Neighborly: A Sandbox for Simulation-based Emergent Narrative

Shi Johnson-Bey
Computational Media Department
UC Santa Cruz
Santa Cruz, CA, USA
ismajohn@ucsc.edu

Mark J. Nelson
Computer Science Department
American University
Washington, DC, USA
mnelson@american.edu

Michael Mateas
Computational Media Department
UC Santa Cruz
Santa Cruz, CA, USA
mmateas@ucsc.edu

***Abstract*—This paper presents *Neighborly*, a customizable, community-scale social simulation engine for procedurally generating settlements of characters for use in research experimentation or entertainment media. *Neighborly* is a rational reconstruction of *Talk of the Town* (*TotT*), an earlier social simulation for emergent narrative focused on simulating small American towns and the townspeople's lives. Based on *Talk of the Town*'s previous success as part of the experimental game *Bad News*, we wanted to reconstruct it as a general-use social simulation authoring tool. In this paper, we delineate the design space of *TotT*-like social simulation and compare *TotT*-likes to other academic projects and commercial social simulation games. Finally, we provide an overview of how *Neighborly* embodies the essence of *TotT* while offering users a customizable tool for creating community-scale social simulations.**

***Index Terms*—social simulation, procedural generation, autonomous characters, story generation**

I. Introduction

Emergent narrative is a sub-domain of computational narrative (creating stories using computers and computation) focused on generating stories organically from the bottom-up interactions between individual agents and systems [1], [2]. Agent-based social simulation (ABSS) has shown to be an effective tool for creating dynamic emergent narratives in video games. ABSS in games model characters as individual agents with attributes that drive their social interactions/behaviors with other characters and the player. Emergent narratives are then produced from the bottom-up interactions between individual agents. Both academic and commercial games have used ABSS to create unique interactions between non-player characters (NPCs) and players. Two examples of commercial games known for their social simulations are the *Middle Earth™: Shadow of Mordor/Shadow of War* series [3], [4] and *The Sims™4* [5]. The *Middle Earth™* games' core feature is the *Nemesis System* which simulates a dynamic power structure of Orcs vying for dominance and the player's evolving nemesis-style relationships with these orcs. Players engage in "revenge-loop" narratives where they battle orcs over multiple in-game lives and influence the orc power structure based on their victories and defeats. *The Sims™4* [5] gives players the experience of a virtual dollhouse, where players dictate how NPCs go about their daily lives, meeting other characters, forming relationships, gaining skills, and having families. In academia, games such as *Prom Week* re-imagine the goal of becoming prom-king as a social physics puzzle requiring a planned series of social interactions [6]. While these are entirely different experiences, social simulation plays a core role in powering them and providing players with the joys of witnessing emergent scenarios unfold.

Talk of the Town (TotT) was one of the most prominent academic social simulation projects within the past decade [7]–[11]. It simulates a small, procedurally generated American town for over 100 virtual years, starting from its founding in the mid-19$^{\text{th}}$-century. Towns can range from small ghost towns to bustling social centers with more than 200 residents. *TotT* uses a simple model of abstract social mechanics combined with pseudo-randomness and population statistics to produce engaging emergent narratives about star-crossed lovers, generational rivalries, family legacies, and romantic love triangles [1]. *TotT* was used in the experimental game *Bad News* [12], in which the player explores a generated town looking for the next of kin of a deceased NPC. They do this by interacting with various NPCs via a human improv-actor who embodies characters using information generated by *TotT*'s simulation. *Bad News* was featured at Slamdance [13] and Indiecade, and covered in numerous press venues including The Guardian [8] and Rolling Stone magazine [14].

Previous work has utilized *TotT*'s effectiveness as a content generator to power their own experimental games [15] and interactive narrative authoring systems [16]. We wanted to do the same but ran into limitations when trying to customize *TotT*'s narrative setting. Projects treat *TotT* as a black-box content generator. However, we wanted more authorial control over the types of roles NPCs can have, the businesses they can own, and their internal decision-making logic. *TotT* was not designed to be used for anything other than *Bad News*. So it incorporates ad-hoc assumptions about *Bad News*' setting into the core simulation code, offering little in the way of code abstractions and authoring support for modifying and extending the simulation. For example, suppose someone wants to investigate the impact of a new personality model on character behavior and the stories that emerge. *TotT* does not afford avenues to feasibly integrate custom modifications outside of its existing infrastructure. Thus, replacing the Big-5 personality model currently employed by *TotT* would require a

complete rewrite of the relationship model, character routines, and event logic. This tight coupling between different elements of the simulation makes it impossible to experiment with alternative representations, decision-making logic, and narrative settings.

We believe that it would be beneficial to have an authoring tool that enables users to leverage *TotT*'s emergent narrative generation capabilities to power new social-simulation-based player experiences. In this paper, we delineate the design space of *TotT*-like social simulation and compare it to other academic and commercial social simulations. Finally, we provide an overview of our prototype system *Neighborly*, a customizable, community-scale social simulation engine, and rational reconstruction of *TotT*. *Neighborly* embodies the essence of *TotT* while offering users a customizable tool for community-scale social simulation.

