Lume: A System for Procedural Story Generation

Stacey Mason*
University of California, Santa Cruz
Santa Cuz, California, USA

Ceri Stagg*
Independent Researcher
San Rafael, California, USA

Noah Wardrip-Fruin
Michael Mateas
University of California, Santa Cruz
Santa Cruz, California, USA

ABSTRACT

Procedural storytelling offers immense promise for games to offer more reactive narrative experiences that feel more deeply tailored to players' decisions. To date, interactive narrative systems have tended toward either a large emergent possibility space with less focus on narrative structure, or toward greater structure with smaller possibility spaces. In this paper, we introduce Lume, a system for procedural narrative generation that combines the best of these two approaches through a novel combinatorial scene architecture in which storylet scenes are comprised of parameterized node-trees. We detail how the system works and discuss how it moves toward creating reactive interactive narratives that are both dynamic and coherent.

CCS CONCEPTS

• **Human-centered computing → Interactive systems and tools**; **Interaction design theory, concepts and paradigms**; **Systems and tools for interaction design**; *Hypertext / hypermedia.*

KEYWORDS

Lume, procedural narrative, narrative generation, game narrative.

ACM Reference Format:
Stacey Mason, Ceri Stagg, Noah Wardrip-Fruin, and Michael Mateas. 2019. Lume: A System for Procedural Story Generation. In *The Fourteenth International Conference on the Foundations of Digital Games (FDG '19), August 26–30, 2019, San Luis Obispo, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3337722.3337759

1 INTRODUCTION

Driven by rising production costs and changing narrative models within the games industry [27], many developers are seeking procedural approaches to narratives within games. The Game Developers Conference over the last few years has seen both academics [18] and industry professionals alike [14] calling for the adoption of procedural narrative methods. Procedural storytelling offers immense promise, not only for generating large amounts of narrative content or as a cost-cutting measure, but also for games to achieve greater narrative agency, offering more reactive narrative experiences that feel more deeply tailored to players' decisions. As a result, we have seen a rise in new approaches to procedural narrative across research communities in both academia and the games industry [29].

One approach to procedural narrative has proven to be particularly promising is what Max Kreminski and Noah Wardrip-Fruin call *storylet* systems [12]. These are systems that treat narrative as a series of short vignettes ("storylets") that when taken together constitute a larger story. Games such as *Reigns* [23] or *Fallen London* [5] have proven that this approach can be successful, both critically and commercially.

Yet while these systems present great potential for harnessing the dynamism of procedural narrative, they struggle with the coherence of their narrative through-lines. Many of the games created using this model prioritize a quick cadence and a core design loop that focuses on the ludic pleasure of player stat-balancing more than on coherent, connected storytelling.

In an effort to improve this narrative coherence while maintaining the variety and player-driven strengths of emergent narrative, we propose Lume, a system for generating procedural narratives rooted in the storylet model. Although still in development, our system presents a rule-based approach built in Prolog—a declarative programming language based in formal logic that acts as a built-in constraint solver for content selection. Each scene is selected from a pool of scenes deemed appropriate (through constraint satisfaction) to that point in the narrative. Parameterized fields for characters, events, and places are dynamically filled in based on the world state and any preconditions specified by the author. The text of each scene is also heavily parameterized, taking advantage of Prolog's definite clause grammars (DCGs), which facilitate dynamic text generation based on the state of the world. Choices may be bespoke or may be assembled dynamically from conditional rules, and in turn their narrated feedback and outcomes may be bespoke or assembled as well. The system can build scenes that refer back to earlier choices or events using an optional *recall phrase* that binds to events using parameterized characters and objects.

While the algorithms we present here are not, on their surface, a radical departure from many standard applications of Prolog, including some used for static (i.e., non-interactive) story generation [24], we believe that our application of these ideas to the domain of interactive narrative, and the design of our system toward the goal of allowing more reactive character relationships and story moments, introduces a promising new technical design approach for interactive narratives. The contribution of this work is primarily in the narrative schema and authorial model that the Lume system uses to create interactive narratives.

*Both authors contributed equally to this research.

FDG '19, August 26–30, 2019, San Luis Obispo, CA, USA

ACM ISBN 978-1-4503-7217-6/19/08…$15.00
https://doi.org/10.1145/3337722.3337759

2 RELATED WORK

Much work on procedural narrative generation has approached the problem by applying various models of narrative from narratology research to generate completed stories [13]. While this research has yielded interesting results, many of the concerns and innovations in those systems are aimed at trying to generate completed (static) narratives with salient plots. Though plot is a useful and important element of rich storytelling, Lume's goal is not to generate a plot arc autonomously, but rather to surface content strategically in response to player interaction. Thus the work done specifically in the space of interactive storylet approaches and procedural systems, and those that surface salient narrative based on player input, prove to be apt starting points for comparison.

