Kismet: A Small Social Simulation Language

Adam Summerville[a], Ben Samuel[b]

[a]California State Polytechnic University
[b]University of New Orleans

Abstract

Social simulation has been a popular domain in computational creativity for decades. However, while it has been used in applications that are easily digestible by end users (e.g., stories, games, theatrical performances, audio plays), it has typically not been modifiable or authorable for people who are not the original developers. Towards addressing this, we present *Kismet*, a small social simulation language. While *Kismet* is not as powerful as other social simulation approaches, it leverages computational machinery (such as an inheritance system) and is authored using natural language inspired syntax that is designed to be end user facing. The ultimate goal of *Kismet* is to facilitate the authoring of scenario content modules, such as those used in table-top role-playing games.

Keywords

artificial intelligence, social simulation, domain specific language

1. Introduction

Since Meehan [1] developed TALE-SPIN, social simulation has seen a number of different applications for creative ends. These creative applications have ranged from games [2, 3, 4], to theatrical performance [5], to podcast audio plays [6].

The use of artificial techniques to enhance and improve table-top role playing games (TTRPGs) is a recent development, with approaches ranging from using constraint satisfaction to perform character and relationship construction [7] to the recommendation of music dynamically during a session [8]. However, to the authors' knowledge, *Bad News* [5] represents the only TTRPG (if it can be considered that) that uses social simulation. While *Bad News* is an interesting experience and experiment, it is not approachable to lay players in the way that something like *Dungeons & Dragons* [9] is. The game requires a dedicated actor and "wizard" to run, and there is no way for a player to make their own scenarios / modify the simulation in any way.

Toward the goal of making an approachable social simulation system to support other creative endeavors (TTRPGs, videogames, etc.) that are authorable and modifiable by people who do not have (or are in the process of acquiring) Computer Science Ph.D.'s focused on artificial intelligence we present *Kismet* – a small social simulation language. *Kismet* is developed with an eye towards languages like *Tracery* [10] and *Inform 7* [11], as well as the casual creator framework [12]. We note that *Kismet* is not as complex and deep as previous social simulations, but this is a feature, not a detraction. *Kismet* is designed for small social simulations, at a relatively low level of fidelity, and it is designed in a way such that authoring new rules for

Joint Proceedings of the ICCC 2020 Workshops (ICCC-WS 2020), September 7-11 2020, Coimbra (PT) / Online
asummerville@cpp.edu (A. Summerville); bsamuel@cs.uno.edu (B. Samuel)

it is relatively simple. It does not require a specialized tool to author for [13], nor does it require writing in a full-fledged programming language [14] – instead the language has simple, expressive syntax that can be written in any text editor – which is later compiled before simulation occurs. Though authoring syntax via text-editors might give the impression that *Kismet* is designed for power users, the authors believe that its syntax is far less intimidating than the typical high-level computer programming language.

In this way, *Kismet* serves as a step towards accessible—yet expressive—social simulation. Its natural language inspired syntax excels at allowing authors to define the types of actions that simulated characters can engage in, and the social, cultural, and personal influences (referred to in the system as *proclivities*) that might sway a character to select any given action over another. Running the simulation yields sequences of character actions that are locally believable (i.e., any given action makes sense), though might lack global coherence (i.e., the system has little machinery for maintaining direct cause-and-effect relations between actions). However, when used in conjunction with an automated story-sifter or a human storyteller as typically found in table-top role-playing games, Kismet's action sequences become the atomic raw material that other systems—human or otherwise—can embellish upon. Though not discussed in this paper, exciting future work remains to explore Kismet's affordances as a tool for co-creation and shared authorship.

In the rest of the paper, we first discuss *Kismet* in context with existing social simulation approaches. We next describe its approach to simulation at a high level. Finally we delve into the syntax and semantics of the language with some worked examples.

2. Related Work

We motivate this work by providing an overview of inspirational previous work, and discuss how it helped inform the development of Kismet.

2.1. Social Simulations

Kismet is meant to be a simple way to author and represent social simulations. By social simulations, we refer to representing agents (typically affectionately known as Non-Player Characters or NPCs) whose behavior is at least in part modeled by social considerations. There have been many previous explorations of social simulation research with a diverse range of application areas.

Social simulation techniques have been applied in many serious games projects, or other applications with pedagogic or training goals. Stacey Marsela's *PsychSim* [15] was an early example of providing authors with an interface to be able to cast NPCs in particular social roles with sets of behaviors associated with each.

Social relationships are at the heart of many narratives, and as such social simulation is frequently a technique used to create or foster dynamic and compelling interactive emergent narrative experiences [16]. Because emergent narrative is, by definition, not explicitly embedded into the experience itself, a new technique known as story sifting [6] is actively being explored to discover latent narratives in generated content, such as the *Felt* system [17], or through

a process of evaluating dynamic games based on a player's anecdotal retelling of gameplay experiences [18].

