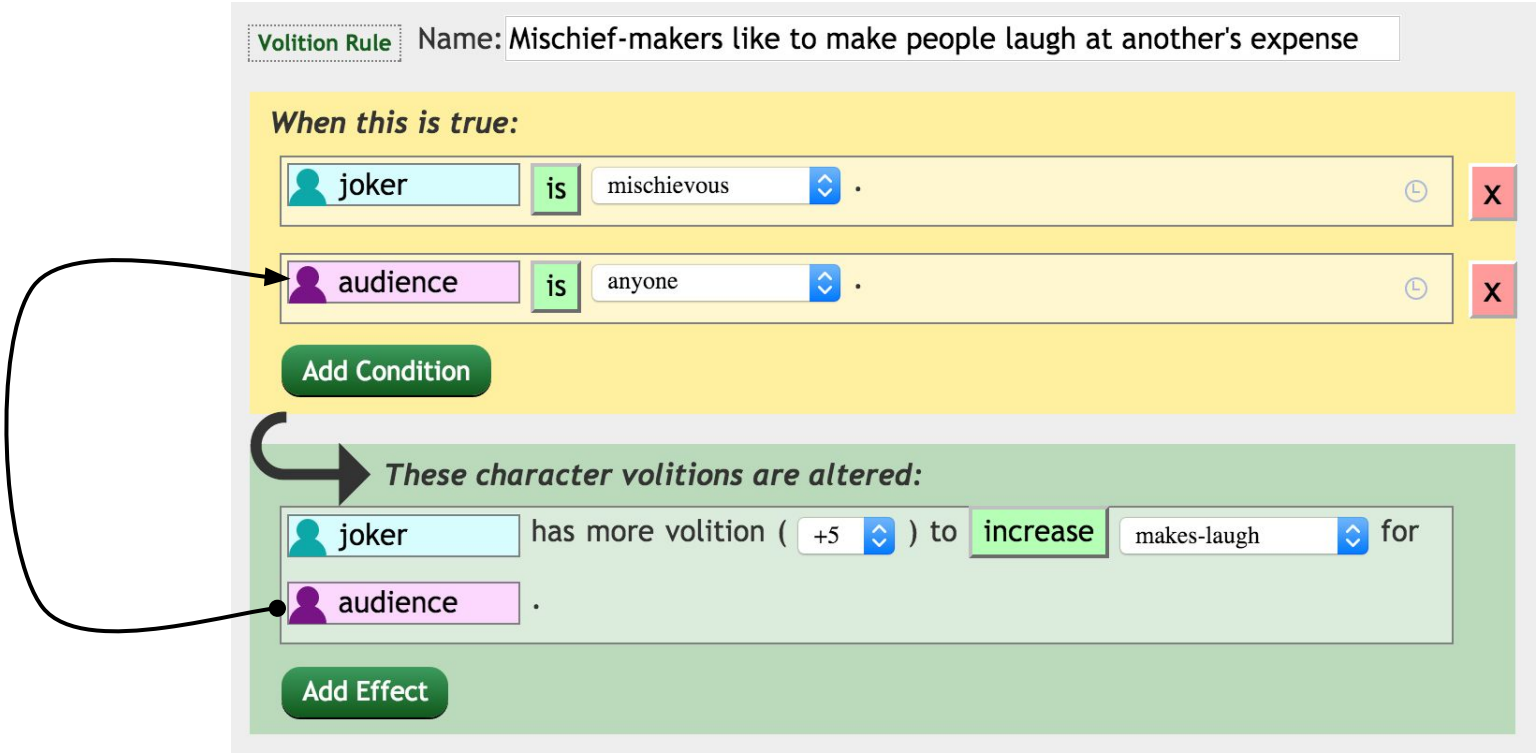
(Rule will never fire, and will not throw an error.)



Authoring Volition Rules

Roles

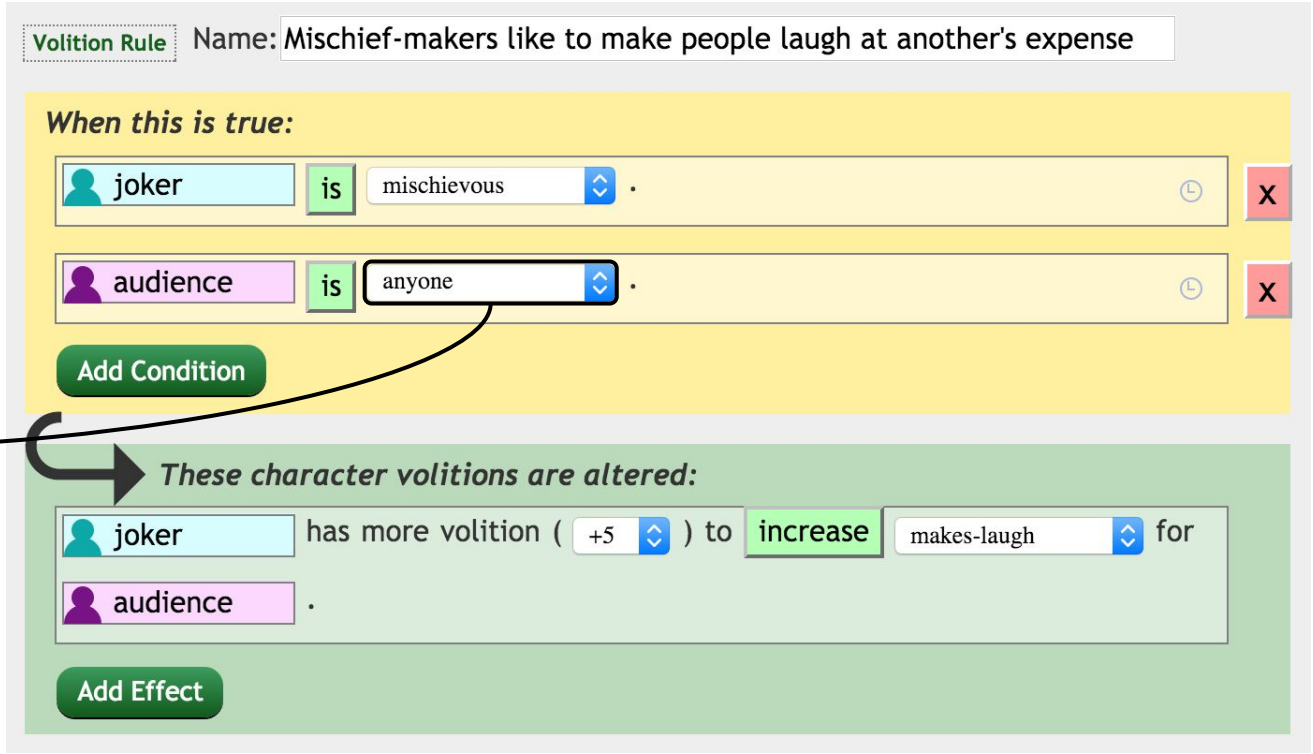
Corrected:



