Building Visual Novels with Social Simulation and Storylets

Shi Johnson-Bey(✉), Kira Liao, Samuel Shields, Daeun Hwang, Noah Wardrip-Fruin, Michael Mateas, and Edward Melcer

University of California Santa Cruz, Santa Cruz, CA 95064, USA
{ismajohn,kiyliao,samshiel,dhwang,nwardrip,mmateas,emelcer}@ucsc.edu

Abstract. Simulationist interactive narrative systems allow game makers to craft reactive stories driven by simulated characters and their social dynamics. These systems produce narrative experiences that feel more emergent but may lack a coherent plot structure. We explored how to combine the emergent possibilities of social simulation with a procedural narrative system that affords writers strong authorial control over the plot. We did this by developing a Unity extension called Anansi that helps people create social simulation-driven visual novels. It enables users to inject simulation data into their story dialogue using logical queries and parameterized storylets written using *Ink*. The paper describes an overview of our extension and how we empower writers to drive narrative progression using cascading social effects from player choices.

Keywords: Storylets · Tools · Social Simulation · Unity · Visual Novel

1 Introduction

Visual novels (VNs) have shown to be practical tools for creating educational interactive storytelling games [1]. These games typically feature hand-authored choice-based branching narrative structures that produce well-constructed narrative experiences. However, a shortcoming of the branching narrative design approach is that it is prone to an exponential explosion in authoring burden as game designers try to give the player the illusion of a living world where the player can express agency [2].

Conversely, Simulationist Interactive Narrative systems develop narratives over time from the conflicts and drama that emerge between the player and non-player characters (NPCs) [12]. These systems are much more welcoming to player experimentation and exploration without suffering from an exponential growth in authoring burden. However, this often comes at the cost of game designers having less control over the final plot and overall narrative experience [20].

Since simulation games have also been shown to be effective for educational use [17], it would be ideal if we could provide players with an experience that combines the freedom of a simulation with the well-formed structure of a choice-based narrative.

We explored how to help storytellers create interactive narrative games that leverage simulated non-player characters and procedural storytelling techniques to increase game replayability and encourage player exploration. We built a Unity extension called Anansi[1]. It's a toolkit for building simulation-driven visual novels in Unity, and we are actively using it to create an educational game that teaches players about the Responsible Conduct of Research within academia.

Anansi helps users create location-based, simulation-driven visual novels where the player can move between various locations and interact with the characters present. Player choices can affect their immediate relationships with NPCs and have cascading, second-order effects on other social relationships. We draw inspiration from games like *Persona 5* [19] and the dating sim genre because they focus on cultivating social relationships with NPCs.

We wanted to explore how to strike a balance between the emergent possibilities of simulation and a writing workflow that affords strong authorial control over the plot. Anansi combines a *storylet runtime*, powered by the Ink narrative scripting language[2], with an underlying world simulation with NPCs that follow schedules, move between locations, track their feelings about other characters, and respond to various social events happening within the game.

This paper provides an overview of Anansi's architecture, design motivations, lessons learned, and plans for the future. It also provides the following contributions:

– Demonstrate how to parameterize storylets to allow dynamic casting of characters into dialogue.
– Discuss how writers could associate cascading social effects with story choices.
– An example of game architecture that combines social simulation, storylets, and branching narrative design.

2 Related Work

2.1 Simulationist Interactive Narrative

Research into simulationist interactive narrative has been driven by the goal of producing endlessly replayable and emergent narrative experiences with autonomous characters that respond to the player and guide the plot [12]. Research in this area has been ongoing for decades. However, there are three systems that we highlight as being particularly relevant to this work: *Facade* [14], *Comme il Faut/Prom Week* [16], and *Versu* [4]. We chose these because they are either complete game experiences or have been used to produce complete game experiences.

Prom Week and Versu used parameterized social scenarios to drive character actions and narrative progression. Based on the current game state, characters

[1] https://github.com/ShiJbey/Anansi.
[2] https://www.inklestudios.com/ink/.

would choose actions based on their goals and what scenarios are available. This architecture afforded NPCs strong autonomy over their actions at the cost of authorial control over the game's plot [20]. Thus, the final narrative experienced by the player is their subjective sequence of scenarios.

Facade provides a middle ground between character autonomy and authorial control by using joint actions (like parameterized scenarios) to coordinate NPC behaviors and a drama manager to moderate the presentation of story beats. However, programming NPC behaviors in Facade is complex and is more akin to multi-robot coordination than writing story prose [15]. For this project, we wanted to empower designers to leverage parameterized scenarios while providing an interface that is more friendly to writers.