II. Background and Related Work

This project was motivated by experiences using *TotT* in previous projects. Our goal was to take advantage of its potential for generating emergent scenarios between characters to experiment with how different simulation parameters and decision-making logic result in different distributions of emergent narrative situations. An earlier project, *Centrifuge*, developed a visual story-sifting language for use in such experimentation [17]. Further, we were interested in exploring the potential of *TotT* to support different media experiences with alternative narrative settings and character decision-making logic.

TotT's representations of town businesses, character occupations, life events, and residences provide an excellent infrastructure for defining narrative settings and situations. However, as we tried to perform such experiments, we ran into the issues of hard-coded specific assumptions that are described in the introduction above. Even relatively simple changes, such as changing the narrative setting from a small American town to a space colony, required significant changes to the simulation code.

This project started as a minor refactor and evolved into a complete reconstruction. In the process of this reconstruction, we identified the core features that define *TotT*-like social simulations (in comparison to other social simulations) and developed a new infrastructure, *Neighborly*, that supports end-user experimentation and authoring. Here we provide the background and related work that define *TotT*-like social simulation, as well as provide some background on rational reconstruction.

A. Talk of the Town

TotT is as a simulationist story generator [1] focused on simulating small, 19th-century American towns over a 140-year period. Simulationist means that content is generated without user intervention. It relies solely on the bottom-up interactions of individual characters and systems to produce interesting content. *TotT* is not a game experience, but a content generator inspired by by the world generation system of *Dwarf Fortress* [18]. We were drawn to *TotT* because of its emergent potential and its swift town generation (simulating decades in minutes). Also, its code was available online[1], enabling us to get hands-on experience without attempting to reimplement it from published literature.

Towns can potentially hold hundreds of characters. So, to optimize computational performance, *TotT* focuses on only simulating the major events in characters' lives and uses a variable level-of-detail approach to model the passage of time. Days were split into day/night cycles, and only a certain percentage of days were modeled each year. The changes to the simulation between update calls are then interpolated to make sense of the time skip. Using this technique, *TotT* can efficiently simulate the lives of hundreds of characters, including their relationships, occupations, personalities, businesses, and residences. For a more in-depth discussion on *TotT's* internal systems, please see [1].

Previous projects have used *TotT* as a base to build on. *Hennepin*, was the spiritual successor to *TotT* [1]. It built on the foundation of character modeling done in *TotT* and revised parts such as personality models and event causality to provide a more explainable environment for emergent narratives. For instance, the revised personality model, borrowed from *Dwarf Fortress*, supports users in more easily inferring character behavior from personality traits. Other projects include *Stories of the Town*, which integrated *TotT* into a more extensive adversarial character-driven story generation pipeline [16], and *Argument Box*, and experimental game exploring character moral values [15].

B. Rational Reconstructions

Rational reconstruction, as applied to AI systems, is a methodology for understanding the essential design decisions of a system by attempting a clean re-implementation, whose refactoring supports experimentation [19]. The goal is to understand what makes a particular system work, its scope, and its limits, separated from details of the initial implementation that may or may not be core to its success.

Rational reconstructions allow us to better understand a system's contribution and potential by observing what happens when altering core components. Previous projects have also used rational reconstructions to investigate story generation and simulation systems [20]–[22]. *Minstrel Remixed* [21] and *Skald* [20] are rational reconstructions of Scott Turner's original *Minstrel system* [23]. Similarly, the *Ensemble Engine* [22] could be interpreted as a rational reconstruction of *Comme il Faut (CiF)* [24], the social physics engine behind *Prom Week* [6]. *Ensemble* took *CiF*'s social model, removed the experience-specific constructs used in *Prom Week*, and gave users a general schema for creating social simulations with a small cast of characters.

III. What are Talk of the Town-Likes?

We define *Talk of the Town*-likes (*TotT*-likes) as a design space of agent-based social simulations in games character-

[1] https://github.com/james-owen-ryan/talktown

ized by procedurally generated character settlements, socially-grounded characters, abstract social interactions, and a focus on characters' major life events. Here we explain each of these characteristics in more detail and discuss how they relate to other types of social simulation tools and games.

We compare *TotT* to the following social simulation artifacts:

- ***Versu*** [25] – The framework used to create interactive fictions involving small casts of characters and many nuanced social actions. It is best known for the game *Blood and Laurels* [26].
- ***PsychSim*** [27] – Framework for authoring characters that make decisions using *theory of mind* mental models of other characters.
- ***Kismet*** [28] – A small social simulation authoring language that is a spiritual successor to *Talk of the Town*.
- ***Ensemble*** [22] – Tool for creating social simulations with a cast of characters who make decisions based on authored volition rules.
- ***Crusader Kings II*** (CKII) [29] – Commercial game about maintaining your bloodline, by conquering land, navigating relationships with other kingdoms, and passing your power to your heir.
- ***The Sims™4*** [5] – Canonical social simulation game where players influence and observe the lives of their characters.