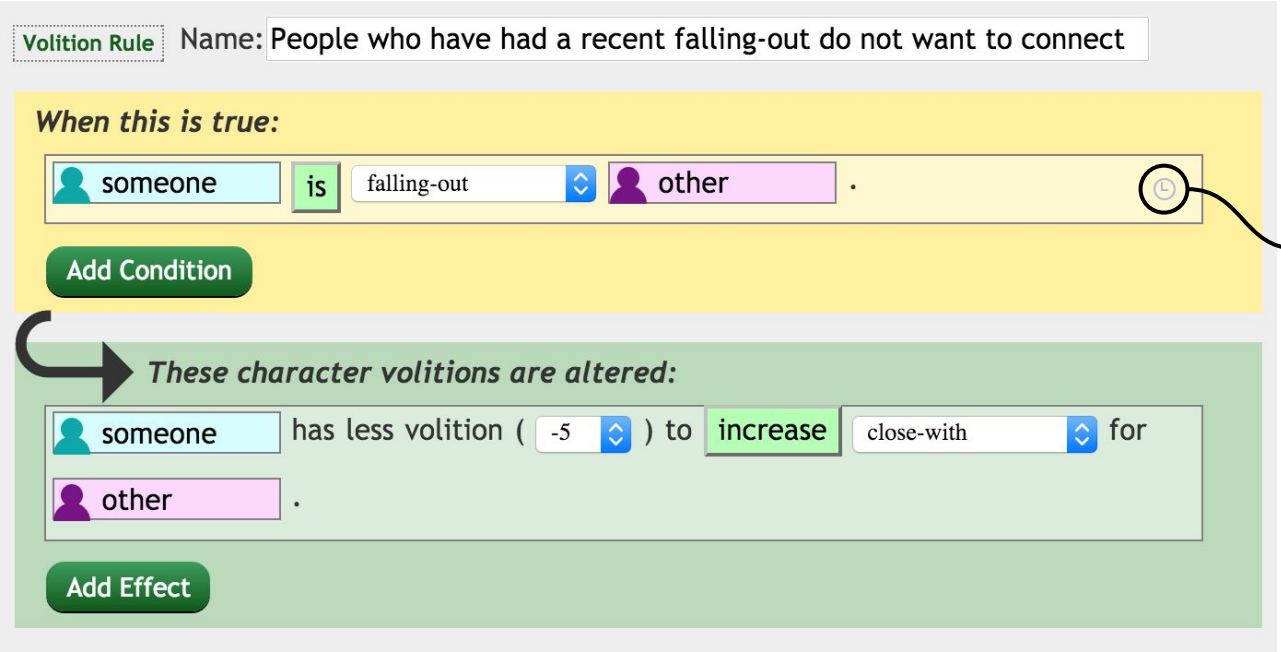
Authoring Volition Rules

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.