2.2 Tools for Creating Choice-Based Interactive Narratives

RenPy[3], *Twine*[4], *Yarn Spinner*[5], and *Ink*[6] are all examples of popular tools and engines for creating choice-based interactive narrative games. RenPy is the most popular platform for creating visual novels. It is based on a variant of Python and comes with all the necessary features to create complete experiences. Twine is for making hypertext games that can easily be shared over the web. YarnSpinner and Ink are narrative scripting languages intended to be embedded within a larger game engine. They feature scripting languages that look like screenplay scripts with code-like markup. The benefit of all these tools is that they handle much of the lower-level systems tasks, freeing game designers to focus on authoring dialogue and higher-level systems.

2.3 Storylets

Storylets are a procedural narrative design approach that divides a story's content into individual chunks that can be sequenced in differing orders based on the player's choices and the current game state [10,21]. Typically, storylets are gated by preconditions that must be true for them to be eligible for presentation to the player. Then, through a manual or automatic selection process, storylets are presented, progressing the narrative. Storylets allow narrative designers to forgo a pre-authored linear or branching structure, affording players the agency to chart their own paths through a narrative. We are particularly interested in exploring them for the intersection between interactive narrative and simulation since they have shown useful for procedural story generation [11,13].

2.4 Storylet-Based Interactive Narrative Systems

Two experimental storylet systems that directly inspired this project are StoryAssembler [5], and Lume [13]. Each takes a different approach to creating

[3] https://www.renpy.org/.
[4] https://twinery.org/.
[5] https://www.yarnspinner.dev/.
[6] https://www.inklestudios.com/ink/.

dynamic narrative experiences at runtime. Nevertheless, they were both used to successfully produce interactive games.

StoryAssembler uses a library of pre-authored content fragments to compose dialogue text and choice sets at runtime. The authors showed how one can combine storylets with a planning algorithm to find sequence content that maximizes a set of story goals.

Lume uses logic programming to manage a complex knowledge base containing information about characters, their shared histories, and the state of their world. It is the most similar to this project and natively supports features that Anansi does not. For instance, it can procedurally recall past events in the dialogue and generate text for those events using grammar-based techniques. Like Anansi, Lume uses logical queries to cast characters into parameterized storylets, assuming they satisfy the conditions. The core difference between Lume and Anansi is the greater focus on simulating the cascading effects of player choices on the relationships between NPCs.

3 Motivations and Design Goals

We built Anansi to support an experimental interactive narrative game that teaches players the Responsible Conduct of Research. In the game, players must navigate scenarios that place ethical decision-making in conflict with social pressures, such as upholding friendships, negotiating power imbalances, and maintaining a positive reputation.

The game's mechanics and our team composition influenced the design requirements for the final narrative system. We had three design goals for Anansi: a streamlined authoring interface, dynamic narrative content sequencing, and moderately complex social relationship modeling.

Our first goal was to provide a streamlined content authoring workflow that was easy to learn and supported more complex functionality. Also, we needed it to integrate well with Unity (our game engine of choice). Having too many tools and workflows in research projects has been shown to cause problems for multidisciplinary research teams [22].

Second, we wanted to be able to write conversational dialogue that we could dynamically sequence based on player competency at navigating particular ethical dilemmas, the current relationships between the characters, and other story state information. Additionally, we wanted to encourage players to replay the game and experiment with different strategies, as replay and failure are part of the learning process [18]. We chose storylets because a branching structure would not scale well to the vast possibility space of states that the social simulation could achieve. Plus [13] showed that dynamically casting characters into storylets based on preconditions could produce engaging and unique stories.

Our final design goal was to model moderately complex social relationship mechanics. The social simulation should help guide conversation choices and the progression of the overall plot. The player should have the power to influence their playthrough's short and long-term progression by utilizing their understanding of social dynamics and the immediate and cascading social consequences of their choices. Simulation games have proven helpful for educational games because they provide safe spaces for players to experiment [17]. Also, visual novels are one of the most popular narrative game formats for educational games. By combining the two, we aim to provide the experimental/systems qualities of a simulation game with a dialogue-heavy narrative presentation [1].

4 Ink

Our Unity extension is built on top of the Ink narrative scripting language created by Inkle Studios. Ink combines pure text with programming markup to empower writers to create highly branching, choice-based interactive narrative games. Inkle Studios has used Ink to create several commercial games, including *80 Days* [7], *Overboard* [9], and *Heaven's Vault* [8].