One of the foundational academic descriptions of a procedural storylet system is Mark Bernstein and Diane Greco's Card Shark [3], a system that "begins with a set of lexia[1], all of which are connected to each other, and builds structure by removing unwanted connections" through a process that Bernstein and Greco term *sculptural hypertext*. The promise of sculptural hypertext—which Aaron A. Reed examines and folds into his broader definition of "sculptural fiction" [25]—is one of assembling a narrative through selection from a database. Yet Bernstein acknowledges,

> Where a sculptural strategy has been employed in the past—most notably, perhaps, in Malloy's (1993) *Its name was Penelope* and in Malloy and Marshall's (1996) *Forward Anywhere*—it has been chosen in part to deemphasize temporal sequence and narrative structure (Golovchinsky and Marshall 2000). [16][17][7]

Bernstein and Greco cite earlier attempts at sculptural fiction whose selection process was largely random. Unlike these, however, Card Shark selects eligible lexia based on the state of the world—that is, it selects *relevant* content—and presents all of those relevant options to the reader, who then decides which lexia to visit next. This approach forces readers to strategize about their own traversal through the work, making a decision as to which content they would like to experience next. In contrast, our system presents only one of the (potentially many) viable lexia to players in an effort to keep players invested in the role-playing opportunities the narrative provides (through diegetic agency). It never asks them to break that role by operating as their own narrator and providing extra-diegetic agency [19].

Like the works Bernstein and Greco cite, many popular games today employing a sculptural approach—most notably, the very card-like *Reigns*[23]—provide something of a loose emergent narrative. Yet this narrative is typically an after-thought to the ludic pleasures of stat-balancing. Little attention is paid to character arcs and development, scene length and pacing, rising and falling dramatic tension curves, and the causal relationship between events. The last of these is perhaps one of the most important considerations for interactive storytelling that seeks to create and maintain a sense of players' narrative agency.

Mateas and Stern [20] note that most interactive narratives fall into two general approaches: a more hand-crafted structure of nodes—such as the plot structures common to action/adventure games, hypertext fiction and hypermedia, and choose-your-own-adventure books—and a more procedural simulation approach that encourages emergent narrative, such as those found in sim games, virtual worlds, or the narrative created in the recounting of a sport or eSport match. Their technical approach to *Facade* sought to find a middle ground between these extremes through a drama management system that selects fine-grained narrative beats appropriate to particular narrative contexts. While *Facade* is widely recognized as a successful experiment in very reactive narrative, the cast was very contained, thus beats were not parameterized to be applicable to multiple characters in multiple narrative contexts. In contrast, parameterized beats applicable to a larger cast was one of the major design goals of *Prom Week* [21]. Yet while it was successful in creating dynamic emergent narrative events, these events lacked a clear, recognizable narrative structure. Our system aims to take from the best of both approaches, using parameterization that makes content more applicable and reusable across different points in the narrative, but also provides the narrative structure to offer stronger signaling of causal relationships.

The idea of building a parameterized storylet system is promising, and research into storylet systems is being pursued on multiple fronts. StoryAssembler [6] is an exciting approach that, like Lume, selects parameterized storylets from a pool of content. However, StoryAssembler uses a hybrid-planning approach (in contrast to our bottom-up logic-programming approach) that is organized around explicit authorial goals, using the concept of goal as a global mechanism for authorial control over narrative progression. As Lume remains an ongoing research project, there will be interesting future work to determine how much additional content-selection scaffolding is needed in practice to formulate satisfying narratives. Our early intuition is that these different approaches will likely yield different characteristics from those of the narratives themselves.

3 NARRATIVE MODEL

Our concept of narrative is derived from a popular theory of narrative practitioners, conceiving of narrative as a series of interconnected moments [22]. Whether creators organize these moments into scenes, beats, film shots, or pages, we experience narrative linearly in discrete moments. We expect these moments to feature consistent characters, and to be related causally. These expectations are so strong that consumers of emergent narrative will often read causality into sequential moments, even where none is explicitly shown or modeled [28].

In general, storylet systems promise an alternative to hand-crafted branching plots and emergent simulated plots. The narrative vignettes themselves may be as well-formed as the author likes, while the procedural combination of these vignettes can lead to the emergence of overarching narrative arcs. The creation of a well-formed narrative arises from selecting *appropriate* moments into *appropriate* slots, and the more specific the selected narrative scenario is to the current world-state, the more reactive the narrative should feel to the player's input.

Our hope for the Lume system is to strike a balance between what we call *narrative dynamism*, the feeling that what is happening is one of many paths that could have been taken, and *narrative coherence*, the feeling that narrative events are developing causally

[1]Lexia is the commonly-accepted term within the hypertext community for a node, often a passage of text, that is linked to other passages.

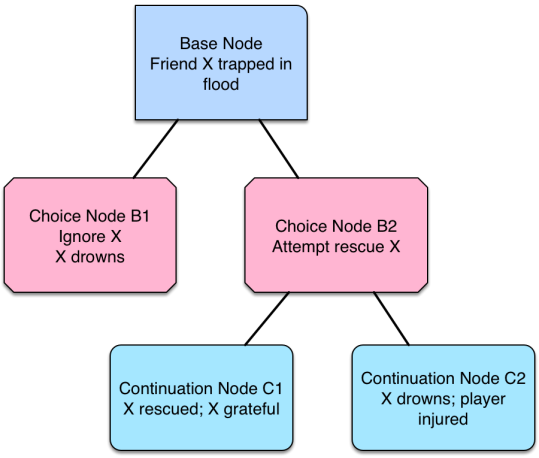

Figure 1: A scene node-tree.

from player actions or from logical NPC reactions. The Lume system offers a hybrid approach in which individual scenes constitute node trees, but the selection of the scene to be displayed next is left up to the system. By building on the capabilities of logic programming approaches to content selection, we are able to construct a coherent narrative through a series of moments appropriate to the given context. In addition, we parameterize several elements of the selected scenes themselves in an effort to allow both content reuse in more places, and (more importantly) a greater sense of narrative reactiveness, as more content is able to be catered to the player's past decisions. Thus the narrative experience is *shaped* by logical rules authored by creators, but the actual narrative experience emerges from the player's decisions.