A. Procedurally Generated Settlements

TotT-like social simulations focus on community-scale simulation (10s-100s of agents) with characters living in various residential buildings. The original version simulated the history of a small American town from the mid-1800s through to the late 1900s. *TotT*-likes are geared towards modeling hundreds of characters, their lives, and relationships. *TotT*-likes do not have a specific narrative aesthetic. They can range from historical fiction to high fantasy to science fiction.

The locations and population of characters are not fixed. They evolve during simulation with the ebb and flow of residents. In the original system, characters can move in/out of the town as businesses open and close, and characters grow old and die. This aspect is different from a number of the other social simulation systems. *Ensemble* and *Versu*, for example, use a fixed cast of characters. *PsychSim* allows users to add characters at runtime, but the characters do not occupy any location.

In addition to the residents, towns are made up of multiple locations that characters can travel to. Locations can be businesses, homes, and public spaces like parks. TotT-likes allow characters to move between the multiple locations in the town: for example, going to the store, going to work, and going home. This movement is the core of how they form their social networks, as characters only interact with those who are at the same location as them. Movement is not smooth, but rather characters teleport from place to place. There is no concept of space or occlusion within a location. All characters are assumed to be visible to all others present at their location.

Other games like *Crusader Kings II* [29], model the lives and legacies of characters over a long expanse of time, but these characters are static in their positioning. They cannot move from country to country as individuals. The Sims™4 offers similar capabilities to simulate towns and lives and deaths of residents. However, its movement is continuous in 3D space. Most academic social simulations focus mainly on modeling a specific social phenomenon and thus do not need to model multiple locations. *Prom Week* and *PsychSim* do not model locations at all, as they focus more on action selection.

B. Socially Grounded Characters

The biggest separator between TotT-likes and other academic social simulations is that it supports characters owning homes, running businesses, and working jobs. These add an extra layer of narrative material to the mix. Businesses are a dynamic part of *TotT's* landscape and afford the formation of business rivalries, workplace friendships, and power imbalances. Owning homes assumes that characters have neighbors.

TotT-likes include more than just social interaction modeling. They include the entire town where characters live. This social landscape embeds characters in an environment of social and physical relationships that emerge dynamically from the simulation. Academic social simulations tend not to have these features. Instead, they focus on modeling a specific social phenomena and less on creating a story world. However, commercial games with social simulations often support such a social landscape, though it is sometimes embedded within a pre-authored story world.

C. Abstract Social Interactions

TotT-likes model interactions between characters using simple models of abstract social interactions. For example, *TotT* only has one explicit "socialize" action available to characters. The relationship model determines the side effects of this socialize action. This is in contrast to having a large number of differentiated actions such as flirt, greet, insult, and gossip. For example, *The Sims™4* provides players with many types of social actions for characters to perform. Versu presents players with many nuanced options based on the currently active social practices [25]. *Prom Week* also gives players plenty of actions for "woo"-ing their crush and climbing the social ladder [6].

In TotT-likes, the interaction resolution is kept low because the simulation mainly focuses on modeling the major life events of characters. Because of this, as well as the larger simulation-scale (number of characters) at which the original *TotT* operated, time steps are stochastically sampled. Not all time steps are simulated with complete character updates. Therefore, more complex and concrete actions that may require multiple time steps for character actions and reactions are not modeled. Instead, actions are atomic operations with immediate side effects.

D. Major Life Events

Related to the above point, TotT-like towns can have hundreds of autonomous characters running around. It is not

feasible to simulate their lives at the same level of detail as on-screen characters in *The Sims™4*. As a solution, *TotT* models the characters' macro or major life events. These events include births, deaths, marriages, divorces, job promotions, and moving into a new home. This is where the story generation magic of TotT-likes resides. TotT-likes simulate the major skeleton of a character's life, leaving it up to users to fill in the gaps with their imagination.

Apophenia is the human tendency to see patterns in random or unrelated things, and it is the glue that makes this fidelity of simulation work for generating narratives. TotT-likes craft low-fidelity stories about characters without detracting from the coherence of their stories. The rest is literally left as an exercise for the reader. Ryan supplied many narrativized traces of stories curated from the original *TotT* [1]. It is fascinating how with a bit of narrative scaffolding, people can piece together more elaborate stories than the simulation actually models.

From our sample of social simulation artifacts, none of them place a specific focus on simulating characters' major life events. We consider the closest analog to be the *The Sims™4*'s low level-of-detail simulation for off-screen characters [30]. One would think that *CKII* would be the most similar given the large span of in-game time it simulates, but CKII has a surprisingly large number of events that range from major milestones to minute events such as your child being afraid of the dark. *Ensemble*, *PsychSim*, and *Versu* do not simulate characters life events at this scale. Events are more like the moment-to-moment changes to the social state resulting from characters performing actions.