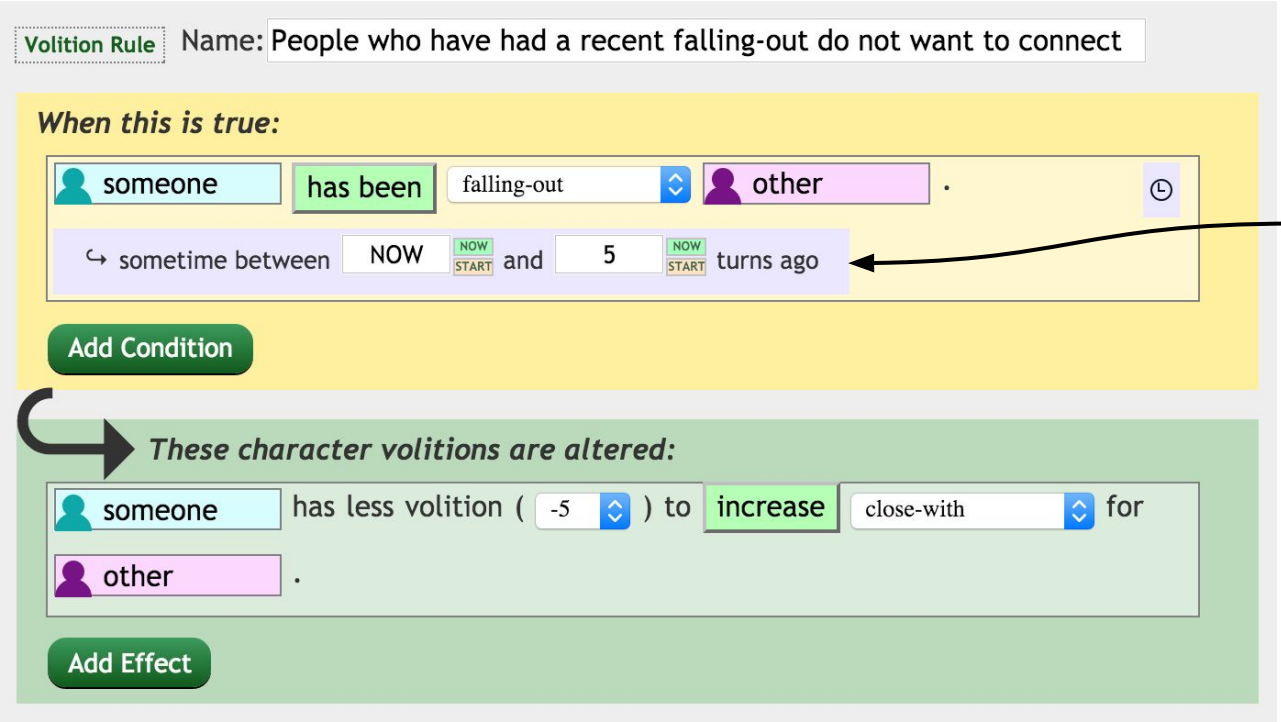
Authoring Volition Rules



Referencing History

Select the clock icon next to a condition predicate

Authoring Volition Rules



Referencing History

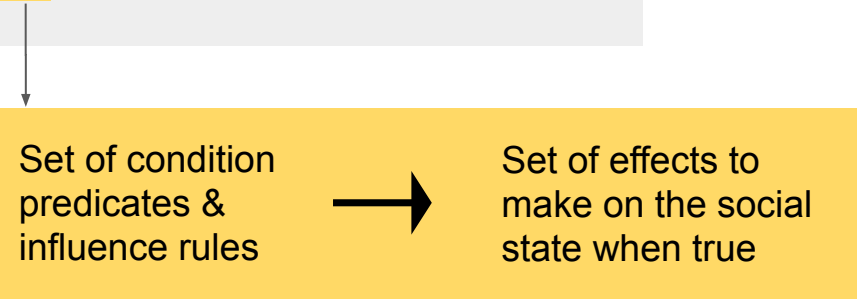
Specify time frame

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**.

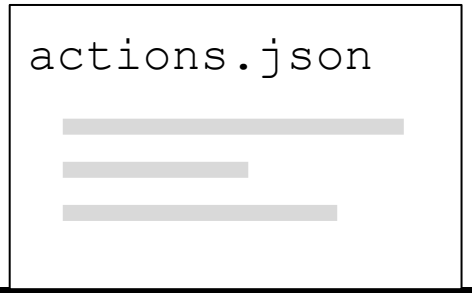
Now we'll enable characters to take action based on their desires

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



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

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

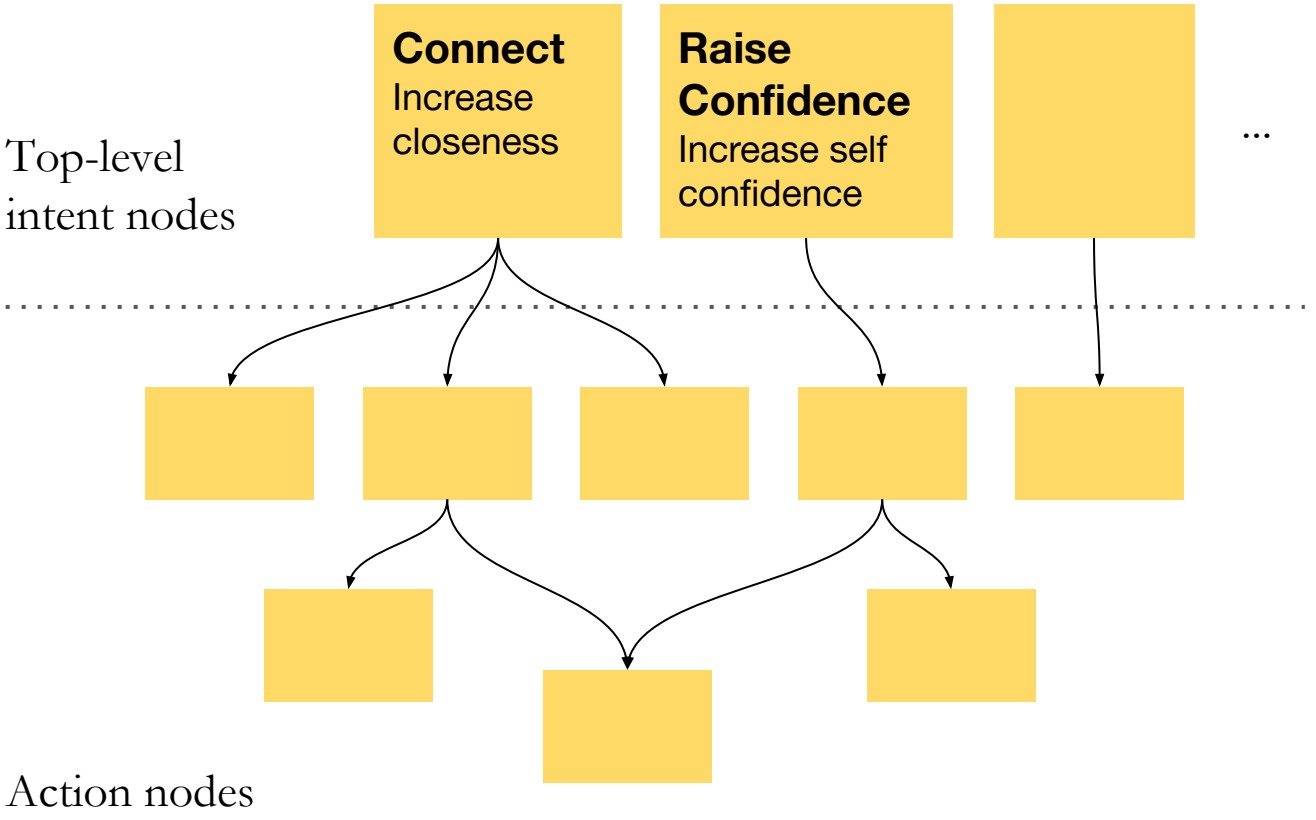
Always required	**Sometimes required**, Depending on where the node is in an action graph