4 SCENE NODE-TREES

"Scenes" are the units of narrative moment we use as our storylets. The system presents a scene, and based upon the outcome of that scene, it updates the state of the world before determining which scene to present next.

Scenes are composed of a tree of nodes: a mandatory base node, which typically presents the premise of the scene and introduces initial characters, and optional child nodes for player choices or variable beats within the scene (discussed below). These nodes, taken together, form a tree that varies the outcome of any particular narrative moment (see Fig. 1).

Our procedural scene design provides a way to hybridize the branching node structure commonly used for well-formed interactive narratives—such as those found in interactive narratives with authored choice-paths—with the huge possibility spaces of more proceduralized approaches such as those found in simulation games.

4.1 Base Nodes

Each scene has a Base Node as its root. The constraints of the base node must be satisfied in order for the scene to be eligible for selection (and child nodes may impose additional constraints). All base nodes contain the following components:

4.1.1 Preconditions. Preconditions are the rules that must be satisfied in order for a scene to become eligible for display. Preconditions are specified by narrative designers. Examples of preconditions include:

- Player is in The Woods
- It is morning
- A cautious NPC is in your party
- The player just changed locations
- The player has killed an NPC
- A Traumatic Event has occurred

4.1.2 Bindings. Since scenes are authored in a parameterized manner that leaves slots to be filled by narrative features such as characters, places, or flags. These bindings allow each slot to attach to specific eligible objects in the game world. For example, a scene's preconditions may specify that a scene takes place between the player and an NPC in the party. The system would then try to use Sara (and then Mehmet, and Alice, etc.) as a candidate character for the NPC binding slot for this scene. It may succeed with some, all, or none of the candidates, and will return scene trees for all successful binding sets and then select one. If a scene has binding requirements, and the bindings cannot be satisfied by available characters/objects, the scene will be ineligible. Bindings particularly leverage Prolog's constraint-solving capabilities: given the current state of the world and the constraints specified, Lume will automatically try all candidate objects in each slot, trying to build viable nodes and trees that work.

4.1.3 Instructions. Instructions are the output of the scene, either in the form of display text, functional tags, or markup to be interpreted and used by other engines. Instructions may be simple strings of static text, but in practice, the system is most powerful when instructions take the form of DCGs (Prolog's definite clause grammars) which allow for text to be generated dynamically. Bindings may be passed into an instruction's DCGs to modify the text dynamically. An example instruction:

> Instructions = [NPC1, "never loved that ", [insultnoun, foolish]," ", NPC2]

might be presented as "Sara never loved that oaf Ralph" if Sara is bound to NPC1 and Ralph is bound to NPC2, and "oaf" is one of the possible expansions of [insultnoun, foolish].

4.1.4 Postconditions and the Event List. Postconditions change the state of the world in response to the scene presented. In our system, "state" is stored as a single long list of events that have occurred in the world, and a scene's postconditions operate by appending events to this Event List. No state exists outside of this event list. Using this approach, we track not only the *current* state of the world, but also all of the previous states and events. Thus we can search the list for the most likely event to have caused major relationship changes or for any similar turning point. We can then refer back to these moments using Recall Phrases.

4.1.5 Recall Phrases (optional). *Recall phrases* generated by a scene are sections of markup text that refer back to the events that took place in that scene. We might want to refer to a moment in which a major relationship change happened, a character died, a building burned down, or any other significant story event. Recall phrases

allow us to query the system not just for a *particular* major event (e.g., the time you killed John), but for any event that fits a designer's criteria (e.g., the last time X's opinion of you changed for the worse).

In practice, different characters will refer to past events in different contexts, some positive and some negative. To handle these different tones, the system supports long (default), short, and negative recall phrases. For example, if the house burns down, we would attach recall phrases to that scene:

- Long recall: the house burned down
- Short recall: the fire
- Negative recall: you let the house burn down

An author might then write "Sarah mumbles, 'Things haven't been the same between us since [ShortRecall].' " The author can then link Recall to a time in which a major event lowered your relationship with Sarah. If the player has seen the fire scene, the line will replace with "Things haven't been the same between us since the fire." If, however, a different major event lowered your relationship, Sarah might instead say "Things haven't been the same since the affair." If the line calls for a more aggressive accusation, we might choose to use the negative recall instead: "Forgive!? Of course I can't forgive; [you let the house burn down]!"

Recall phrases are entered by a human author, but may be further parameterized by DCGs (see section on DCGs below). They may be attached to a base node ("that time the church flooded") or may be attached to choice nodes ("that time you chose to leave George in the flood") or other child nodes.

4.2 Choice Nodes

In addition to base nodes, scenes may optionally contain choices for the player to make. Each choice and its subsequent feedback to the player is contained within a choice node. Choice nodes have all of the same components as base nodes—preconditions, bindings, instructions, postconditions, and optional recall phrases—with some slight distinctions, detailed below.