IV. Neighborly

Neighborly is a social simulation sandbox for creating TotT-likes. It is our rational reconstruction of *Talk of the Town*. Its goal is to be a tool for experimenting with town-scale social simulations for research, entertainment, or creative projects. We accomplish this by taking a data-driven approach to simulation that affords easier content authoring. *Neighborly's* most significant selling point is the plugin architecture that encourages users to package their authored content (character types, behaviors, places, and custom AI logic) in a modular and shareable fashion. We take special care to provide a clean API (application programming interface), thorough documentation, and examples.

A. Design Goals

While developing *Neighborly*, we focused on supporting the core characteristics of TotT-likes, while generalizing some of *TotT's* more *Bad News*-specific elements (i.e., hard-coded social norms) and improving authorability. Here are our goals for the new architecture:

1) Support the core characteristics of TotT-likes
2) Provide straightforward methods for extension and customization
3) Minimize built-in assumptions about character behavior ans social norms
4) Support a heterogeneous population of characters (characters can have different functionality)
5) Provide a built-in query language for story sifting/story recognition. We talk about this in the Future Work section
6) Structure the project so that its ready to use as a downloadable Python package in third-party projects

We took special care to ensure that Neighborly's project structure, API, and documentation were conducive to integrating into future projects. That is a core part of addressing the lack of code reuse and reproducibility of results. To the best of their ability, software-based research artifacts should adhere to software engineering best practices. These practices exist to create software that is easy to maintain, extend, and use. If we are making software tools for other people to use, we must make it easy to get started using our tools. We offer code samples that demonstrate how to integrate *Neighborly* into different projects and how to use various features.

B. The Engine

Neighborly uses a custom component-based architecture to represent entities in the story world. Our architecture is a combination of Unity's GameObject Scripting API [2] and a traditional entity-component system (ECS) [3]. We refer to entities in *Neighborly* as GameObjects. *Neighborly's* ECS manages all the game objects in the world, a priority queue of Systems (update functions) to run each timestep, and a collection of global resources that all game objects can access (date/time, weather, and relationship networks). We chose to use an ECS because it stresses modularity and would allow users to compose new types of characters and locations.

Each *Neighborly* simulation instance has an engine that is responsible for constructing new entities at runtime. It uses archetype definitions to determine what components should be attached to a particular entity at creation. Archetypes are baseline configurations for different types of entities that can spawn. Users specify entity archetypes in data files that get loaded into the engine by plugins before the simulation starts (See Listing 2).

Neighborly's engine relies heavily on the factory software design pattern. It maintains an internal record of component factory class instances responsible for making specific component types. This workflow was inspired by Kevin Dill's approach to data pipelines for game AI architectures [31]. *Neighborly* focuses on two core types of entity archetypes: characters and locations, each with a different set of components (see Fig. 1). New entity archetypes are specified in YAML. Assuming one uses existing components, no Python coding is required to define new characters and locations.

C. Modeling Characters

Our goal is to provide a platform that supports a heterogeneous population of autonomous characters. This means

[2]https://docs.unity3d.com/ScriptReference/GameObject.html

[3]Entity-component system is a common software architecture pattern for games that stresses modularity by separating data from processes.

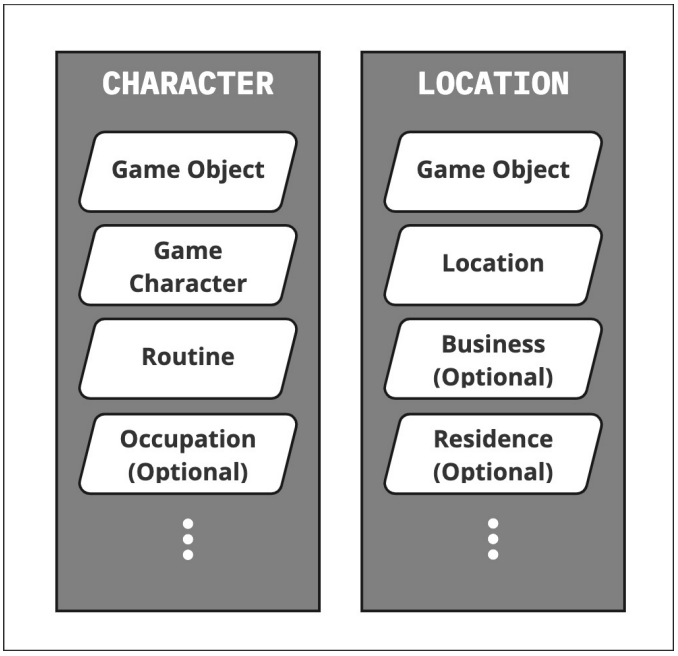

Fig. 1. Entity archetypes are collections of components that are associated with GameObject instance. Generally, entities are either characters or locations. Global resources are singleton components available to all entities in the ECS.

that characters with varying collections of components should still interact within the same space. All characters need a "GameCharacter" component that holds their name, age, current location, life values, and active statuses. However, additional components that control behavior are optional, such as personalities and routines.