These too have frequently been made with pedagogic aims; *FearNot!* [19] and the SIREN project [20] both were developed with the goal of promoting conflict resolution (i.e., peaceful resolutions to bullying) for elementary school aged children. Social simulation systems also drive story experiences designed for entertainment. Emily Short and Richard Evans' Versu System [21] was used to create the ambitious *Blood and Laurels*. Though not designed specifically for social simulation, Mateas and Stern's A Behaviorial Language (ABL) [22] was used, in conjunction with drama management techniques [23], to guide the behavior of the characters of the interactive drama *Façade* [2].

Indeed, some interactive experiences revolve entirely around playable social interactions. Some games, such as the wildly popular *Sims* [24] series and the research game *Prom Week* [3], are entirely about managing aspirations and social relationships of virtual characters. In particular, *Prom Week*'s "social physics" engine, Comme il Faut [13] its successor, Ensemble [25], and their metaphors have been employed towards a variety of ends, including as a "good stranger" training tool for soldiers deployed overseas [26], for cross-cultural competency [27], and has been integrated into popular games like *Skyrim* through modding [4, 28].

In comparison to the information-dense simulation system of *Prom Week*, which made use of thousands of "social influence rules" to power its characters, or the powerful affordances of ABL to simultaneously enforce *Façade's* strong narrative while adapting to player input, *Kismet* is very simplistic. However, we claim that this simplicity is one of Kismet's greatest strengths, and certainly what makes it well suited for use in casual creator systems more so than powerful-but-weighty systems as CiF and ABL. Although *Kismet* has no mechanisms for ensuring satisfying dramatic arcs, it allows for users to quickly and easily spin up and simulate virtual characters governed by simple social laws. "Good" story content can be divined either through the use of story sifters, or by embedding *Kismet* into a larger playable experience. And though tools like *Ensemble* permit a rich interplay of individual social forces, it typically demands a significant authoring effort before character's dynamic behaviors begin taking shape. Though driven by less sophisticated motivations, characters in *Kismet* can begin acting believably with a fraction of the initial authoring effort, resulting in a much less daunting barrier to entry for the end user.

Though *Kismet* does not seek to aim to emulate the fine degree of expressivity of these other systems, other inspirations from these systems remains as future work. Namely, many of these existing systems have companion authoring tools that enable non-programmers to create content for them. *Kismet* has no such authoring tool, but developing one would eliminate the need to author syntax by hand, and would no doubt be a boon to its use as a casual creator for non-technical users.

2.2. Small Casual Creator Languages

Despite the popularity of the casual creator framework, there has been a relative dearth of languages devoted to use by casual creators. *Tracery* [10] is a prime example as a language designed for casual creators and has seen wide spread use via *Cheap Bots, Done Quick!* [29]. *Tracery* is a context free grammar writing language that allows for creators to quickly generate procedural text. Similar to *Tracery* is *Expressionist* [30] – another context free grammar writing

language, however a key difference between *Expressionist* and *Tracery* is that *Expressionist* is designed to be used with an associated authoring tool, while *Tracery* has no such barrier to entry and is designed for any text editor. Though *Kismet*'s current output is purely textual, grammar-based casual creator languages such as *Context Free Art* [31] have successfully been developed in the service of facilitating procedurally generated graphical art.

Given the lack of casual creator languages, it is hard to draw many conclusions, but a common thread is the use of natural language syntax – used in both *Inform 7* [11] and *Imaginarium* [32]. *Inform 7* is a language/tool for authoring of parser based interactive fiction that uses a very naturalistic approach. E.g., "A cheerio is a kind of thing. There are 20 cheerios in the couch." is valid *Inform 7* syntax. *Imaginarium* is a language/tool for constraint based random generation that uses natural language syntax. E.g., "Persian, tabby, and Siamese are kinds of cat. A cat can be large or small. Imagine a cat." is valid syntax that will procedurally generate a cat. While *Kismet* does not fully use natural language, there are aspects that do utilize it, with the hope being that it leads to more readable/authorable/moddable code.

2.3. AI For Table-Top Experiences

Table-Top Role Playing Games (TTRPGs) have recently become a focus for artificial intelligence techniques. *Bardo* [8] is a system that listens to TTRPG sessions and attempts to play fitting music based on the emotional context of the current scene. E.g., if the players are in battle with a dragon, it would try to play something fittingly action packed and epic, whereas if the players are strolling through a forest, it might try to play something soothing and calm. Fiascomatic [7] uses constraint satisfaction to generate characters and relationships for games such as *Fiasco* – one shot TTRPGs that rely heavily on the construction of characters and their interpersonal relationships. *Dear Leader's Happy Story Time* [33] is a completely artificial intelligence driven role playing experience that generates characters, settings, and story beats that players then act out using a constraint satisfaction system. *Dear Leader's* is a definite inspiration for *Kismet* and the kind of experience it could enable; However, *Dear Leader's* content is written in ProLog, not making it suitable for most end users to write/mod content.