Ink's syntax is easy to learn (See Listing 1.1 for an example). At its simplest, each line of text in an Ink script corresponds to a line of dialogue in the game, and writers can display choices to the players by adding an asterisk to the beginning of a series of dialogue lines. Story content in Ink is usually grouped into sections called *knots*. Based on player choices, writers can divert the story to and from knots, giving rise to the branching narrative flow. In this project, we also heavily use tags, which are non-dialogue metadata that writers can associate with the global story, a knot, a choice, or a single dialogue line. Later, we discuss how we leverage Ink's knots and knot-level tags to support parameterized storylets that dynamically cast characters into roles.

We chose to use Ink for four reasons. First, it has excellent integration with Unity and C#. Second, we were already familiar with Ink's syntax. Third, as stated on the Ink website,"[Ink is] a narrative engine designed to slot into a game engine.". This distinction is very important because we wanted the social simulation to live independently of the writing, and we needed to pre-process the narrative scripts to extract storylet content. Lastly, we chose to use Ink because it saved us time from inventing an entirely new language. We could build on a well-established platform with an active developer community.

5 System Overview

This section provides an overview of Anansi's system architecture. We mainly focus on the storylet runtime component. A full explanation of the social simulation is outside the scope of this paper. However, we briefly explain the various parts with a longer explanation of how it enables writers to create cascading social effects for dialogue choices.

Anansi helps game designers create location-based visual novels where players move between locations and talk to characters. Screenshots of a sample game built using Anansi are provided in Fig. 1. The entire toolkit has four layers: a

```
It's dangerous to go alone. Take this!

 * Take sword. # PlaySound: achievement.wav
    -> adventure_start
 * Leave the sword.
    -> leave_cave_get_eaten

=== adventure_start ===
// This knot continues the story ...

=== leave_cave_get_eaten ===
You leave the cave and get eaten by a monster.
-> END
```

Listing 1.1. A sample Ink story with a single starting dialogue line, two choices, and two knots. The choices divert to separate knots. The first choice has an Ink tag (starting with '#') that tells the engine to play a sound when the choice is selected. Lines that start with "//" are comments and not displayed as dialogue.

presentation layer for UI, a dialogue layer, a world simulation layer, and a game management layer (See Fig. 2). Each layer communicates with the others using a collection of event listeners and callback functions. The only exceptions are the storylet runtime and the world simulation, which communicate using a Logic database (explained later).

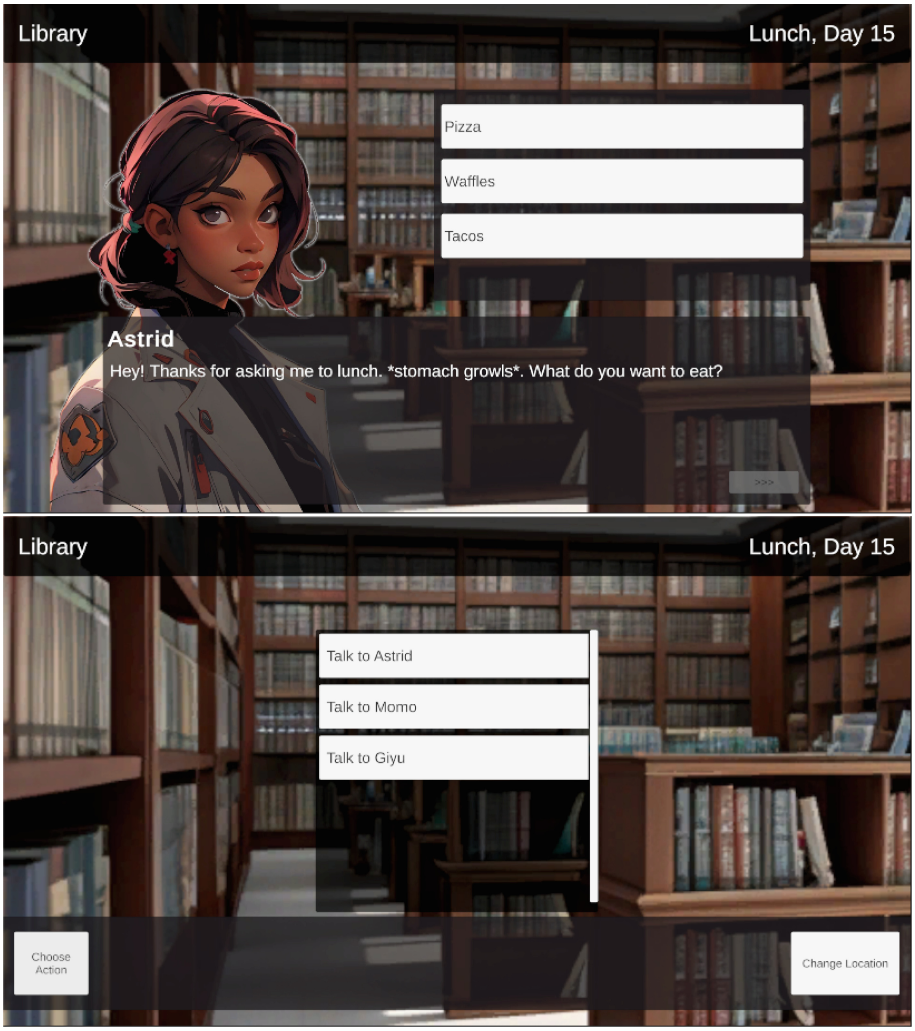
The dialogue layer manages the flow of dialogue that is presented to the player via the UI Layer. It is also responsible for post-processing information from the storylet runtime to manage, among other things, which character to present on screen, the name of the speaker, and the current background to show. The storylet runtime is responsible for managing the flow of the story. It wraps an Ink story instance and provides additional infrastructure for instantiating, searching for, and sequencing storylets.