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

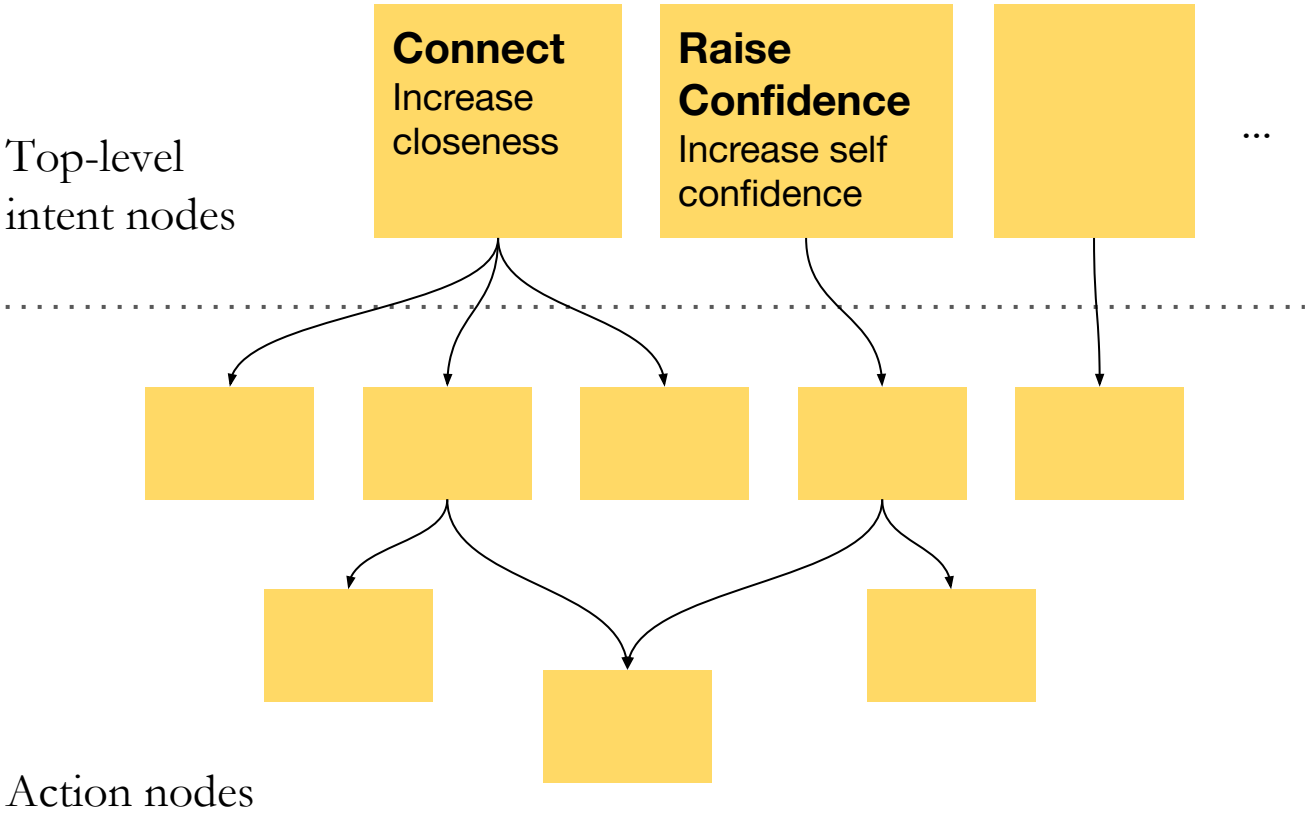


Actions

Ensemble actions have a graph structure.