We imagine one way that *Kismet* will be a valuable contribution to AI for TTRPGs is in its capacity to facilitate co-creation. TTRPG game masters can author *Kismet* content scenarios to generate small worlds with social dynamics, suitable for quickly creating environments replete with characters, relationships, and backstories that players can explore. Though not a traditional TTRPG, mediated exploration of generated worlds such as this is akin to the experience *Bad News*, which we describe below.

2.4. Bad News

Perhaps *Kismet*'s greatest influence is *Bad News* [5], which leverages the Talk of the Town simulation engine [14]. *Bad News* is a computationally assisted performance [34]; an installation piece which combines procedural generation, social simulation, and live performance. In *Bad News*, Talk of the Town simulates nearly a century and a half of quotidian life in a generated, small American town. A performance of *Bad News* primarily revolves around a player engaging with a human actor, who throughout the performance may potentially portray dozens of the

hundreds of virtual denizens of the town. All the while, another human performs the role of Wizard (so named after the Wizard-of-Oz experiments of the projects' origin [35]), who performs manual story sifting to discover interesting latent generated narrative and suggests possible moves the actor might make to facilitate the player discovering them. In addition, a third member of the performance serves as the Guide, to help acclimate the player into the world of the game, as well as explain the underlying performance and simulation to audience members while the actor and Wizard are occupied with the player.

This piece has been performed internationally, at game festivals, film festivals, computer science conferences, universities, and the San Francisco Museum of Modern Art. It has won awards at both the ACM SIGCHI conference and the IndieCade festival of independent games. It appears to be an experience that captures the imagination on both a technical and emotional level. However, one of it's greatest strengths—the spectacle of a computer simulation and three humans in the roles of Actor, Wizard, and Guide creating and portraying a unique, ephemeral world for each player—is also one of its greatest limitations. The game is an installation piece, and as such requires a hefty amount of logistics to be performed, severely limiting the number of people who could ever experience it.

We hope that *Kismet*'s potential as a co-creative partner can serve as a first step towards a "*Bad News: The Home Game.*" There are many challenges to creating such a system. However, giving players the ability to quickly and easily develop their own social simulation modules is an important first step to achieving it. To do so will allow players to personalize their simulated worlds with virtual denizens that think about, care about, and behave in accordance to values aligned with the types of story-worlds any given end user hopes to capture.

3. Kismet

The goal of *Kismet* is to be a small language that supports small social simulations. The core entities of the *Kismet* social simulation are *Characters* and *Locations*.

Unsurprisingly, characters are the most important entity found in *Kismet*. A *Kismet* character is a set of *traits* and *statuses* that influence their actions and the actions that others take towards them. Traits and statuses are effectively the same thing – a single predicate that is attached to a character that influences how likely they are to take certain actions – with the key difference being that traits are inherent to a character (i.e., they are attached to the character when the character is instantiated) while statuses are (and must) be applied to a character during the simulation.

Traits come in two varieties – *default* and *random*. Default traits are traits that are inherent to all characters in a simulation. These can be thought of as encoding societal norms – e.g., people are generally not rude to each other, or people avoid incestuous relationships. Random traits are the spice of the simulation – they provide different behavioral patterns that let characters have personality and differentiation – and each character is assigned a random number of them. This random number is configurable by an end user / module author, but in practice we have found three to five to be an effective number that allows for enough personality, while still being understandable by an end user.

Some traits do not make sense in conjunction with each other – e.g., a character can not

be both a drunkard and a teetotaler or a character can not be both serene and rage-filled simultaneously. To account for this, traits can be stated to be in opposition to each other, so that a character can have only one of those traits.

The other major class of entity are locations. Locations are where the simulation takes place. Each location has a number of roles that the location supports, each of which can support a different number of characters. E.g, a grocery store might have one manager, between three and five grocers, and up to 30 patrons. Some roles are *cast* upon instantiation of the location (which is the instantiation of the simulation) while others are cast and recast every tick of the simulation. Some actions are only available to certain roles, and only when the character enacting that role is at the location where they enact that role – e.g., a character might perform the action "tend bar" if they are enacting the role of bartender at the bar where they work.

To save on duplication of effort, roles have a form of inheritance. Roles can inherit all of the preconditions and tags of their ancestors, minimizing the burden of authoring many different roles. E.g., the role of `bartender` might inherit from `standard-labor` which inherits from `labor`. `labor` might have the tag `work` which `standard-labor` and `bartender` both inherit. Consequently, perhaps `standard-labor` might have the precondition that the character enacting the role must be older than 18 years old. This allows for a split in the inheritance of `labor` where `child-labor` inherits from `labor` but requires the character be less than 18 years old.

These pieces come together in the simulation. At initialization, a number of characters are instantiated – and their personality traits are chosen. Then the locations are spawned – each of which cast their initial roles. During each tick of the simulation, the characters choose which location they should go to. Once there, they choose which action to take based on their traits, statuses, and the other characters (who may be colocated, located elsewhere, or unspecified).