The database is the connective tissue between the world simulation and the storylet runtime. It originally existed as part of the relationship system but was later integrated into the storylet runtime to facilitate leveraging social simulation data in the story. The simulation syncs various bits of data with the database to make that information available to storylet queries.

The presentation layer is responsible for displaying the current speaker, the speaker's sprite, and any background images. This part of the extension is flexible and can be swapped out by end-users for something that better fits their game's style.

5.1 The World Simulation

The world simulation manages all the aspects of the social simulation (characters, the locations in the world, relationships between characters, social rules



Astrid
Hey! Thanks for asking me to lunch. *stomach growls*. What do you want to eat?

Fig. 1. Our extension provides users with two basic UI modes, one for dialogue (Top) and another for world navigation (Bottom). The dialogue UI has the standard visual novel features: a text box for dialogue, a dialog box containing available choices, a background image, and an image of the currently speaking character. The world navigation UI has a background image, a status bar displaying the current location and time, and an interaction panel with buttons for choosing actions or moving to other locations.

governing relationships, and social events that change relationships). The simulation operates on discrete time steps/ticks. On each simulation tick, we update the current time, move characters to various locations based on their given schedules, update any timed modifiers, and trigger any eligible social events. Our social

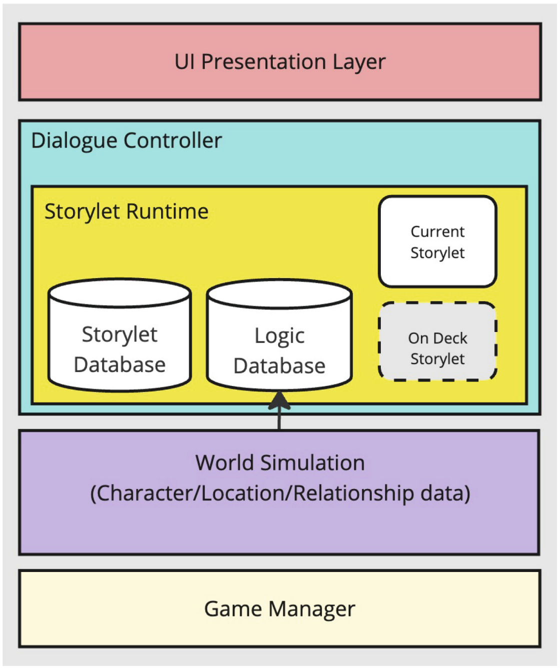


Fig. 2. The extension is divided into four layers. The UI manages what is seen by the player. The dialogue controller manages an instance of the story runtime and feeds dialogue lines and other information to the UI. The world simulation manages characters, locations, and social relationships. It feeds information into the database for the storylet runtime. Last, the Game Manager coordinates resource instantiation and manages all game-specific data not handled by the dialogue layer.

sim is intended to breathe more life into the story world by allowing characters to reason about their relationship with the player and their relationships with other characters. We wanted characters to change their opinion of the player based on how they treat other characters around them. For instance, being nice to a character's family member should help you gain favor with that character. Or, betraying a character should cause you to lose favor with their loved ones but gain favor with their enemies.

Characters. All characters have a unique ID, a display name, a current location, a collection of sprites, and a collection of schedules. Their IDs are used to identify them as the speaker within the story. They are also used to reference the character within the story database. Currently, we use the relationship system (explained later) to track information about character traits and their stats.