We encourage users to extend the base character entity archetypes with additional components. For example, if their town is supposed to be a mix of goblins, trolls, and humans, they should create new components to support the specific behaviors of these factions. If goblins eat humans, one may want a Predator component that keeps track of what kind of entities are eaten (in this case, humans), how many have been eaten, the ID of the most recently eaten entity, and so forth. Then, assuming an action is available for eating, all goblins could participate in eating characters tagged as humans. Further, the Predator component could be reused for other types of characters that eat entities. At the moment, creating new components for the simulation has to be done entirely in Python. Listings 1 and 2 give an an example of creating a custom component, adding it to a plugin, and then using the component in an entity specification.

1) Character Values: Characters can optionally have concepts they value in life (i.e., confidence, material things, knowledge, power). We call these *character values*, and they can influence how characters' relationships evolve and the types of places they enjoy visiting. Character values are a new addition and provide an alternative to *TotT*'s Big 5 personality model.

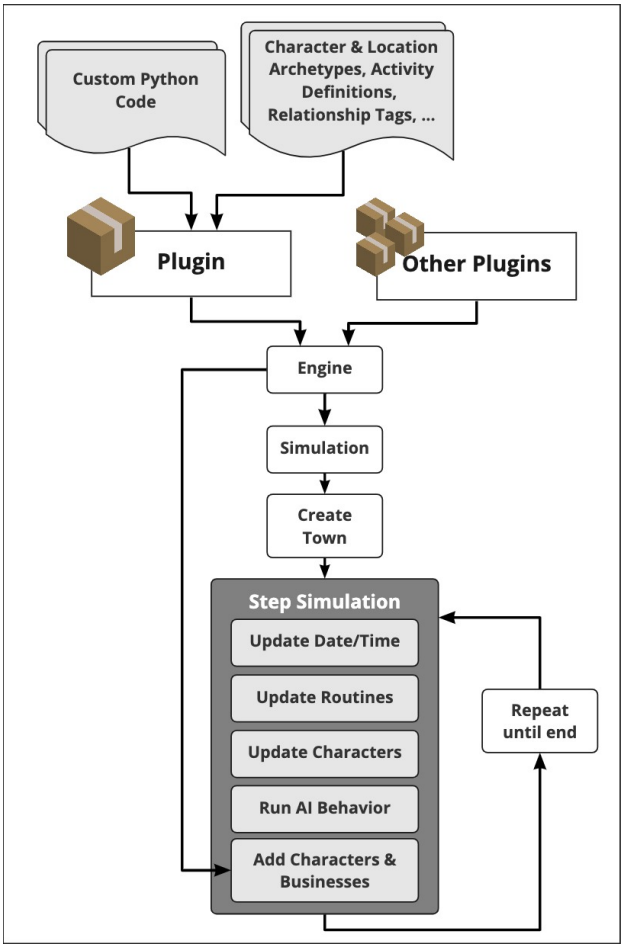

Fig. 2. An overview of *Neighborly's* architecture. The first step is loading entity definitions, component definitions, and other simulation data from plugins. Then the simulation procedurally generates an empty plot of land for the town. Once there is a town, the simulation will simulate each time step until it reaches the specified date to end generation. During each time step, time advances, characters move based on routines, character attributes are updated, character behaviors run, and new characters and businesses are added to the town.

2) Routines: By default, characters in *Neighborly* follow 7-day weekly routines instead of the single-day stochastic routines defined in *TotT*. We made this decision because we wanted to explicitly model weekends. Each day is divided into 24 hours, and scheduled activities may last for multiple hours during the day. At the beginning of the time step, all characters move to their scheduled location. If there is not an activity scheduled, then the characters are free to choose where they want to go based on what locations are available.

3) Roles: Currently, *Neighborly* does not have a general model for representing character's roles within their community. For now we maintain *TotT*'s occupation model. Occupations are the roles that characters gain based on where they work. As such, occupations affect a character's routine, the evolution of their relationships, eligibility for future occupations, and other aspects of their social lives.

4) Behaviors: In *Neighborly*, characters' behaviors are defined by event callback functions that dictate which life events

character's engage in and what happens when they do. We talk more about *Neighborly's* event system in a later section. Giving characters the ability to select which events they accept ensures that characters only engage in events they know how to handle, and allows characters with different decision-making logic to participate in the same interactions.

To help users compose reactive behaviors, *Neighborly* offers a behavior tree API for creating event-callbacks. Behavior trees are a behavior definition data structure made famous by Halo 2 [32] and widely used in commercial games. They compose behaviors using a series of control-flow and action nodes. As with extending characters with new components, users can define new behavior tree nodes to extend the existing ones and compose behaviors using their new nodes and existing nodes. The logic inside of nodes needs to be authored in Python for now. However, if nodes do not depend on one another, designers can reuse them in multiple trees.