The *actions* are the most important aspect of the simulation. They are how the characters and their relationships evolve during the course of the simulation. It is the history of these actions and the resultant relationships that form the material that end users mine from. Each action has a number of *tags* that describe how the action should be interpreted by the characters. E.g., the action `gossip` might have the tags `talk` and `rude`. A character with the trait `introvert` might be less likely to take `talk` actions, so they would be less likely to take the `gossip` action. Similarly, a `polite` person may be less likely to take `rude` actions. Thus, a `polite introvert` would be highly unlikely to `gossip`, while an `impolite extrovert` would be more likely to take the action.

While the tags – in concert with the character traits – affect how likely a character will be to take a certain action, there are also binary conditions that affect whether a given action can even be attempted. Actions all have an enactor that needs to be cast, but they can have any number of targets – those who the actions are performed upon – and subjects – those who the actions are about. E.g., the `gossip` action would have the enactor `gossiper`, target `gossipee`, and subject `gossiped-about`.

The action also has a set of preconditions that must be met for the action to be possible. These conditions can be of:

- Unary Arity – is a fact true about a single character – e.g., A is `drunk`
- Binary Arity – i.e., is there a relation between two characters – e.g., A `likes` B
- Arbitrary Arity – is there a relation that holds over multiple characters – e.g., `love-triangle`(A,B,C)

All of the preconditions must be true, i.e. they are all connected via logical and. To author an *or*, one would need to create two nearly identical actions that differ only in those preconditions. The results of the action are two-fold. First, the action occurs and is recorded in the history of the simulation – which can be used by end users as well as the simulation to check for certain relationships. The actions are also observed by bystanders, based on how noticeable they were. Two characters chit-chatting is not particularly engaging, so it might not be observed, while two characters getting in a fistfight is going to be observed by everyone at the same location. Second, the actions can *add* or *delete* statuses and relations. E.g., the action `drink` might apply the status `drunk` which can have downstream impacts, while the action `sober-up` might remove the `drunk` status. Similarly, the action of A `flirt`ing with B might add the relation B `likes` A, while the action `break-up` might remove the relation `dating`.

The aim of these small simulations is to provide a scaffolding for further downstream processes (e.g., Table-Top Role Players) to embellish upon. This leads to a design decision where the characters have somewhat caricaturish personality traits – e.g., a drunk is much more likely to go and drink at a bar – and the actions that the characters take are to be seen as simplified stand-ins – e.g., "drink at bar" should not be interpreted as a person getting a single drink, but rather could be interpreted as "wasted the afternoon getting drunk at the bar." Furthermore, the goal is that the language allows for end users to modify scenario modules, and possibly create their own. To this end, the language has been designed to be approachable, using some standard programming syntax, but more heavily inspired by natural language approaches like *Imaginarium* and *Inform 7*.

3.1. Language Description

In this section we will describe the language – first starting with some syntax. Some notes about *Kismet* syntax: (1) variables are upper cased, (2) definition names are lower cased, (3) lists are separated with commas (4) the components of a definition are separated with semi-colons , and (5) definitions are ended with periods. Argument lists of characters use special syntax to define who is the enactor (`>`), the target (`<`), the subject (`^`), or an action (`*`). Also, anything in bold below is a keyword in *Kismet*.

Random selections are contained in square brackets:

```
[ ]
```

These can be ranges of natural numbers:

```
[0 - 10]
```

or text strings separated by `|`'s

```
[heads | tails]
```

Text strings can reference Tracery grammars (in separate files) using the standard hashtag syntax:

```
[#coin toss# | #dice roll#]
```

For ranges of numbers users can specify the shape of the probability distribution using the symbols `_ - ^`. `[0-100]_-^-_` is approximately a normal distribution while `[0-100]_-^` has increasing probability for selecting larger numbers.

A location is specified with the **location** keyword:

location LOCATION-TYPE:
  supports: LIST OF ROLES;
  name: TEXT;
  initialization: LIST OF CAST;
  each_turn: LIST OF CAST.

A role is defined as the number of that kind of role that the location supports as well as the name of the role:

```
[NUMBER] NAME
```

The cast commands are the **cast** keyword, the role to cast, and the number of characters to cast in that role. **initialization** is for the roles that are cast when the location is created, and **each_turn** is for the roles that are cast on each turn of the simulation. Putting this all together a description of a bar that has an owner, a few bartenders, and a number of patrons might look like:

location bar:
  supports: `[1] owner, [2-3] bartender, [1-10] patron`;
  name: `"The #adjective# #animal#"`;
  initialization:
    cast `[1] owner`,
    cast `[1-2] bartender` ;
  each_turn:
    cast `[1-10] patron` .

Roles are defined with an optional extension of another role, a set of tags that the role embodies, and the preconditions required for a character to be capable of taking on that role.

role ROLE-TYPE(CHARACTER)
    [**extends** ROLE-TYPE]:
  tags: LIST OF TAGS;
  if: LIST OF PRECONDITIONS.