4.2.1 Preconditions. Choice nodes may have their own preconditions that must be satisfied in order for this particular choice to be presented to a player within the Scene. Unlike base nodes, if a choice node's preconditions are not met, it does not necessarily invalidate the Scene's eligibility (see section on Scene Eligibility).

4.2.2 Bindings. Bindings are inherited from the base node, but choice nodes may specify additional bindings if needed. (These are inherited in turn by any children of this choice node.)

4.2.3 Choice Text. This is the text displayed to the player before a choice is selected.

4.2.4 Instructions. A choice node's instructions are presented/executed if the player selects this choice. They usually take the form of feedback text offered to the player.

4.2.5 Postconditions. A choice's postconditions are applied after the player makes this particular choice (e.g., Penny's relationship with the player increases by a large amount as you chose to rescue her from the burning garage), whereas a base node's postconditions apply no matter what choice is selected (the garage has been destroyed).

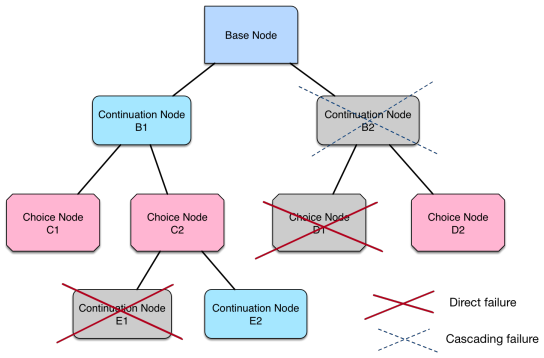

Figure 2: Eligibility in scene node-trees. This is a valid scene, because at least one continuation or at least two choice nodes are available at each level, thus a viable scene can be presented.

4.2.6 Recall phrase. Recall phrases attached to choice nodes refer back to the exact choice a player made in a given situation. Though recall events may be applied to any node, they are most powerful when attached to choice nodes.

4.3 Continuation Nodes

Continuation nodes are used when the scene continues without a player choice. Continuation nodes may have any of the other node components. One useful way to use them is to specify multiple continuation node children, and to attach preconditions to each based on things like characters present in a party, flags raised by other scenes, etc., to branch the scene based on context. Another useful application is to place die-roll preconditions on continuation node children to add variability and unpredictability to outcomes after a choice.

4.4 Scene Eligibility

Scenes are eligible if the preconditions of the base node are met and any specified bindings can be filled. Whenever a scene node has choice nodes as children, at least two of those children must be valid for the node itself to be valid (i.e., the choice nodes' preconditions are satisfied and bindings can be filled). Likewise, if a node has continuation node children, at least one child must be valid for the parent to be valid (see Fig. 2).

A node-tree is viable if the following are true:

(1) All child nodes of any given node are of the same type
(2) If a node has choice node children, it is only viable if at least two children are viable
(3) If a node has continuation node children, it is only viable if at least one child is viable

5 SCENE SELECTION RULES

Scenes are drawn at random from the list of currently viable scenes, and authors may add preconditions to make certain scenes viable only in a particular context. However, Lume does have some built-in mechanisms to assist with scene selection and repetition suppression:

- Scenes may be given a priority by their author. Viable higher priority scenes will be chosen ahead of any lower priority scenes
- If two scenes have an equal priority band, the system will prioritize the scene that has been shown fewer times to the player, and will deprioritize scenes that have been shown recently
- In practice, scenes with tighter preconditions—that is, scenes which are more specific to a particular event needing to have occurred before—are more likely to fire than common ones.
- If priority bands are equal and the player has encountered two scenes an equal number of times, the system will select one at random
- Authors can force scenes on a given turn, e.g., "Always fire Scene X as the 4th Scene in the game," or "Fire scene X on the turn after this one if it's available"
- Authors can combine tight preconditions with high selection priority to make a scene extremely likely to fire if its (rare) conditions are met

6 INCREASING PROCEDURALITY WITHIN SCENES

While the Lume system offers highly variable plot structures through the ability to pull appropriate scenes from the pool of eligible scenes in any given moment, the true power of the Lume system comes from the fact that many elements of scenes are parameterized. Parameterized character bindings (see section on Bindings) are extremely powerful when combined with conditional generative text. This section discusses additional features that increase combinatorial potential *within* scenes.

6.1 Definite Clause Grammars

The Lume system takes advantage Prolog's built-in Definite Clause Grammars (DCGs), which offer a means to implement context-sensitive grammars. Context-sensitive grammars are able to carry out all of the functions of context-free grammars—structures widely used in other successful text-generation tools [26], [4])—while also allowing for more specific expansions of text based on context-specific information. In practice, this means that Lume authors can provide preconditions for text-expansions that allow for text to be expanded in certain ways if certain preconditions are met.

Additionally Lume allows for certain DCGs to override others, giving authors the opportunity to impose hierarchy and specifiy which DCGs they would prefer to be expanded if certain conditions are met. For example, an author could specify that a [Greeting] by default expands to either the phrase "Hi." or the phrase "Hello." But if the [Greeting] is performed by an AngryCharacter, override those expansions with "How could you do this?"