D. Modeling Locations

Location game objects give characters places to go in the town. They track what characters are present and maintain a maximum capacity to prevent too many characters from being in one place at a time. Locations can also define a set of metadata such as available activities (gambling, socializing, and reading) or services (errands, leisure, work). Locations can also house businesses or residences. We borrowed these representations directly from *TotT*.

Activities are a new addition and take the role that "occasions" (i.e., errands, leisure, home, work) had in the original *TotT*. Activities are associated with different character values (explained earlier) and so characters will tend to frequent locations with activities that match their values. When characters need to find a location to go during their free time, they will search the ECS for locations that have activities that match their character values.

Businesses help shape the social landscape of the town. They offer social spaces where characters can develop relationships. They also add thematic flavor to the world. The original implementation of *TotT* only featured businesses that were historically relevant to the mid-1800s to late-1900s, and it had date checks to ensure historically accurate businesses appeared during specific eras. With the new plugin interface, designers can easily create new business types that completely change the setting and theme of their simulated town.

E. Modeling Relationships

Relationship data is a global resource in *Neighborly's* ECS. We support two types of social connection networks: reciprocal and directed. Directed relationships allow characters to be friends with someone who considers them enemies. The abstractions for relationship networks are templated containers that can hold any data type. Out of the box, *Neighborly* defines a single directed relationship network where connections between characters are measured using friendship and romance, adapted from *charge* and *spark* from the original *TotT*. Authors can modify or add new relationship networks as desired.

Friendship and romance are modeled as two scalar affinity values ranging from -50 to 50. Friendship is a character's platonic affinity for another character. A Friendship score of 50 means they could be best friends, and a score of -50 means they could be worst enemies. Romance is a character's romantic affinity for another character. A score of 50 means that a character loves another, while a score of -50 means they are repulsed.

Relationships may also have added modifiers, such as *Friend*, *Enemy*, *Coworker*, and *Love Interest*. Modifiers add searchable tags and optional buffs to the friendship/romance values. Users can define new modifiers and load these into Neighborly. Relationship modifiers can be attached manually or added by event handlers. For example, when the *Become-Friends* event fires, the *Friend* modifier is automatically added to each characters' relationship model.

F. The Event System

While the simulation is running, various events may fire and characters have the ability to determine if they want to engage in that event and what happens when they do. We implemented an event-based callback system to handle character behaviors. Since *Neighborly* supports a heterogeneous population of custom agents, we can't make assumptions about the characters' internal structure and their decision-making logic. To keep the simulation flexible and allow different characters to engage in the same events, character components can register as event listeners and accept, reject, and run bespoke code when particular events fire. Users can even create new event types and rules. Characters will continue to function properly even when confronted with unfamiliar events.

Life events are triggered automatically by event rules. Event rules are production rules with preconditions and post-effects. *Neighborly*'s event rules run on set intervals and search the simulation state for characters that match a set of conditions. When they find matches, the event system asks each character involved if they wish to engage in the event. If all characters agree, then each character calls specific event callback functions based on their internal configuration. These callback functions should update the character's internal state and relationship stats with associated characters. *Neighborly's* behavior trees are represented as event callbacks and may be used to compose sequential and conditional behavior rather than calling single functions.

G. Extending Neighborly with Plugins

Plugins are the core way users extend *Neighborly's* functionality without editing the core code. Within plugins, users can define new businesses, residences, occupations, characters, behaviors, and ad-hoc components. Their purpose is to give users the ability to customize the simulations narrative setting and the behavior of the characters that inhabit it. Plugins address the content modularity limitations in *TotT*.

Neighborly plugins are Python modules/packages that are imported prior to starting the simulation. Users can define data using a combination of Python code and YAML data files (See

Listing. 2). Plugins may add new content to Neighborly or override existing definitions.

Since plugins are self-contained Python modules they can be shared as packages on GitHub and the Python Package Index (PyPI) [4]. PyPI is used by Python's package manager and is the official source of Python packages. Having such publicly accessible venues for sharing content should help foster a community around *Neighborly*'s plugin ecosystem. We were inspired by the game mod communities around *Cities: Skylines* [33] and *RimWorld* [34].

```
class CustomPersonalityModel(Component):
    # Personality model attributes
    ...

class CustomPersonalityModelFactory(
    AbstractFactory):
    # Create PersonalityModel instance
    ...

def initialize_plugin(engine):
    engine.add_component_factory(
        CustomPersonalityModelFactory())
    ...
```

Listing 1. Creating custom components requires defining a component class and a factory for that class. The factory is added to the simulation's engine instance when the plugin is initialized.

```
Characters:
  - name: BaseCharacter
    default: yes
    components:
      - type: GameCharacter
        options:
          name: '#first# #last#'
      - type: Routine
  - name: CustomPersonalityCharacter
    inherits: BaseCharacter
    components:
      - type: CustomPersonalityModel
        options:
          # key-value pair options
```

Listing 2. YAML definitions for two character archetypes: BaseCharacter and CustomPersonalityCharacter. All entity archetypes use the same structure of defining an archetype name, optional attributes (default/inherits), and a list of components and their options. Here we define the CustomPersonalityCharacter archetype with inherits properties from BaseCharacter and adds an additional CustomPersonalityModel component.