E.g., to represent the concept of having a job – and only being allowed to have a single job at a time – and that bartending is a kind of job one might use the following roles:

```
role job(>Worker):
  tags: labor;
  if: Worker is missing job.

role bartender(>Worker)
     extends job(>Worker):
  tags: drinking;
  if: Worker.age >= 18.
```

Traits have the name (or names) of the trait and the proclivities that the trait entails. Optionally, the trait can be flagged as being *default* and/or the trait that is opposition to the trait:

```
[default] trait NAMES [opposes NAMES]:
  LIST OF PROCLIVITIES.
```

A proclivity is represented as the impact to how much more or less likely a character is to take an action based on the tags associated with the action. These proclivities can also be conditioned on traits, statuses, and relations between characters:

```
VALENCE(LIST OF TAGS
        [if LIST OF CONDITIONS])
```

E.g., to represent the fact that there are extroverts and introverts who are more or less likely to talk than others, respectively, one might write:

```
trait extrovert/extroverted
       opposes introvert/introverted:
  +++(talk).
```

Opposition traits are defined as having the opposite valence, so an identical way to represent this would be:

```
trait introvert/introverted
       opposes extrovert/extroverted :
  –(talk).
```

The strength of the proclivity is represented by the number of pluses or minuses. All actions are assumed to default to a score of 0, but for all engaged proclivities the score is modified. Traits and statuses can work in concert, or – alternatively – can counteract each other. By default, an action with two pluses is twice as likely to be selected as one with no pluses, but the temperature of the selection (as in simulated annealing) can be set an end user or content module author to make their characters more or less predictable (lower or higher temperature,

respectively). The multiple names delimited with slashes is a bit of syntactic sugar that allows an author to use multiple names to represent the same concept.

An example of a default trait that all characters would have might be that people are more likely to do nice things (and less likely to do mean things) to people that they like:

```
default trait kind-to-people-i-like:
  +++(nice if >Self likes <Other).
```

This uses a conditional proclivity, such that the odds of doing a nice action towards someone only increase if the enactor likes the target. The greater than and lesser than symbols tell the system how to bind to the characters in an action.

Next, we come to actions. Actions have a name, an argument list of involved characters (and possibly historical actions), text for how the action should be descibed, how (if at all) the action extends another, a list of tags, constraints on the location(s) of the characters, the visibility of the action, the preconditions for the action, and the results of the action.

```
action NAME(LIST OF CHARS)
    [extends ACTION]:
  "TEXT"
  location: LIST OF LOCATIONS;
  tags: LIST OF TAGS;
  if: LIST OF PRECONDITIONS;
  result: LIST OF RESULTS;
VALENCEvisibility.
```

E.g., a simple action like chit-chatting might look like:

```
action chit-chat(>Chatter,<Listener):
  "Chatter chats with Listener"
  location: (Chatter, Listener);
  tags: talk.
```

Which means that chit-chat is a `talk` action, and that the participants must be co-located (but that otherwise there are no constraints on their location).

An action can also be tied to specific roles, in which case the action must take place at the location where those people enact those roles. E.g., a bartender might "pour the troubles away" for a sad patron:

```
action pour-their-troubles-away
    (>Tender:bartender,<Patron:patron):
  "Tender listens to the troubles
    of Patron and pours them a drink"
  location: (Tender, Patron);
  tags: work, drinking, talk, nice;
  if: Patron is sad;
  result: Patron likes Tender,
```

```
    Patron is drunk.
```

Note, this action can only occur at a bar, since that is where the Tender is enacting the role of bartender.

Actions can also have different levels of visibility – e.g., everyone is aware that a fistfight occurred in the same location.

```
action fight(>Fighter,<Fightee):
  "Fighter and Fightee come to blows"
  location: (Fighter, Fightee);
  tags: scandalous,violent, angry);
  ++++visibility ;
  result:
    Fighter and Fightee
      dislike each other,
    Fighter and Fightee
      do not like each other.
```

Two notes, (1) A and B C "each other" is syntactic sugar for A Cs B and B Cs A, (2) "do not" removes the relationship (or fact) if it exists. We can create a more specific version of the action if we want quite simply:

```
action bar-room-brawl(>Fighter,<Fightee)
 extends action fight(>Fighter,<Fightee):
  "A barroom brawl breaks out
    between Fighter and Fightee"
  location: bar(Fighter, Fightee);
```

The primary difference between bar-room-brawl and fight is that bar-room-brawl specifies that the location that the fighters are in must be a bar. The two actions also have different descriptions.

Finally, since the fight is a very visible, scandalous action, it might be gossiped about by people who observed it:

```
action gossip(>Gossiper,<Gossipee,
  Ŝubject,*Event):
  "Gossiper gossips to Gossipee
    about Subject"
  tags: talk, rude, nosy;
  if:
    Gossiper knows Event,
    Gossiper did not do Event,
    Gossipee did not do Event,
    Gossipee did not receive Event,
```

```
    Subject did Event,
    Event is scandalous
  result: Gossipee heard Event.
```

Knowledge about actions can be *seen* (directly observed), *heard* (received second-hand), *did* (the enactor of the action), *received* (the target of the action), *known* (any of seen, heard, did, or received), and *forgotten* (the character no longer knows about the action). The gossiping makes sure that people don't gossip about their own or their target's scandalous exploits. Further, the tags of the action can be reasoned about as one would the status, traits, and relations of characters.

