Felt: A Simple Story Sifter

Max Kreminski, Melanie Dickinson, and Noah Wardrip-Fruin

UC Santa Cruz, Santa Cruz CA 95064, USA
{mkremins,mldickin,nwardrip}@ucsc.edu

Abstract. *Story sifting*, also known as *story recognition*, has been identified as one of the major design challenges currently facing interactive emergent narrative. However, despite continued interest in emergent narrative approaches, there has been relatively little work in the area of story sifting to date, leaving it unclear how a story sifting system might best be implemented and what challenges are likely to be encountered in the course of implementing such a system. In this paper, we present Felt, a simple query language-based story sifter and rules-based simulation engine that aims to serve as a first step toward answering these questions. We describe Felt's architecture, discuss several design case studies of interactive emergent narrative experiences that make use of Felt, reflect on what we have learned from working with Felt so far, and suggest directions for future work in the story sifting domain.

Keywords: emergent narrative · story sifting · content authoring

1 Introduction

The problem of *story sifting* involves the selection of events that constitute a compelling story from a larger chronicle of events. Often this chronicle is generated through the computational simulation of a storyworld, whose output consists of a profusion of events, many of which are relatively uninteresting as narrative building blocks. The challenge, then, is to sift the wheat from the chaff, identifying event sequences that seem to be of particular narrative interest or significance and bringing them to the attention of a human player or interactor.

Ryan, who introduced the term "story sifting" [33]—as well as its predecessor, *story recognition*[1]—has identified story sifting as one of four major challenges [34] currently facing work in the domain of interactive emergent narrative. Emergent narrative, which Ryan characterizes as the approach taken by many of both the greatest successes and failures in the area of story generation, remains an area of interest for interactive narrative design [20, 22] and narrative generation [1, 19] communities. Despite this ongoing interest in emergent narrative approaches, however, story sifting has received relatively little attention to date.

There has also been a recent wave of interest in *retellings* [8, 16, 17]—the stories players tell based on their play experiences in interactive narrative games—and in how design elements of games can facilitate and frustrate the player's

[1] As distinct from the natural language understanding term "story recognition", which refers to the identification of embedded story content in natural language text.

creative process. From this perspective, story sifters could be viewed as mixed-initiative creativity support tools [21] that help players narrativize their play experiences by surfacing sites of potential narrative interest as they emerge.

One goal of the *Bad News* project [38] was to learn lessons about story sifting needs that could be applied to the design of a computational system that performs story sifting. Unfortunately, a computational story sifter that incorporates the learnings from *Bad News* has yet to materialize. At the same time, our own recent work has involved the design and development of several interactive emergent narrative projects, and we have increasingly found ourselves making use of approaches that resemble story sifting, especially in designing interactive narrative systems that position the human interactor as a narrative co-author. As a result, we have begun to develop a simple story sifter geared primarily toward use in a mixed-initiative context—a system that assists players in the process of narrativizing their play experiences by helping them locate sites of potential narrative interest in a larger simulated storyworld.

Our system, Felt, implements a variation of one of the approaches to story sifting discussed by Ryan, namely that involving the human specification of interesting event sequences. In order to ensure that our human-specified event sequences are generalizable, we implement them not as literal sequences that must be matched exactly, but as *sifting patterns*: queries that seek out ordered sets of events matching certain criteria, with the possibility that other events may be interspersed between the events that are matched. In the remainder of this paper, we discuss related work in story sifting and adjacent areas; elaborate on the design of Felt; present three design case studies of in-development interactive narrative projects that make use of Felt; and discuss what we have learned from the design, development and application of Felt about story sifting in general.

Many of the design decisions that went into Felt are naïve. This is by design: at each turn, we attempted to do the simplest possible thing that had a reasonable chance of realizing our design intent. It is our hope that Felt functions as a *computational caricature* [40] of a query language-based story sifter, oversimplifying where necessary to ease development while still containing fully realized versions of the key features that are needed for the system to serve as an effective argument for the value of our approach.

2 Related Work

Ryan's original paper introducing the term "story recognition" [34] provides a partial list of existing systems that do something similar to story sifting, including *The Sims 2* [25], which recognizes sequences of events that match the early steps of pre-authored "story trees" and nudges the simulation engine to promote the completion of the story tree [4, 29]. Also of note is the Playspecs [30] system, which applies regular expressions to the analysis of game play traces. Samuel et al. have made some use of Playspecs in a narrative-focused context in *Writing Buddy* [37] and in the analysis of *Prom Week* playtraces [36].