V. Conclusion

We identify the design space of *TotT*-like social simulation for generating emergent narratives. These systems operate at community-scale, simulating the evolution of communities of characters (tens to hundreds) over decades of time. These simulations are non-interactive and aim to generate content for other media and projects. Due to the population of characters and a desire to keep generation times low, *TotT*-likes use simulation level-of-detail techniques, focusing on abstract social interactions and significant life events over fine-grained character actions.

Neighborly is a rational reconstruction of *Talk of the Town*, designed to be a reusable, extensible sandbox environment for creating TotT-like social simulations. Like in *TotT*, *Neighborly* models characters moving between various locations, holding jobs, opening businesses, and raising families. *Neighborly's* plugin architecture allows users to inject custom code and define characters, behaviors, events, and locations. This empowers users to expand *Neighborly*'s built-in functionality and share their plugins using established Python workflows. Our goal is to create a welcoming environment for non-technical users while still offering experienced programmers extensibility for more advanced add-ons.

A. Future Work

Neighborly is still a work-in-progress and is missing some features. One of the most important is built-in support for story sifting/recognition [35], [36]. Story sifting is the process of extracting intriguing emergent narratives from storyworlds. Simply running a social simulation and reporting everything that happens often results in "boring" narratives: too many facts just told one after another. The original *Talk of the Town* was designed to be used with a human-in-the-loop operator in the *Bad News* live experience. The operator was then tasked with manually inspecting the simulation data and piecing together potential narratives. We plan to integrate our story sifting visual scripting tool, *Centrifuge* [17]. Users could toggle sifting patterns and add new ones. Then with these patterns, users could inspect the story world for narrative fragments. For example, users could look for event sequences where a family rises to prominence in the town, only to have an heir that departs, leaving the legacy in ruin. Story sifting could also be a method for designers to perform expressive range analysis [37] on their towns by specifying a set of patterns and visualizing the distribution of story appearances.

Next, we plan to evaluate *Neighborly* by comparing its generated stories to *TotT*'s. We want to see if the same general set of character scenarios materialize, such as generational legacies, infidelity, asymmetric friendship, love triangles, and rivalries. Character scenarios may not materialize in quite the same distribution, but it is important to measure just how much. We are still determining a proper metric for evaluating the accuracy of *Neighborly* as a reconstruction of *TotT*.

Finally, we plan to perform user studies to evaluate *Neighborly*'s usability and expressiveness. First, we will put *Neighborly* in the hands of professional game designers and ask them to reinterpret story settings from books, shows, or other games into *Neighborly*'s data format. We are inspired by [38], and how they tested their *WizActor* tool by having a professional game designer create a regency-era world using *Kismet*. Second, we will evaluate the usability for *Neighborly* users less versed in simulation authoring.

[4]https://pypi.org/

References

[1] J. Ryan, *Curating simulated storyworlds*. University of California, Santa Cruz, 2018.

[2] R. Aylett, "Narrative in virtual environments-towards emergent narrative," in *Proceedings of the AAAI fall symposium on narrative intelligence*. AAAI Press Menlo Park, 1999, pp. 83–86.

[3] Monolith Productions, "Middle Earth: Shadow of Mordor," [PlayStation 4, Xbox One, PlayStation 3, Xbox 360, Microsoft Windows, macOS, Linux, Classic Mac OS], 2014.

[4] ——, "Middle Earth: Shadow of War," [PlayStation 4, Xbox One, PlayStation 3, Xbox 360, Microsoft Windows, macOS, Linux, Classic Mac OS], 2017.

[5] Maxis, "The Sims 4," [PlayStation 4, Xbox One, macOS, Microsoft Windows, Macintosh operating systems, Classic Mac OS], 2014.

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

[7] K. Stuart, "Keith Stuart on AI, acting and the weird future of open-world games," *Eurogamer*, 2015. [Online]. Available: https://www.eurogamer.net/articles/2015-11-21-keith-stuart-on-ai-acting-and-the-weird-future-of-open-world-games

[8] ——, "Video games where people matter? the strange future of emotional ai," *The Guardian*, 2016. [Online]. Available: https://www.theguardian.com/technology/2016/oct/12/video-game-characters-emotional-ai-developers

[9] A. Wawro, "How devs are working to design game AI that plays like a human," *GameDeveloper*, 2015. [Online]. Available: https://www.gamedeveloper.com/programming/how-devs-are-working-to-design-game-ai-that-plays-like-a-human

[10] D. Heaven, "Cleverness isn't always everything for a gaming artificial intelligence," *NewScientist*, 2016. [Online]. Available: https://www.newscientist.com/article/2078134-cleverness-isnt-everything-for-a-gaming-artificial-intelligence/

[11] J. Ryan and M. Mateas, "Simulating character knowledge phenomena in Talk of the Town," in *Game AI Pro 3*. AK Peters/CRC Press, 2017, pp. 433–448.