By default, eligible DCG terminals are selected at random with an equal distribution. Thus in our example above "Hi." and "Hello." would have equal chance of selection. Authors can "weight" the expansions in certain directions by adding die rolls as preconditions.

As with other text-generation tools, DCGs can also combine to make up other DCGs. Thus we could also author [SayName] to expand to "My Name is [CharName]" or "I go by [CharName]" and then author the DCG [Introduction] to expand to the combination of {[Greeting],[SayName]}.

6.2 Pronoun Replacement and Point of View

To further parameterize the content of scenes, Lume implements a pronoun replacement system that allows authors to write text (with markup) that is more broadly applicable to multiple character bindings. Thus rather than authoring

> "John killed Jane! She had it coming."

the author would instead write

> NPC1, " killed ", NPC2, ". ",
> NPC2, " had it coming."

The system interprets that in this case, NPC2 is a woman and substitutes the appropriate pronouns. Note that the first time NPC2 is called, it expands to NPC's name, and the second time, it expands to a pronoun. In the case in which the bindings are reversed, and NPC1 is Jane and NPC2 is John, the pronouns substitute appropriately.

Additionally the Lume system implements a dialogue function that takes the speaker into account. Thus, if we pass the same marked up sentence into some dialogue in which John is saying the line, he will appropriately declare:

> "I killed Jane. She had it coming."

This functionality is particularly useful for recall functions, in which authors can specify "Player killed NPC2" and different characters might approach the player with an accusation "We all know [you killed Jane]" or the player might come upon two characters gossiping "Did you hear [John killed Jane]?" The pronoun and point of view functionalities ensure that one piece of content can be used in as many places as possible, increasing the potential for content reuse to limit authorial burden.

7 CREATING REACTIVE NARRATIVES

The potential for highly reactive narratives in Lume comes from the interplay between the several combinatorial aspects of the system:

(1) At the scene level, scenes are selected to showcase the most relevant scene to the player at that moment in the story.
(2) Character dialogue within a scene is written in DCGs, thus is able to be highly dynamic. The lines themselves may have preconditions that ensure the most relevant line is selected for the current moment.
(3) Character dialogue may also utilize recall phrases to surface players' past decisions. So rather than "We question your leadership." the player might see "What kind of leader are you? You left Jane to die!" or with different events leading up to this scene, "What kind of leader are you? You stole John's dearest possession!" These can easily be authored into a scene's normal DCG dialogue.
(4) The decisions presented to players within a scene can also have preconditions, thus are highly dynamic and customizable to specific circumstances.
(5) Follow-up scenes may be specified by the author or left up to the system for selection.

These elements, taken with the apophenia that leads players to overestimate causal connections in emergent narratives even

when none are specifically modelled [28], present fertile ground for narratives to feel highly reactive.

7.1 Example Scene

Let us suppose we want to tell a simple story over three scenes that involves:

(1) A dramatic incident
(2) A reckoning incident
(3) A consequence incident

For our dramatic incident we might have a list of scenes to choose from. For a simple example, suppose two of these are:

(1) Flood Scenario
(2) Attacked Scenario

The flood scene base node would be authored with the following fields:

- Preconditions: There is an NPC in your party (NPC1)
- Bindings: NPC1 should be attached to the history of this event
- Instructions: [Description of flood], [NPC Reaction]

The descriptions in brackets under the Instructions field demonstrate descriptions that would exist as DCGs. This text is dynamically adapted so the description of the flood will be varied on each play. The NPC reaction could similarly be authored to feel appropriate to the NPC that binds to this scenario.

We want this scene to have two choices: leave NPC1 behind, or attempt a rescue (which, for the sake of simplicity of this example, is doomed to fail). Each choice is authored as a child node. The first child node (abandon our companion), has the following fields:

- Preconditions: (none)
- Bindings: (NPC1 is inherited, no new bindings)
- Choice Text: "Leave", NPC1
- Instructions: [Description of leaving], [NPC Reaction]
- Postconditions: NPC1 leaves your party; your relationship with NPC1 descreases
- Recall phrase: "You abandoned", NPC1, "during the flood"

The second child node (attempt a rescue) has the following fields:

- Preconditions: (none)
- Bindings: (NPC1 is inherited, no new bindings)
- Choice Text: "Attempt a rescue"
- Instructions: [Description of rescue attempt], [NPC Death]
- Postconditions: NPC1 dies Recall phrase: NPC1, "died during the flood"

See Fig. 3 for a representation of the scene tree.

The process for authoring a scene in which our party is attacked by wolves would follow a similar format. If both scenes are tagged as a "dramatic scene" in their scene ID, the system will select one of the scenes.

7.2 Follow-Up Scenes and Emergent Narrative

As the player continues to experience the narrative, follow-up scenes can use preconditions to build on previous narrative experiences. Fig. 4 demonstrates how leaving our companion in the flood provides context for future follow-up scenes. We might author the following reckoning scene in which some character is angry at us about something:

- Preconditions: You have recently decreased relationship with an NPC
- Bindings: That NPC should be attached to the history of this event (NPCa)
- Instructions: [NPCa expresses anger over [Recall negative event]]

In this case, we (the players) have recently upset Alice, so this scene is a good candidate for selection. Alice will approach us, tell us—in a way specific to Alice—that she is upset and we will attach the description of that event. The player would see an aggressive Alice ask some variation of "[How could you?!][You abandoned me during the flood]!", using the recall event we attached to that incident and with the appropriate pronouns to express that recall phrase from her point of view.