Character Schedules. Character schedules tell characters what location they should move to at a given time of day. Users can supply characters with multiple schedules, each containing entries specifying what time of day they are for, the location to move to, and their priority. We use entry priorities since characters can have multiple valid schedule entries for the same time of day [6]. Users may also precondition schedules on the current date, individual NPC variables, or relationship states.

Locations. Locations are anywhere that characters can be. Like characters, each location has a unique ID, a display name, and a collection of sprite images. Locations also track what characters are present.

On the writing side, we associate storylets with locations by giving storylets the same unique ID as the location they correspond to. We also add "location" to the storylet's tag set so the game manager can retrieve it. Every location the player can navigate to must have a corresponding storylet. The storylet does not need to contain dialogue, but it should contain function calls to update the background image and the player's location in the simulation.

Relationship System. The relationship system is responsible for tracking the social relationships between characters, relationship traits, relationship stats, social rules, and social events. It stores relationship information as a directed graph of characters (nodes) and relationships (edges), creating a social network. Each relationship tracks how one character feels about another. Thus allowing characters to have asymmetric feelings toward each other. Relationship information is also synced with the story database.

A full explanation of the relationship system is outside the scope of this paper and will be the subject of another publication. However, the system is publicly available to download within Unity's asset store[7].

Date/Time. The world simulation tracks the current week, weekday (7-day weeks), day, and time of day (morning, afternoon, evening, night). Currently, the time is mainly used to manage character schedules. Locations may also update their backgrounds if they have one tagged for the current time of day. Additionally, the current date/time is also synced with the story database, allowing writers to use it within storylet preconditions and other logic.

Writers can advance time from the Ink script using the `AdvanceTime()` Ink function that steps the simulation until the next time of day is reached. The time of day is incremented every fixed number of simulation steps. This interval can be specified by the game designer in the Unity editor. Whenever the current date/time is updated, the simulation also emits an event that allows third-party systems to react to the change in time.

The Logic Database. The relationship system uses a logic database to facilitate querying for relationship patterns and other information

[7] https://github.com/ShiJbey/TDRS.

https://github.com/ShiJbey/RePraxis. The database is based on the *Praxis* language used in the *Versu* engine [4]. It uses the same exclusion logic syntax and has been streamlined for data insertion, deletion, and queries.

Casting characters into storylets requires us to be able to find characters that meet specific preconditions. So, the relationship systems logic database is shared with the storylet runtime (discussed later) to allow writers to leverage its query system to find character IDs and store them within variables in the Ink script. Database queries are used within the storylet runtime for preconditions, and writers can assert/insert/delete statements directly within Ink. The query syntax used in storylets is passed directly to the logic database when instantiating storylets. The logic database is the core way for the Ink logic to access information about the world simulation when designing storylets.

Cascading Social Effects. Anansi uses *social events* to create cascading social effects in response to player choices. Social Events are a construct of the relationship system. They are instantiated by binding characters to specified roles. Then, based on who is bound and any additional preconditions, the event executes changes to the relationship system and its characters.

The following is an example inspired by a scenario one might encounter within the RCR training game. It showcases an example of how a writer might approach presenting the player with a choice that has ripple effects across their social network.

Suppose the player has been given control of a character named Avery, a new graduate student in a highly competitive lab. They recently discovered that their labmate and friend, Chris, falsified data on a paper they were a co-author on. Chris, a more senior PhD student, is popular in the lab and claims this publication is essential to their dissertation defense. Avery understands that if someone were to find out about the falsified results, it could be detrimental to their research career. However, Avery also worries that notifying their advisor could ostracize them from their other labmates. What will Avery do?

A simplified storylet of the above scenario might look like the Listing 1.2. In it, we define a storylet that would cast an NPC in the role of the labmate, display some dialogue, and give the player a choice of how to handle the situation. The `DISPATCH_EVENT` function is used to call into the social simulation to fire an event in which the player betrays the trust of the labmate (in this case, Chris). Listing 1.3 defines a betrayal event in our social simulation. It uses a query to the logic database to find all friends of the person who was betrayed so that we may adjust how they feel toward the betrayer (in this case, the player/Avery).