Finally, there are a syntactic sugaring shorthand *patterns* that can be used in actions, traits, and other patterns – easing the authoring burden.

For instance, a large portion of the above gossiping action's preconditions could be summarized as "A and B weren't involved in a scandalous event". One might find themselves using that pattern in other actions, so they could specify:

```
pattern not-involved-in-scandal(>A,<B,*E):
  if:
    A did not do E,
    B did not do E,
    B did not receive E,
    E is scandalous.
```

Which could then be reference elsewhere. These can also be exposed to an end user, as a way to highlight potentially juicy tidbits:

```
pattern love-triangle(A,B,C):
  if:
    A loves C,
    B loves C.
```

or

```
pattern unrequited-love(A,B):
  if:
    A loves B,
    B does not love A.
```

3.2. Implementation

Kismet is parsed using ANTLR4 [36]. The preconditions for the actions, patterns, traits, and statuses are then compiled to AnsProlog. Once in AnsProlog, Clingo [37] is used to calculate which actions are possible and how likely a character is to take the action. This is done by

combining the event history, the current social state (composed of the traits, statuses, and relationships of the characters), as well as the authored rules and solving with Clingo. First, the proclivities are calculated to determine how likely a character is to go to each location. These are then sampled for each character after being transformed into a probability distribution via the softmax function:

$Pr(l_i) = \frac{e^{w_i/T}}{\Sigma e^{w_j/T}}$

Where w_i is the weight assigned to going to location l_i. Once characters are at a location, Clingo is once again used to determine the proclivity weights, and an action is sampled for each character as with the locations. Once the actions are selected, their results are applied and the current social state is updated. This process proceeds until one of two things occurs – (1) most simply until a certain number of steps has occurred or (2) until a set of conditions is met – either of which can be set by a content module author or the end user.

4. Limitations and Future Work

Although a formal evaluation of *Kismet* has yet to take place, some preliminary testing has transpired with an undergraduate research assistant who used *Kismet* to create a world of responsible business employees and casino frequenting layabouts.

The "experimental set-up" of this preliminary work is simple to explain: this student is a sophomore in computer science, and had never been exposed to many of the underlying principles of Kismet—such as Clingo, answer set programming, or the very notion of a domain specific language—prior to working on this project. However, they did have a background in the Java programming language, and general approaches to procedural thinking. *Kismet*'s abstractions away from it's underlying processes made it possible for the undergraduate to design and create a social world, which is heartening. At the same time, their attempts to use the system highlight some of its limitations, both in terms of expressively and accessibility. Although perhaps having some programming experience is a requirement for the "target user," a mark of a successful casual creator is a tool which is easy to jump into. Thus, points of confusion this undergrad experienced highlight potential problem areas that future users might encounter as well, and warrant our attention moving forward.

Some of the issues the undergraduate encountered pertained to simple syntax misunderstandings, though some of these spoke to more fundamental conceptual issues. For example, the first set of actions written had no post-conditions specified, indicating a disconnect between recognizing a character taking an action and the resulting underlying social state changes.

Other questions frequently revolved around the notion of "roles" in the system. For example, in example code included in the *Kismet* documentation, it introduces a location called a bar, and says that it has one owner, two to three bartenders, and one to ten patrons. The roles of "owner" and "bartender" speak to one's profession, and have an air of permanence to them; e.g., even as the person cast as the owner lives their life and visits other locations, say, goes grocery shopping, they are still a bartender even as they are a patron of the market. However, the role of patron speaks to something much more ephemeral; even though one might always consider themselves a patron of a spot they frequent often, it is likely less a central part of one's identity than the occupational roles such as bartender or owner would be. This distinction

between roles referring to professions and roles referring to momentary qualifiers boils down to differences between the "initialization" specification of a location, and a location's "each_turn" specification, as described above, but this distinction was a challenge to understand.

Other limitations centered around the notion of tags. One such limitation is that there is currently no way to mark a duration of a tag. For example, the undergraduate wished to create a tag to mark a character as being "recently promoted." This should heavily influence behavior in the short term (e.g., celebrating during the week of the promotion) but should not influence behavior in the long term (e.g., influencing the character's desire to celebrate years after the promotion feels incorrect). At present, the way this is handled is to have certain actions remove tags. As previously mentioned, the "sober_up" action could remove the "drunk" tag; the tag will continue to influence the character's behavior until they happen to take the aforementioned action to remove it.