Fig. 5 demonstrates an alternate path to the same scene with angry Alice. This time, the system has selected that our dramatic scene is one in which your party is attacked by wolves. A companion comes to us later and we ask him not to tell Alice. This might immediately lower our relationship with Alice even though she is not in the scene and attach its own recall description to that event. The system might still pick the angry scene because we have had a drop in relationship. Alice approaches, this time with the appropriate recall.

There are some interesting things to note about this second scenario: we have reused the angry NPC scenario with no additional authoring and it has still yielded a narratively interesting result. Additionally, we did not model Bob telling Alice what happened; from the system's perspective the angry scene was selected in the second example purely because Alice's relationship decreased. But through DCGs we could potentially offer explicitly that Bob told Alice or leave that information off completely and let the player infer whether that happened. Finally, this angry NPC scene could just as easily bind to another character. If the player upset Bob recently, she might instead experience Bob approaching her, upset with her recent action.

While stories larger than 3 scenes require more robust rule-scaffolding to yield well-formed narratives across more complicated structures, this small example demonstrates excellent early results, and the system easily allows for the addition of these rules.

8 DISCUSSION

The primary goal of the Lume system is to strike a balance between *narrative dynamism*, the feeling that what's happening is one of many paths that could have been taken, and *narrative coherence*, the feeling that narrative events are causally following from player actions or logical NPC reactions. Several of Lume's features—and especially the combination of these features—offer fruitful steps toward these goals.

8.1 Toward Narrative Dynamism

8.1.1 Narrative Dynamism Through Scene Selection: Lume narratives are inherently dynamic, in that they are comprised of scenes that will always appear in different orders with different characters within them, given a large enough content pool. This architecture

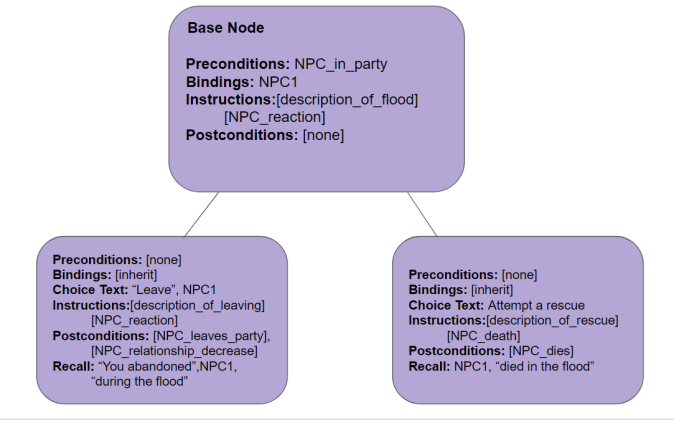

Figure 3: A representation of a basic scene.

means that authors must take care to impose their own narrative scaffolding.

8.1.2 Narrative Dynamism Through DCGs: DCGs ensure that players are seeing varied content on each playthrough. Well-formed DCGs can support huge combinations of possible text output, and the context-specific nature of Prologs DCGs mean that expansions can be tailored to current game circumstances.

8.1.3 Narrative Dynamism Through Conditional Choices: Preconditions on choice nodes mean that players may be presented with different choices in different contexts. Yet because of the nature of the content selection mechanisms, we do not have to worry that more choices will necessarily lead to greater authorial burden.

8.2 Toward Narrative Coherence

8.2.1 Narrative Coherence Through Bindings: A careful combination of bindings and preconditions allows us to construct coherent narratives. In particular, bindings allow us to have narrative throughlines that provoke causality in the player's mind. Bindings allow us to fill the appropriate character, place, or event into a narrative moment based on the current world state.

8.2.2 Narrative Coherence Through Content Control Flows: In a well-formed narrative, one scene is the logical outcome of another, and indeed there may be places where authors want to specify that if X happens, Y should happen in response. Lume gives authors control of the flow of narrative on a variety of explicit specificities by allowing authors to:

- Directly control which scene should fire next
- Specify that the next scene should be from certain content pools
- Specify that a scene from a certain pools should fire in the next X turns
- Indirectly steer toward certain scene relationships by authoring post-conditions to feed into other scenes' preconditions
- Indirectly steer flow through general scene priority

8.2.3 Narrative Coherence Through Recall Phrases: Recall phrases are useful devices to remind players that their decisions have changed the world, and changed characters' perceptions of them. This feature directly highlights the causal relationships in the narrative, and gives players a sense that their actions have caused the current narrative events. The fact that recall phrases are dynamic furthers the feeling of a deeply-tailored narrative experience.

8.2.4 Narrative Coherence Through Knowledge Representation: We have developed a set of rules and DCGs that allow characters to refer to other characters with generic tags if they don't know them. Thus characters could be "the man" or even "the man who [attacked us in the woods]" until he introduces himself as Robert. The knowledge representation capabilities are outside of the scope of this paper, but present interesting and promising future work.