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

[13] S. F. Festival. Slamdance dig spotlight - bad news. Youtube. [Online]. Available: https://youtu.be/jrrpLFS9Zcw

[14] S. Wright, "How the mixed reality game "bad news" brings towns like "twin peaks" to life," *Rolling Stone*, 2017. [Online]. Available: https://web.archive.org/web/20171124073022/http://www.rollingstone.com:80/glixel/news/mixed-reality-game-bad-news-brings-small-town-usa-to-life-w479065

[15] R. AlJammaz, Y. She, and M. Mateas, "Argument box," *AIIDE Experimental AI in Games Workshop*, 2020.

[16] C. Miller, M. Dighe, C. Martens, and A. Jhala, "Crafting interactive narrative games with adversarial planning agents from simulations," in *International Conference on Interactive Digital Storytelling*. Springer, 2020, pp. 44–57.

[17] S. Johnson-Bey and M. Mateas, "Centrifuge: A visual tool for authoring sifting patterns for character-based simulationist story worlds," *The 17th AAAI conference on Artificial Intelligence and Interactive Digital Entertainment: Programming Languages and Interactive Entertainment Workshop*, 2021 'in press'.

[18] T. Adams and Z. Adams, "Dwarf fortress," [Linux, macOS, Microsoft Windows, Macintosh operating systems, Classic Mac OS], 2009.

[19] A. Bundy, "What is the well-dressed ai educator wearing now?" *AI Magazine*, vol. 3, no. 1, pp. 13–13, 1982.

[20] B. Tearse, P. Mawhorter, M. Mateas, and N. Wardrip-Fruin, "Skald: minstrel reconstructed," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 156–165, 2013.

[21] B. Tearse, M. Mateas, and N. Wardrip-Fruin, "Minstrel remixed: a rational reconstruction," in *Proceedings of the Intelligent Narrative Technologies III Workshop*, 2010, pp. 1–7.

[22] B. Samuel, A. A. Reed, P. Maddaloni, M. Mateas, and 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.

[23] S. R. Turner, "Minstrel: a computer model of creativity and storytelling," Ph.D. dissertation, University of California, Los Angeles, 1993.

[24] J. McCoy, M. Treanor, B. Samuel, B. Tearse, M. Mateas, and N. Wardrip-Fruin, "Authoring game-based interactive narrative using social games and comme il faut," in *Proceedings of the 4th International Conference & Festival of the Electronic Literature Organization: Archive & Innovate*, vol. 50. Citeseer, 2010.

[25] R. Evans and E. Short, "Versu—a simulationist storytelling system," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 113–130, 2013.

[26] Emily Short, "Blood and laurels," [iPad OS], 2014.

[27] D. V. Pynadath and S. C. Marsella, "Psychsim: Modeling theory of mind with decision-theoretic agents," in *IJCAI*, vol. 5, 2005, pp. 1181–1186.

[28] A. Summerville and B. Samuel, "Kismet: a small social simulation language," in *Casual Creator Workshop at the 2020 International Conference on Computational Creativity*, 2020.

[29] Paradox Interactive, "Crusader kings ii," [macOS, Microsoft Windows, Linux, Classic Mac OS], 2012.

[30] M. Brown, "Emergent storytelling in the sims," 2018. [Online]. Available: https://www.gdcvault.com/play/1025112/Emergent-Storytelling-Techniques-in-The

[31] K. Dill, "Six factory system tricks for extensibility and library reuse," in *Game AI Pro 3: Collected Wisdom of Game AI Professionals*. CRC Press, 2017, pp. 49–62.

[32] Bungie, "Halo 2," [iPad OS], 2004.

[33] Paradox Interactive, "Cities: Skylines," [PlayStation 4, Xbox One, macOS, Nintendo Switch, Microsoft Windows, Linux, Macintosh operating systems], 2015.

[34] Ludeon Studios, "Rimworld," [macOS, Linux, Microsoft Windows, Macintosh operating systems], 2013.

[35] J. O. Ryan, M. Mateas, and N. Wardrip-Fruin, "Open design challenges for interactive emergent narrative," in *International Conference on Interactive Digital Storytelling*. Springer, 2015, pp. 14–26.

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

[37] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 2010, pp. 1–7.

[38] B. Samuel, A. Summerville, J. Ryan, and L. England, "A quantified analysis of Bad News for story sifting interfaces," in *International Conference on Interactive Digital Storytelling*. Springer, 2021, pp. 142–156.