Another limitation of tags is that they they help influence the behavior of those who wish to take actions, but they do not influence the selection of other non-initiating characters in the action. This can be circumvented, however, through clever use of traits. Though the "+" and "-" syntax is still applying from the enactor's perspective (i.e., making them more or less inclined to take an action), it is possible to write traits such as:

```
default traitpromote_hardworkers:
  +++(promotion
    if <Other is hardworking).
```

where hardworking is a tag here applying to a character. Still, the +++ is applying specifically to the enactor; they are more likely to engage in promote actions if someone around them is hardworking.

Shifting away from tags, the undergraduate was curious about capturing a sense of operating hours for certain locations. In *Bad News,* which offered much inspiration for *Kismet*, each simulated day was split into two time blocks: day and night. Any given business operated either during the day, or the night, or both day and night. At present, the only sense of time in *Kismet* is the granularity with which an author specifies actions. However, there are currently plans to implement time cycles. A simple time cycle might look like:

```
time cycle[am,pm]
```

Which would cycle between morning and night, and then could become conditions for certain behaviors, e.g.,

```
if time is pm...
```

This could then be extended to nested time cycles, e.g.,

```
time cycle[m,t,w,th,f,sa,su][am,pm]
```

Which would cycle through monday am, monday pm, tuesday am, tuesday pm, wednesday

am, until eventually wrapping back around to monday am. These cycles could be extended indefinitely (e.g., capturing four weeks in a month, twelve months in a year, etc.).

Another desire the undergraduate had was to design situations in which characters could either choose to go to work or play hooky. Although this could be achieved through traits and actions (e.g., a trait "diligent" that increases the likelihood of taking "go_to_work" actions and an equivalent "lollygagger" trait which reduces the likelihood), this still demands that any given "work" action is then explicitly written and given the "go_to_work" tag. Another piece of future work is to implement prototypical actions (such as a generic "work"). Connected to the notion of inheritance described above, prototypical actions would not be directly selectable by characters to perform but could be extended to subactions which themselves are selectable. This ensures the tag only need to be specified once for ease of authoring and revising, such as:

```
prototype action work:
  tags: go_to_work.
```

ensuring that every action which inherits from work will have the `go_to_work` tag, without it needing to be specified for each individual action.

An exciting next step is to begin making sample modules and playable experiences using *Kismet*. This process will undoubtedly lead to accessibility improvements, which in turn will further enable *Kismet* to be adopted as a casual creator tool.

Additionally, formal user studies with additional participants with a variety of backgrounds would further substantiate the preliminary heartening observations that *Kismet*'s abstractions permit for non-technical authors to utilize *Kismet* to create meaningful or otherwise interesting stories. To this end, additional authoring support such as an authoring tool would provide valuable scaffolding to introduce newcomers to the language. Lastly, a user study that compares *Kismet* to other social simulation systems, such as *Ensemble*, could be extremely valuable, and would validate (or invalidate) the author's belief that Kismet permits users to generate small but dynamic social worlds at far quicker rate than its contemporaries. Such a study could begin to further explore the trade-offs between accessibility and expressivity in social simulation systems, and in so doing not only inspire future directions for *Kismet*, but help establish goals and baselines for any future systems that may be developed.

5. Conclusion

Kismet is a small language for small social simulations, with an eye towards end user authorability. It is not as powerful or as in-depth as other expressive social simulation, but it is the first that has a focus on language design for novice use. It is still relatively early in development, with additional language constructs planned. Furthermore, our belief that *Kismet* is easier to use and author than other social simulations is mostly untested. In the future, we would like to get *Kismet* into the hands of more novices to be able to test this, to examine the range of social simulations they create.

References

[1] J. R. Meehan, Tale-spin, an interactive program that writes stories, in: Proc. of the 5th International Joint Conference on Articial Intelligence, Aug. 1977, volume 1, 1977, pp. 91–98.

[2] M. Mateas, A. Stern, Façade: An experiment in building a fully-realized interactive drama, in: Game developers conference, volume 2, 2003, pp. 4–8.

[3] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, N. Wardrip-Fruin, M. Mateas, Prom week, in: Proceedings of the International Conference on the Foundations of Digital Games, 2012, pp. 235–237.

[4] M. Guimarães, P. Santos, A. Jhala, Prom week meets skyrim., in: AAMAS, 2017, pp. 1790–1792.

[5] B. Samuel, J. Ryan, A. J. Summerville, M. Mateas, N. Wardrip-Fruin, Bad news: An experiment in computationally assisted performance, in: International Conference on Interactive Digital Storytelling, Springer, 2016, pp. 108–120.

[6] J. Ryan, Curating simulated storyworlds, Ph.D. thesis, UC Santa Cruz, 2018.

[7] I. D. Horswill, Fiascomatic: A framework for automated fiasco playsets, in: Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2015.

[8] R. R. Padovani, L. N. Ferreira, L. H. Lelis, Bardo: Emotion-based music recommendation for tabletop role-playing games, in: Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference, 2017.

[9] G. Gygax, D. Cook, The Dungeon Master Guide, No. 2100, 2nd Edition (Advanced Dungeons and Dragons), TSR, Inc, 1989.