8.2.5 Narrative Coherence Through Conditional Text Generation: In addition to DCGs offering a large possibility space for dynamic content, their context-specific nature means that we can ensure the most relevant expansion occurs. Thus our characters might speak in text that is specific to that character's voice, or might offer different reactions altogether based on their mood, the players' relationship, or a status. Locations might contain generic descriptions to be overridden by current story details. Narrations of different events might change based on the mood of the narrator. And so on.

8.3 Authoring Considerations in Lume

The authors of this paper have worked as professional narrative and technical designers at influential games and interactive narrative studios such as Telltale Games, Ubisoft, 2K Games, Radical Entertainment, Eastgate Systems, Sonderlust Studios, and the Expressive Intelligence Studio at UC Santa Cruz. The creation of Lume has largely grown out of the needs we found to be unaddressed with various systems at these organizations. A recurring theme has been a desire for a node-based system that accommodates greater levels of genuine dynamism [15]—the absence of which is a critique often leveled at games created with the Telltale Tool [10]—while maintaining narrative coherence. While we have not conducted a user-study to address whether the system does what we need—largely because such a study would not be able to determine such without a greater amount of authored content than is currently available—our experiences as seasoned authors have given us insights into the pros and cons of using the system so far.

Having authored in several interactive narrative systems, we find the qualitative experience of designing narratives in Lume to have different considerations than authoring from a choice-based perspective in a tool like the Telltale Tool or Ink [8] or Twine [11] or Storyspace [2]. Those systems follow an authoring pattern in which authors are generally writing in branches. Authoring in Lume feels more like authoring decks of cards, with many of the same design insights being applicable:

- Narratives only become interesting once there is a critical mass of cards to choose from.
- Narratives feel as coherent as the rules that govern them. Clean, coherent bucketing of scenes leads to greater coherence.
- Narratives are more dynamic with more content to select from

Figure 4: Even with additional scenes inserted for pacing, a simple storyline emerges from scenes selected from a pool. The decision from the first scene is recalled in the third.

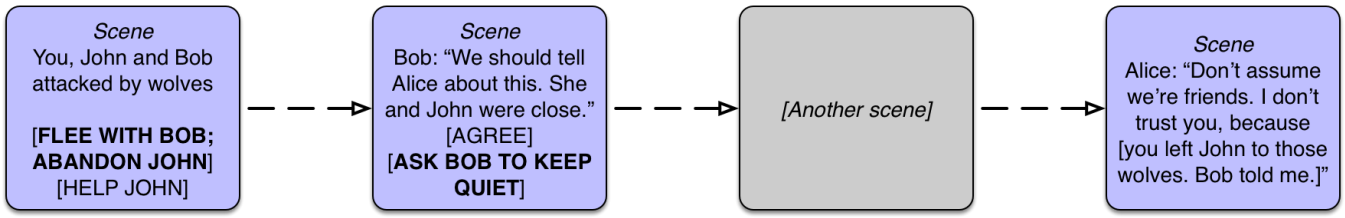

Figure 5: A more complex storyline emerges here. The more specific a scene's preconditions, the more reactive to player choice it feels. We can also reuse scenes at narratively appropriate moments with recall events filling in the correct details of the final scene.

- The most interesting scenes are the rare ones. The more conditions that must be met for a scene to fire, the more exciting it feels when it does.

Lume has shown potential for great narrative coherence, with the trade-off being that additional design effort must go into ensuring that rules, bindings, and control flow uphold that coherence. Lume has also shown potential for great dynamism, with the trade-off being that additional authoring effort is required. In addition to writing alternate scenes to fulfill the same narrative function, we find that in practice alternate DCG expansions should contain variations on form as well as variations on content to avoid feeling stilted, as though madlibs have been dropped in. Thus, in order to achieve both dynamism and coherence, the authorial burden is nontrivial.

We should also restate the we do not claim that Lume is the first system to offer the features detailed in this section. Other systems have conditional choices, such as Inkle games [9] or *King of Dragon Pass* [1]; *Facade* [20], and particularly *Prom Week* [21] make heavy use of recall phrases; dynamism through binding is a feature in StoryAssembler [6], Prom Week, and many HTN-based story generators. Instead, we believe that the particular way we have combined these features has potential for creating interesting narratives.

9 FUTURE WORK

While Lume offers first-results toward highly-tailored player experiences, it also raises interesting questions about best practices for designing within such a system. Currently Lume exists in a prototype phase that, while functional for creating the kinds of experiences detailed above, is extremely unfriendly for use by non-expert authors. Additionally, while capabilities for quests, knowledge representation systems, and dynamic nemeses are all present in the system currently, design best-practices for these features require a more systemic approach to narrative design that remains untested in terms of usability for non-expert narrative designers. Lume is currently being used to develop a full-scale narrative game, and we anticipate that many fascinating future questions will arise from that process.

ACKNOWLEDGMENTS

Mark Bernstein, Emily Short, Carl Muckenhoupt, Jacob Garbe, Aaron Reed, and Max Kreminski provided excellent conversations and feedback that contributed greatly to this paper.