—Directed, but not strictly hierarchical

—Many root nodes, corresponding to each volition that appears in volition rules



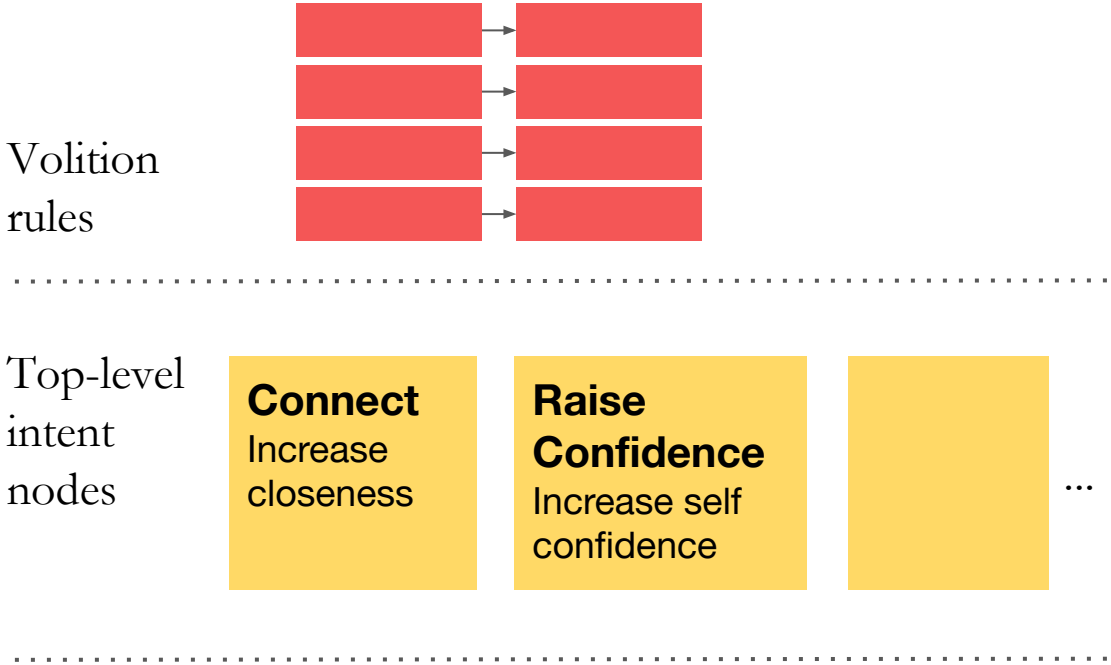
Top-Level Intent Nodes

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

Top-Level Intent Nodes

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

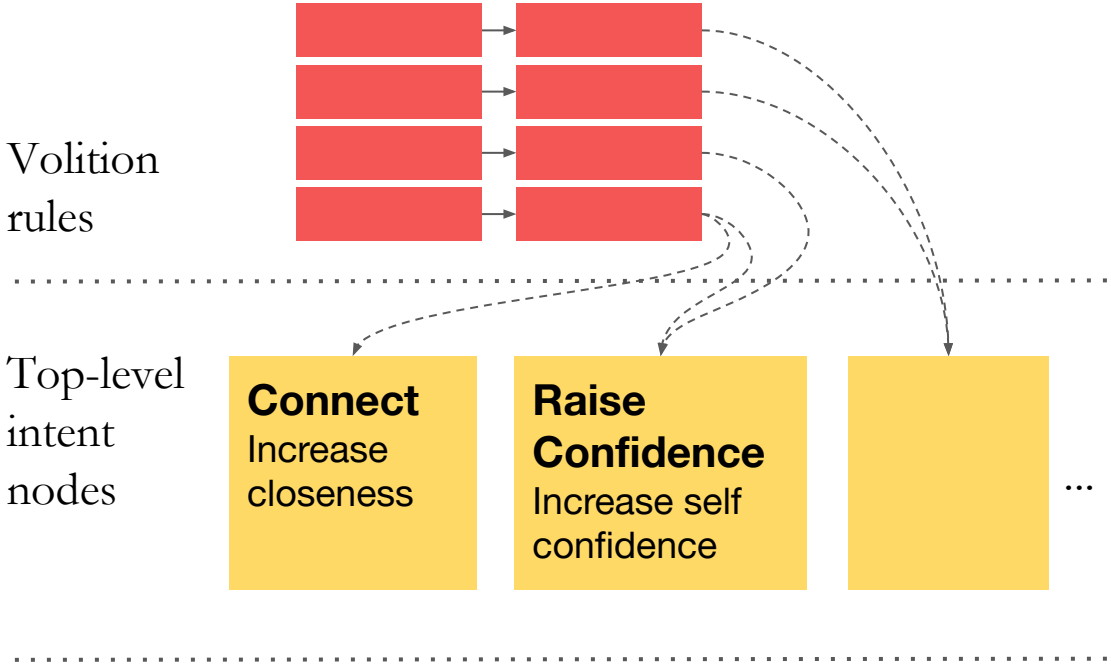
They're what connect volition rules to actions.



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.



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`

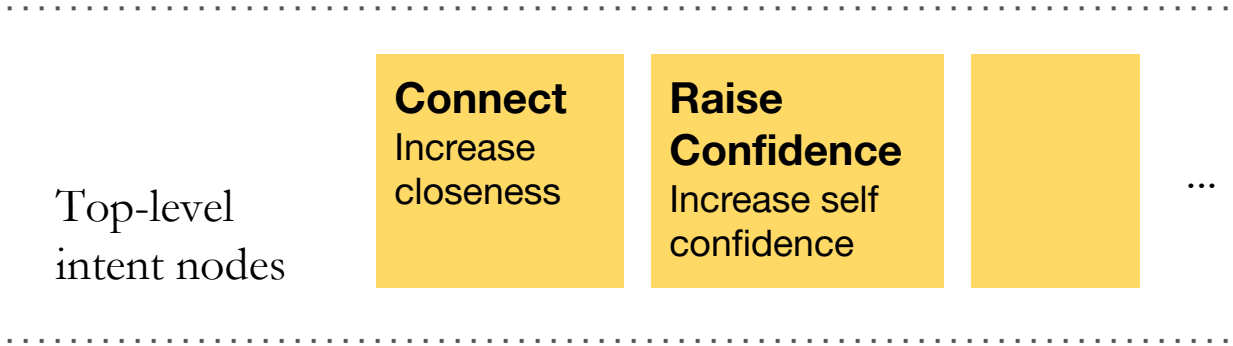
Array of hard preconditions that must hold true for this action to be considered.