5.2 The Storylet Runtime

Anansi's storylet runtime allows us to leverage the social simulation within the Ink scripts by dynamically casting characters using logical queries. The runtime is a C# class wrapper around a standard Ink Story class instance that allows us

```
=== false_data_dilemma(labmate) ===
# ---
# @query
# player.relationship.?labmate.tags.coworkers
# @end
# ===
// Other Dialogue
* Talk to your advisor about your discovery
    {DISPATCH_EVENT("betrayal", "player, {labmate}")}
        {ADVANCE_TIME()}
    -> // Go to other part of the story
* Check in with {CHARACTER_NAME(labmate)}
    -> // Start conversation with labmate
* Let it be and hope no one finds out.
    {ADVANCE_TIME()} // Advance time and simulate if the
        misconduct is discovered
    -> // Go to other part of the story
-> DONE
```

Listing 1.2. "Avery's Research Misconduct Dilemma."

to extract storylet information and high-jack Ink's native control flow. We are able to interleave Ink-native branching and dynamic jumps to storylets.

In Anansi, storylets can be used to represent locations, actions players can take at a location, individual conversations, and scenes. We use them anywhere we want to conditionally gate or parameterize a bit of story content based on the state of the social simulation.

Anansi allows writers to treat character conversations as a "mystery bag" of topics that the game randomly selects from and presents to the player. Additionally, writers can associate selection tags and weight values to storylets to affect their eligibility and likelihood of being selected.

5.3 Defining Storylets

Anansi storylets are Ink knots with specialized metadata. All the metadata about a storylet is stored in its *header*. This is a collection of Ink tags placed below the name of the storylet. The header contains information about a storylet's weight of being selected, choice label (text displayed when offering the storylet as a choice), selection tags, repeatability, and cooldown time (see Listing 1.4). When loading a new Ink script, Anansi inspects all the knots in the script and extracts those containing storylet headers.

Additionally, there are also commands for working with precondition queries. All precondition queries have a `@query ... @end` section where all lines between `@query` and `@end` are collected to form a single query. We also supply users with the `@set X to ?Y` and `@using X as ?Y` commands for binding query variable value to Ink variables and vice-versa.

```
- name: Betrayal
  roles:
    - "?betrayer"
    - "?victim"
  description: "[betrayer] betrayed [victim]."
  responses:
    - effects:
        - DecreaseRelationshipStat ?victim ?betrayer
            Friendship 10
    - preconditions:
        - ?victim.relationships.?victim_friend.traits.friend
        - neq ?victim_friend ?betrayer
      effects:
        - DecreaseRelationshipStat ?victim_friend ?betrayer
            Friendship 5
```

Listing 1.3. "A betrayal social event defined using YAML."

Precondition queries are how we cast characters into roles. Within the query section of the storylet header, designers can precondition storylets on the position of characters in the world, any of their existing relationships, or their personal stats. Using variables (e.g., "?Y") within a query will cause the system to try and find a value in the database that satisfies the query when the variable is substituted. So, we bind character IDs to variables within query results in C# and rebind those variables to Ink-level variables. In Listing 1.2, the storylet header queries for a player relationship with the tag, "coworker". It binds the resulting character ID to "?labmate" and passes the resulting value to the "labmate" parameter variable in the Ink knot.

5.4 Instantiating Storylets

For a storylet to be available, we need to be able to *instantiate* it. This means, if the storylet has a precondition query, when that query is run against the logic database, the query must be successful. If the query contains variables, Anansi creates individual *storylet instances* for each valid set of variable bindings. If the query does not contain variables or there is no query at all, Anansi creates a single storylet instance. The storylet instances of all eligible storylets represent selection space when Anansi needs to choose a new storylet. For example, suppose we have a storylet about having lunch with a friend (see Listing 1.4). It requires the player to be in a location that serves food and for an NPC that considers the player a friend to be present in the same location. If there are two NPCs that meet the criteria, this will result in two potential storylet instances to choose from, one for each valid NPC.

Whenever we run a storylet instance, the runtime binds all the instance's variables to the story state and stores a reference to the current instance. From here, the runtime passes control down to Ink's runtime. When the writer wants

```
=== lunch_with_friend ===
# ---
# tags: convo
# @using speaker as ?speaker
# @query
# ?speaker.relationships.player.traits.friend
# player.location.traits.serves_food
# @end
# ===
{speaker}.happy: Hey! Thanks for asking me to lunch.
    *stomach growls*. What do you want to eat?
* Pizza
    -> // Dialogue about pizza
* Waffles
    -> // Dialogue about waffles
* Tacos
    -> // Dialogue about tacos
-> DONE
```

Listing 1.4. "An Ink knot containing a storylet header with associated metadata."

to jump the story to another storylet, they call helper functions that "queue" a storylet to be executed next. If the runtime can successfully instantiate the storylet, it selects one of the instances, using weighted random selection, and saves this instance as "on deck". So, when the story is advanced, we swap the active storylet with the one on deck, jump to its knot path in the Ink runtime, and continue the story. Since all this is hidden from the user, they see a seamless continuation of dialogue content, just like with branching diverts in Ink.