REFERENCES

[1] A Sharp, LLC. 2015. King of Dragon Pass. Herocraft (pub). PC Game for Windows. https://store.steampowered.com/app/352220/King_of_Dragon_Pass/
[2] Bernstein, M. 2016. Storyspace 3. In Proceedings of the 27th ACM Conference on Hypertext and Social Media (HT '16). ACM, New York, NY, USA, 201-206.
[3] Bernstein M., Greco D. 2002. "Card Shark and Thespis: Exotic Tools for Hypertext Narrative." In: Wardrip-Fruin N, Harrigan P (eds) First Person. MIT Press, Cambridge.
[4] Compton K., Kybartas B., Mateas M. 2015. Tracery: An Author-Focused Generative Text Tool. In: Schoenau-Fog H., Bruni L., Louchart S., Baceviciute S. (eds) Interactive Storytelling. ICIDS 2015. Lecture Notes in Computer Science, vol 9445. Springer, Cham.
[5] Failbetter Games. 2009. Fallen London. http://fallenlondon.storynexus.com/
[6] Garbe, J. Kreminski, M. Samuel, B. Wardrip-Fruin, N., Mateas, M. 2019. "StoryAssembler: An Engine for Generating Dynamic Choice-Driven Narratives". In: The Fourteenth International Conference on the Foundations of Digital Games (FDG âĂŹ19), August 26âĂŞ30, 2019, San Luis Obispo, CA,USA.
[7] Golovchinsky, G. Marshall, C. C. 2000. "Hypertext Interaction Revisited." In Proceedings of Hypertext 2000, San Antonio, TX.
[8] Inkle. 2019. Ink. https://www.inklestudios.com/ink/
[9] Inkle. 2015. 80 Days. PC Game for Windows. https://store.steampowered.com/app/381780/80_Days/
[10] Klepek, P. 2017. "Telltale's Ancient Technology is Now Badly Hurting Their Games". Vice: Apr 21, 2017. https://www.vice.com/en_us/article/pg5k9n/telltales-ancient-technology-is-now-badly-hurting-their-games
[11] Klimas, C. 2018. Twine. https://twinery.org
[12] Kreminski, M. Wardrip-Fruin, N. "Mapping the Storylets Design SpaceâĂĺ In: International Conference on Interactive Digital Storytelling (pp. 160-164). Springer, Cham.

[13] Kybartas, B. Bidarra,R. 2018. A Survey on Story Generation Techniques for Authoring Computational Narratives. In: IEEE Transactions on Computational Intelligence and AI in Games TBP(99): 1-1. (2016)
[14] Levine, K. 2014. Narrative Legos: Building Replayable Narrative Out of Lots of Tiny Pieces. Game Developers Conference, 2014. San Francisco. https://www.gdcvault.com/play/1020434/Narrative.
[15] Litton, Z. Muckenhoupt, C. 2019. How Telltale Designed Tools for Efficient Narrative Development. NarraScope, 2019. Boston.
[16] Malloy, J. 1993. its name was Penelope. Watertown, MA: Eastgate Systems, Inc.
[17] Malloy, J. Marshall, C. 1996. Forward Anywhere. Watertown, MA: Eastgate Systems, Inc.
[18] Martens, C. Cardona-Rivera, R. 2017. Procedural Narrative Generation. Game Developers Conference, 2017. San Francisco. https://www.gdcvault.com/play/1024143/Procedural-Narrative
[19] Mason S. 2013. On Games and Links: Extending the Vocabulary of Agency and Immersion in Interactive Narratives. In: Koenitz H., Sezen T.I., Ferri G., Haahr M., Sezen D., CÌlatak G. (eds) Interactive Storytelling. ICIDS 2013. Lecture Notes in Computer Science, vol 8230. Springer, Cham.
[20] Mateas, M., Stern, A. 2003. Facade: An Experiment in Building a Fully-Realized Interactive Drama. Game Developers Conference, 2003: Game Design Track. San Jose, CA.
[21] McCoy, J., Treanor, M., Samuel, B., Mateas, M., Wardrip-Fruin, N. 2011. Prom Week: Social Physics as Gameplay. FDG '11Proceedings of the 6th International Conference on Foundations of Digital Games pp. 319âĂŤ321.
[22] McKee R. 2016. Dialogue: The Art of Verbal Action for Page, Stage, Screen. Twelve, New York.
[23] Nerial. 2016. Reigns. Devolver Digital (pub). iOS/Android app. v.1.0
[24] Pemberton, L. 1989. A Modular Approach to Story Generation. Fourth Conference of the European Chapter of the Association for Computational Linguistics.
[25] Reed, A. A. 2017. Changeful Tales: Design-Driven Approaches Toward More Expressive Storygames. Dissertation, Univ of California, Santa Cruz.
[26] Ryan J., Seither E., Mateas M., Wardrip-Fruin N. 2016. Expressionist: An Authoring Tool for In-Game Text Generation. In: Nack F., Gordon A. (eds) Interactive Storytelling. ICIDS 2016. Lecture Notes in Computer Science, vol 10045. Springer, Cham.
[27] Schrier, J. 2018. "Why Ubisoft is Obsessed with 'Games as a Service'." Kotaku. https://kotaku.com/why-ubisoft-is-obsessed-with-games-as-a-service-1822938255.
[28] Short, T.X. 2018. Procedurally Generating Personalities. Gamasutra. https://www.gamasutra.com/blogs/TanyaXShort/20161216/310387/Procedurally_Generating_Personalities.php.
[29] Short, T.X., Adams, T (eds.) 2017. Procedural Generation in Game Design. A K Peters/CRC Press: Boca Raton, FL.