[10] K. Compton, B. Kybartas, M. Mateas, Tracery: an author-focused generative text tool, in: International Conference on Interactive Digital Storytelling, Springer, 2015, pp. 154–161.

[11] G. Nelson, Natural language, semantic analysis, and interactive fiction, IF Theory Reader 141 (2006) 99–104.

[12] K. Compton, M. Mateas, Casual creators, in: Proceedings of the Sixth International Conference on Computational Creativity, 2015, p. 228.

[13] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, M. Mateas, N. Wardrip-Fruin, Social story worlds with comme il faut, IEEE Transactions on Computational intelligence and AI in Games 6 (2014) 97–112.

[14] J. O. Ryan, A. Summerville, M. Mateas, N. Wardrip-Fruin, Toward characters who observe, tell, misremember, and lie, in: Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference, 2015.

[15] S. C. Marsella, D. V. Pynadath, S. J. Read, Psychsim: Agent-based modeling of social interactions and influence, in: Proceedings of the international conference on cognitive modeling, volume 36, 2004, pp. 243–248.

[16] M.-L. Ryan, From narrative games to playable stories: Toward a poetics of interactive narrative, Storyworlds: A Journal of Narrative Studies 1 (2009) 43–59.

[17] M. Kreminski, M. Dickinson, N. Wardrip-Fruin, Felt: A simple story sifter, in: International Conference on Interactive Digital Storytelling, Springer, 2019, pp. 267–281.

[18] M. Kreminski, B. Samuel, E. Melcer, N. Wardrip-Fruin, Evaluating ai-based games through retellings, in: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive

Digital Entertainment, volume 15, 2019, pp. 45–51.
[19] R. S. Aylett, S. Louchart, J. Dias, A. Paiva, M. Vala, Fearnot!–an experiment in emergent narrative, in: International Workshop on Intelligent Virtual Agents, Springer, 2005, pp. 305–316.
[20] G. N. Yannakakis, J. Togelius, R. Khaled, A. Jhala, K. Karpouzis, A. Paiva, A. Vasalou, Siren: Towards adaptive serious games for teaching conflict resolution, Proceedings of ECGBL (2010) 412–417.
[21] R. Evans, E. Short, Versu—a simulationist storytelling system, IEEE Transactions on Computational Intelligence and AI in Games 6 (2013) 113–130.
[22] M. Mateas, A. Stern, A behavior language for story-based believable agents, IEEE Intelligent Systems 17 (2002) 39–47.
[23] D. L. Roberts, C. L. Isbell, A survey and qualitative analysis of recent advances in drama management, International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning 4 (2008) 61–75.
[24] W. Wright, L. Humble, The sims, EUA: Electronics Arts (2000).
[25] B. Samuel, A. A. Reed, P. Maddaloni, M. Mateas, N. Wardrip-Fruin, The ensemble engine: Next-generation social physics, in: Proceedings of the Tenth International Conference on the Foundations of Digital Games (FDG 2015), 2015, pp. 22–25.
[26] D. G. Shapiro, J. McCoy, A. Grow, B. Samuel, A. Stern, R. Swanson, M. Treanor, M. Mateas, Creating playable social experiences through whole-body interaction with virtual characters, in: Ninth Artificial Intelligence and Interactive Digital Entertainment Conference, 2013.
[27] J. Klafehn, P. Inglese, M. Treanor, J. McCoy, Walking a mile in simulated shoes: Development of an assessment of perspective taking, in: Proceedings of the Eleventh annual MODSIM World Conference, 2018.
[28] B. G. Studios, The elder scrolls v: Skyrim, Bethesda Game Studios, 2015.
[29] G. Buckenham, Cheap Bots, Done Quick!, https://cheapbotsdonequick.com/, 2020.
[30] J. Ryan, E. Seither, M. Mateas, N. Wardrip-Fruin, Expressionist: An authoring tool for in-game text generation, in: International Conference on Interactive Digital Storytelling, Springer, 2016, pp. 221–233.
[31] C. Coyne, M. Lentczner, J. Horigan, Context free art, URL: www. contextfreeart. org (2010).
[32] I. Horswill, Imaginarium: A tool for casual constraint-based pcg, in: Proceedings of the AIIDE Workshop on Experimental AI and Games (EXAG), 2019.
[33] I. D. Horswill, Dear leader's happy story time: A party game based on automated story generation, in: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference, 2016.
[34] B. Samuel, J. Ryan, A. Summerville, M. Mateas, N. Wardrip-Fruin, Computatrum personae: toward a role-based taxonomy of (computationally assisted) performance, in: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference, 2016.
[35] J. O. Ryan, A. J. Summerville, B. Samuel, Bad news: A game of death and communication, in: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems, 2016, pp. 160–163.
[36] T. Parr, The definitive ANTLR 4 reference, Pragmatic Bookshelf, 2013.
[37] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Clingo= asp+ control: Preliminary report,

arXiv preprint arXiv:1405.3694 (2014).