5.5 Enabling Gameplay

Locations are places where characters can be, and actions are things the player can choose to do when at a location. Actions and locations are represented as storylets containing "action" and "location" tags, respectively. The Game Manager is responsible for determining which actions and locations are available given the player's current location and the state of the rest of the simulation.

Representing actions and locations as storylets allows us to maintain most of the authoring effort within Ink scripts. Also, it means that we can trigger special dialogue or events when a player navigates to a location or takes an action. The storylet representation provides a nice middle-ground between gameplay mechanics and narrative flow.

The background image changes with the current location and time of day. We are working on allowing writers to change the background image on a per-dialogue line basis using Ink tags.

The speaker sprite changes based on the specified speaker ID and sprite tags provided in the Ink dialogue. The dialogue manager post-processes each dialogue line and extracts any ID and optional tags provided prior to a colon ":". To illustrate, the line, "ben.smiling.blue_shirt: Howdy!" will attempt to display a sprite of the character with the ID of "ben" and tags for "smiling" and "blue_shirt". When the line is fed from the dialogue manager to the UI, the speaker information is removed, and only "Howdy!" is displayed.

6 Discussion

In this section, we discuss lessons learned and challenges encountered thus far while developing our educational game using Anansi.

6.1 Designing for Fungible Characters

Characters in Anansi are fungible. This means that, depending on the storylet, we can substitute them with any other character that meets the storylet's preconditions. This is great for incremental development, games using characters as downloadable content, or games with procedurally generated characters because new characters can be leveraged in existing content without any need to explicitly write them into the story. However, the downside of this fungibility is that dialogue lines might not fit the characters being cast or the social dynamics associated with them.

Not all characters speak the same way, and not all characters interact with each other in the same way. Dialogue lines must account for the individual personality of the character and their social context. Is the character speaking with a new acquaintance, an old friend, or a love interest? What is the character's personality? Are they boisterous or reserved? Do they like using fancy words to appear erudite, or do they use more colloquial terms?

The responsibility falls on writers to add conditional checks within the Ink script to ensure the appropriate dialogue lines are presented to the player. Thus far in our game development, we have found that limiting the number of dynamically cast characters and being more specific about trait/relationship requirements in storylet precondition queries helps temper the combinatorial complexity.

6.2 The Tendency Toward Traditional Branching

Storylet and social simulation systems have a learning curve for those unfamiliar with procedural narrative design. They require writers to think about the story as a possibility space for things to happen rather than a series of set plot paths for players to traverse. Tracking the story state is challenging because, at times, writers cannot fully guarantee what the player has seen or done. There could be multiple ways for players to reach the same place in a story.

When teaching historians how to use a social simulation system for interactive storytelling, DeKerlegand et al. noticed that new users tend to avoid dynamism in favor of structuring content to resemble linear and branching structures [3]. A similar phenomenon was encountered by Garbe et al. when writers were tasked with creating story content within StoryAssembler [5]. Writers tended to avoid more dynamic/procedural designs to quickly reach operational results.

Even as we develop our educational game, we occasionally fall into the trap of a strictly branching narrative design. Keeping track of all the information during the design phase is challenging. As we continue development, we aim to identify techniques to help others become more comfortable with the combinatorial complexity of storylets and social simulation for interactive storytelling.

6.3 Deciding Between Ink Variables or the Database

Anansi provides two options for storing variables/game state: Ink variables and the logic database. We use Ink variables whenever possible because they have full support within the Ink language. When we need the social simulation to respond to a value in Ink, we store a copy of the value in the logic database. This requires an additional step, but it allows for additional bi-directional communication between the story content and the social simulation.

7 Future Work

We are currently using Anansi to develop an interactive narrative game that teaches players about ethics and the Responsible Conduct of Research within academia. Players must navigate various scenarios that place ethical decision-making in conflict with navigating social pressures. We plan to report a post-mortem of our development experience and an analysis of the players' feelings and learning outcomes.

Additionally, we want to elicit qualitative feedback from a broader audience by conducting a user experience test as a 48-hour game jam. Participants will be tasked with creating a visual novel game using Anansi. At the end of the jam, participants will submit their games and complete a system usability survey regarding their experience. We will analyze how participants utilize Anansi's features and identify what works well, needs improvement, or is missing.