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

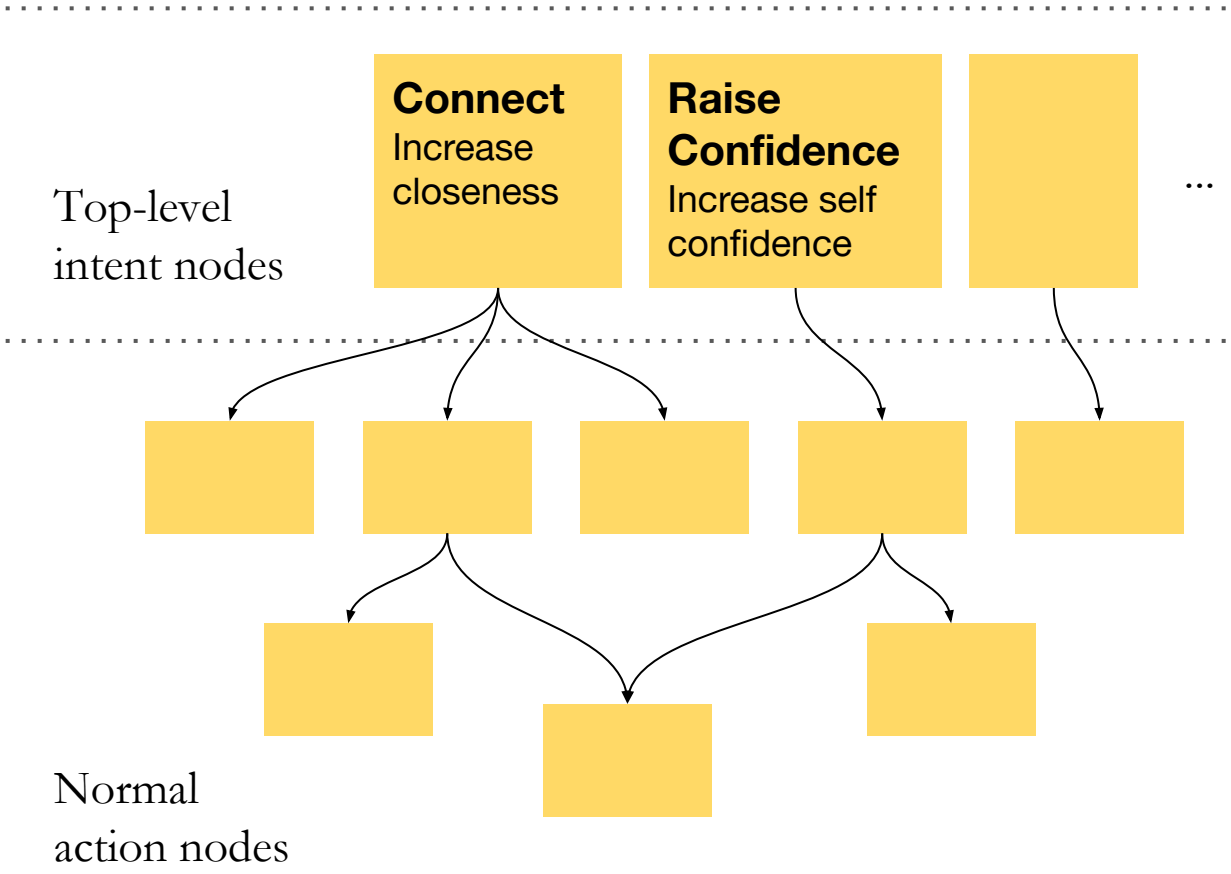
Intent level action nodes can't be the only nodes in your action graph.



Top-Level Intent Nodes

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.

```
{
    "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

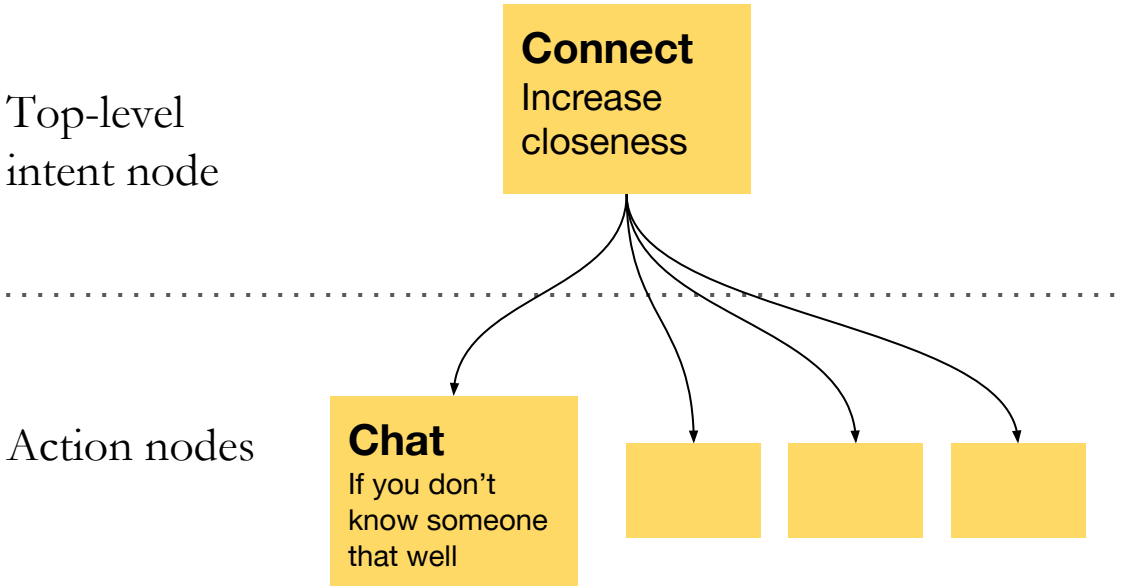
Always required	**Sometimes required**, Depending on where the node is in an action graph
✔ "name" ✔ "influenceRules" ✔ "conditions"	✔ "intent" "category" "type" "intentType" "first" ("second") ✔ "leadsTo" "effects" "isAccept"

Action Node Properties

Always required	**Sometimes required**, Depending on where the node is in an action graph
✅ "name" ✅ "influenceRules" ✅ "conditions"	✅ "intent" "category" "type" "intentType" "first" ("second") ✅ "leadsTo" "effects" "isAccept"

Actions

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



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

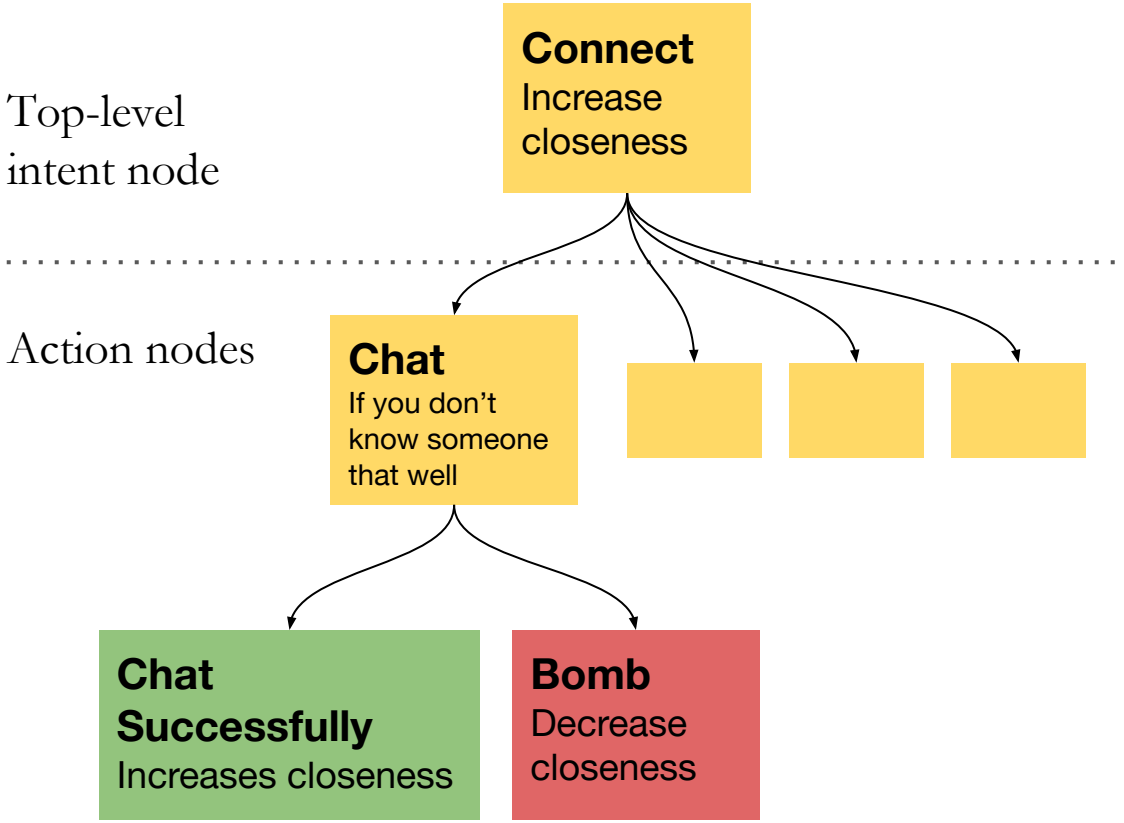
Only difference between action effects and volition effects:

```
{
      "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.



isAccept

`isAccept`

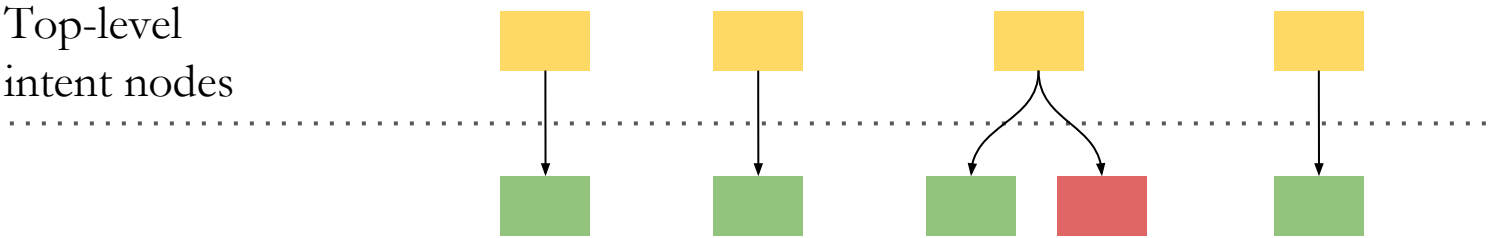
Boolean. Specifies whether this is an action that should be played when the responder accepts (true) or rejects (false). True by default.

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)

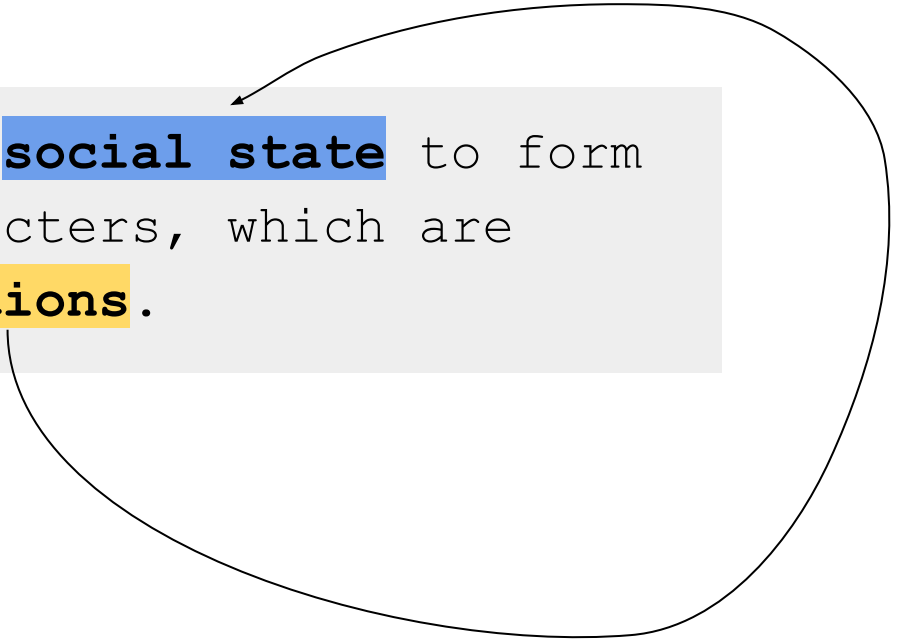
Simplest Action Structure



- 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**.