8 Conclusion

In this paper, we provided a system overview of Anansi, our Unity extension for building visual novels using a combination of social simulation, storylets, and branching choices. Anansi's storylet runtime allows storytellers to write storylets, define roles for who may be involved, and have characters dynamically into those roles based on their relationships and the current story state. This enables storytellers to reuse content and customize it for different character combinations. Additionally, leveraging a social simulation enabled us to provide players with story choices that can have immediate and cascading social effects on their relationships with NPCs. By combining simulation-driven gameplay, storylets, and hand-authored branching narrative content, we aim to support game replayability and player exploration while providing storytellers with adequate authorial control over the plot.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant No. 2202521. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Camingue, J., Melcer, E.F., Carstensdottir, E.: A (visual) novel route to learning: a taxonomy of teaching strategies in visual novels. In: Proceedings of the 15th International Conference on the Foundations of Digital Games. FDG 2020. Association for Computing Machinery (2020). https://doi.org/10.1145/3402942.3403004
2. Crawford, C.: Chris Crawford on Interactive Storytelling. Pearson Education (2004)
3. DeKerlegand, D., Samuel, B., Treanor, M.: Pedagogical challenges in social physics authoring. In: Mitchell, A., Vosmeer, M. (eds.) Interactive Storytelling, pp. 34–47. Springer, Cham (2021)
4. Evans, R., Short, E.: Versu-a simulationist storytelling system. IEEE Trans. Comput. Intell. AI Games **6**(2), 113–130 (2013)
5. Garbe, J., Kreminski, M., Samuel, B., Wardrip-Fruin, N., Mateas, M.: Storyassembler: an engine for generating dynamic choice-driven narratives. In: Proceedings of the 14th International Conference on the Foundations of Digital Games. FDG 2019. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3337722.3337732
6. Graham, R.: Game AI Pro, chap. Breathing Life into Your Background Characters. CRC Press (2013)
7. Inkle: 80 days. [iOS, Android, Microsoft Windows, MacOS, Nintendo Switch] (2014)
8. Inkle: Heaven's vault. [Nintendo Switch, PlayStation 4, Microsoft Windows] (2019)
9. Inkle: Overboard. [Nintendo Switch, Android, Microsoft Windows, iOS, MacOS] (2021)
10. Kreminski, M., Wardrip-Fruin, N.: Sketching a map of the storylets design space. In: Rouse, R., Koenitz, H., Haahr, M. (eds.) Interactive Storytelling, pp. 160–164. Springer, Cham (2018)
11. Lessard, J., Paré-Chouinard, S.: Dramatic situations for emergent narrative system authorship. In: Vosmeer, M., Holloway-Attaway, L. (eds.) Interactive Storytelling, pp. 217–228. Springer, Cham (2022)
12. Louchart, S., Truesdale, J., Suttie, N., Aylett, R.: Emergent narrative: past, present and future of an interactive storytelling approach. In: Interactive Digital Narrative, pp. 185–199. Routledge (2015)
13. Mason, S., Stagg, C., Wardrip-Fruin, N.: Lume: a system for procedural story generation. In: Proceedings of the 14th International Conference on the Foundations of Digital Games. FDG 2019. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3337722.3337759

14. Mateas, M., Stern, A.: Façade: an experiment in building a fully-realized interactive drama. In: Game Developers Conference, vol. 2, pp. 4–8. Citeseer (2003)
15. Mateas, M., Stern, A.: A behavior language: joint action and behavioral idioms. In: Life-Like Characters: Tools, Affective Functions, and Applications, pp. 135–161. Springer, Heidelberg (2004)
16. McCoy, J., Treanor, M., Samuel, B., Reed, A.A., Wardrip-Fruin, N., Mateas, M.: Prom week. In: Proceedings of the International Conference on the Foundations of Digital Games, pp. 235–237 (2012)
17. Minnery, J., Searle, G.: Toying with the city? Using the computer game simcity^{TM}4 in planning education. Plan. Pract. Res. **29**(1), 41–55 (2014)
18. Mitchell, A.: Writing for replay: supporting the authoring of kaleidoscopic interactive narratives, pp. 131–145. Springer, Cham (2022)
19. P-Studio: Persona 5. [Nintendo Switch, PlayStation, Xbox, Windows] (2016)
20. Riedl, M.O., Bulitko, V.: Interactive narrative: an intelligent systems approach. AI Mag. **34**(1), 67 (2012). https://doi.org/10.1609/aimag.v34i1.2449. https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2449
21. Short, E.: Storylets: You want them (2019). https://emshort.blog/2019/11/29/storylets-you-want-them/
22. Szilas, N., Spierling, U.: Authoring issues in interdisciplinary research teams, pp. 287–302. Springer, Cham (2022)