Trigger Rules

Similar to volition rules:

Set of condition predicates	→	What **volition scores** to change and by how much
Set of condition predicates	→	What **state** to change and by how much

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

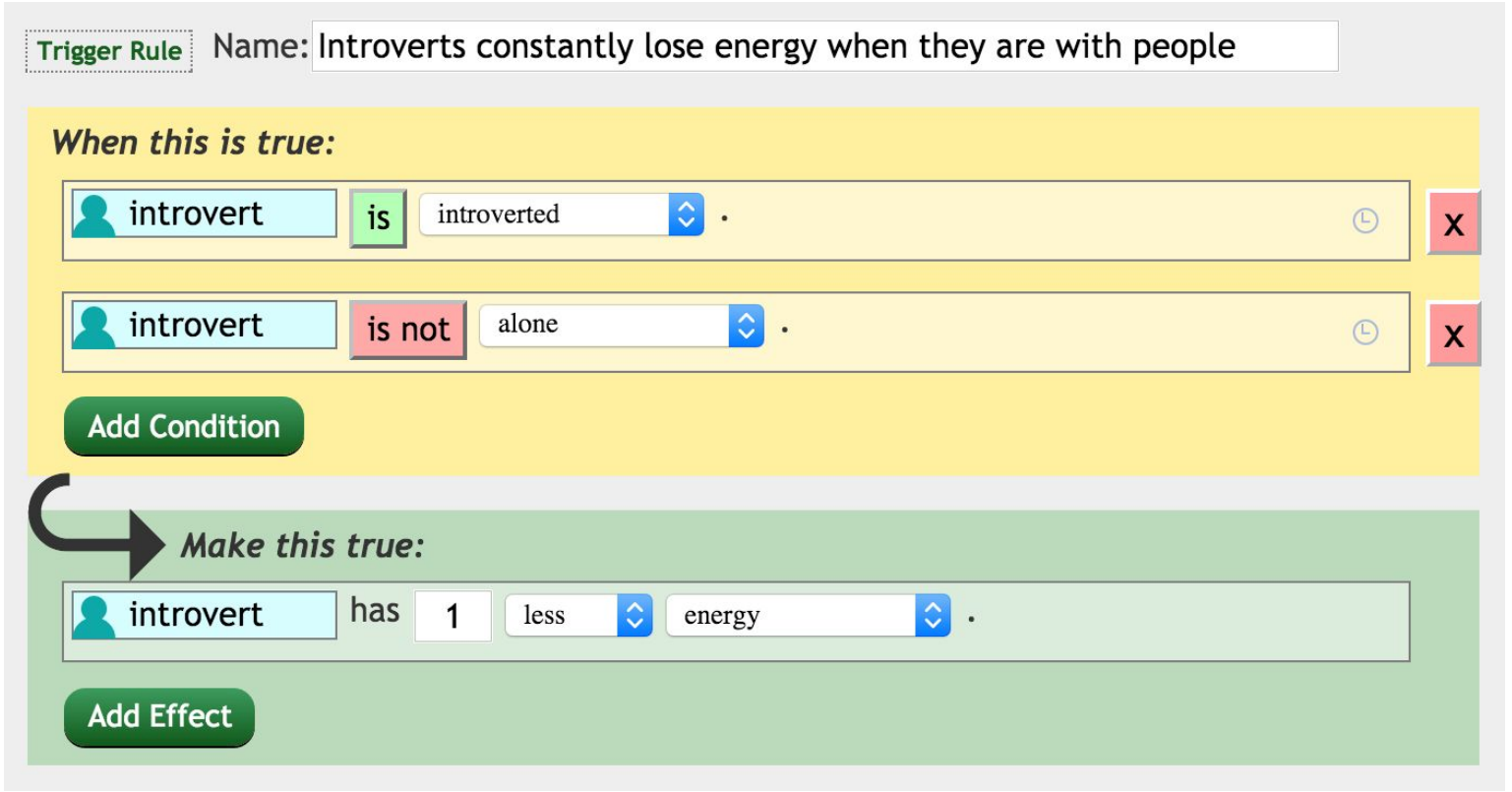
TriggerRules.json

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



That covers the major concepts

Social State

History

Character Desires

Actions

Trigger Rules

Ensemble Engine

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		Calculated Volition		Calculated Action	
JSON Properties and expected values or types	name	String	category	String	name	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] ↓
Legend:			isBoolean	boolean					influenceRules	[influenceRule] ↓			first	String (from Cast)	influenceRules	[influenceRule] ↓
Required property			actionable	boolean					*leadsTo*	*[String]*			*second*	*String (from Cast)*	effects	[effect] ↓
Optional property			defaultValue	boolean, number					*effects*	*[effect] ↓*			englishInfluences	[englishInfluence] ↓	isAccept	boolean
↓ = Def'ed below			*minValue*	*number*					*isAccept*	*boolean*			weight	number	lineage	String
, = or			*maxValue*	*number*											goodBindings	[binding] ↓
			duration	*number*											weight	number
															salience	number
Sub-Object					**Condition**		**Condition**		**Condition**		**Predicate**		**Calculated Influence**		**Condition**	
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)*	weight	number	second	String (from Cast)
					operator	>,<,>=,<=,==	*operator*	>,<,>=,<=,==	*operator*	>,<,>=,<=,==	value	boolean, number			operator	>,<,>=,<=,==
					value	boolean, number	value	boolean, number	value	boolean, number					value	boolean, number
					Effect		**Effect**		**Effect**						**Effect**	
					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)
					intentType	boolean	intentType	boolean	intentType	boolean					intentType	boolean
					first	String (from Cast)	first	String (from Cast)	*first*	*String (from Cast)*					first	String (from Cast)
					second	*String (from Cast)*	*second*	*String (from Cast)*	value	boolean, number					value	boolean, number
					weight	number	value	boolean, number	*operator*	"+","-"					operator	"+","-"
							operator	"+","-"	**InfluenceRule**						**InfluenceRule**	
									name	String					name	String
									conditions	[condition] ↑					conditions	[condition] ↑
FYI, the authoring tool generates some fields and values for file I/O (e.g., origin, id). Those are safe to ignore.									weight	number					weight	number
									Intent						**Binding**	
									category	String (from Schema)					String (from Actions)	String (from Actions)
									type	String (from Schema)					...	
									intentType	boolean						
									first	String (from Cast)						
									second	*String (from Cast)*						

@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

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!

More info, but in academic paper form:

aaronareed.net/2015/Ensemble-Engine-FDG2015.pdf

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

Categories and Subject Descriptors

General Terms

Keywords

1. INTRODUCTION

@meldckn