Several systems discussed in Ryan's dissertation [33] also make use of story sifting. Foremost among these is *Sheldon County*, a generative podcast set in a listener-specific simulated American county. In *Sheldon County*, a sifter called *Sheldon* operates over a chronicle produced by the *Hennepin* simulation engine to recognize, extract and narrativize (in the form of podcast episodes) sequences of events that match certain human-defined sifting patterns. These patterns are defined as chunks of procedural Python code that search for candidate events and then bind relevant aspects of these events (such as the perpetrator of a crime) to pattern-specific variables. This approach is similar to the approach we use in Felt. In *Sheldon*, however, authoring a sifting pattern requires knowledge of both the Python programming language and the specific data structures used within the *Hennepin* engine, and even simple pattern definitions are often lengthy due to the verbosity of the procedural code used to implement them.

The "wizard console" in *Bad News* [38] provides an expert human interactor (the "wizard") with a view into the underlying simulation of a small American town. Behind the scenes of the main performance, the wizard uses the console to seek out narratively potent information about the state of the storyworld, and—in real time—relays this information to a human actor who is performing as one of the town's simulated inhabitants. The wizard console is essentially a Python interpreter that enables the wizard to examine the state of the simulation data structures. As such, it provides little computational support for story sifting, although the wizard may make use of a set of helper functions intended to make common sifting tasks easier. The wizard console makes no attempt to realize sifted stories as prose, leaving it largely up to the human actor to decide how to leverage the information gathered through sifting, and—like *Sheldon*—requires familiarity with both Python programming and the particulars of the underlying *Talk of the Town* simulation engine [35] to use effectively.

Dwarf Grandpa [12], an extension to the Legends Viewer interface for browsing *Dwarf Fortress* [3] world data, makes use of story sifting to extract and narrativize the lives of certain notable characters from the game world. Dwarf Grandpa performs story sifting exclusively in a backwards-looking manner, rather than attempting to sift in real time as the simulation runs, and performs only fully automatic sifting, without a human in the loop. Unlike many existing sifters, Dwarf Grandpa also performs the natural language generation needed to automatically present sifted stories as human-readable prose.

Caves of Qud's [11] biography generation system for notable historical characters [13] also makes use of story sifting. Biographies are generated by selecting a sequence of random actions for a character to perform, then running sifting patterns over these random events to retroactively justify each action with an in-world reason. Where no pre-existing reason for an action can be located, new facts about the world are generated on the fly to produce a working rationalization. Like Dwarf Grandpa, *Qud*'s biography generator operates fully automatically and realizes sifted stories as prose.

Rules-based simulationist narrative generation systems often provide some way for events to be directly dependent on or make direct reference to past

events, and therefore have some similarity to story sifting. Here we include systems such as Comme il Faut [27], the rules-based "social physics" system that underlies the social simulation game *Prom Week* [26], and Versu [10]. In both of these systems, characters may act in ways that are directly dependent on the presence or absence of a set of past events that meet a set of specified criteria—essentially a sifting pattern. This can arguably be viewed as a form of "internal story sifting": these systems recognize patterns of relevant past events, but only for internal use, and without surfacing the fact that a given pattern was recognized to the audience directly. In Ryan's terms, these systems lack *story support*: the presentation of system-recognized stories to an audience.

Prom Week in particular complicates this evaluation somewhat by presenting players with a list of all of the "social facts" that contribute to a given character's evaluation of the present social situation. Social facts are often directly tied to past events that have played out within the simulation. Arguably, the surfacing of these relevant past events to the player could be considered to be a form of story support, especially if the player is viewed as a co-author alongside the system rather than a mere experiencer of a totally system-curated story. However, in this case, narrativization of the sifted events does not occur within the system; it occurs totally within the player's head, if it occurs at all.

More generally, to describe an approach as making use of story sifting, it is arguably necessary for the underlying plot generation technique to be one that produces a profusion of events, including many mundane events about which the audience does not necessarily care. Many approaches to story generation that allow events to directly reference past events have some similarities to sifting-based approaches, but aim to exclusively produce narratively interesting events that are worthy of being surfaced to the audience. This includes many planning-based techniques [31, 43]. We recognize that these approaches may have much to offer the developers of sifting-based systems, but we do not include them under the label of "story sifting" here.

3 System Description

Felt is a query language-based story sifter coupled with a rules-based simulation engine. Events that have transpired in the storyworld are stored in the database as entities, and users of the system write queries—which we, following Ryan, refer to as *sifting patterns*—to identify scenarios and sequences of past events that might make for good narrative material. A sifting pattern is defined in terms of a set of *logic variables* to bind—effectively "slots" or "roles" into which certain database entities, such as events or characters, can be substituted—and a set of relations between these logic variables, which constrain the values that each variable is allowed to take. A sifting pattern could specify, for instance, that `eventA` must be an instance of the `betray` event type; that `eventA` must have taken place before `eventB`; that both events must have the same protagonist, a character `char`; and that `char` must have the `impulsive` trait. The system

will then consult the database and return a list of all possible combinations of variable bindings for the pattern as a whole.

In designing a Felt storyworld, users combine sifting patterns with several other features to define *actions*. The structure of actions is directly inspired by the structure of rules in Ceptre [23], a linear logic programming language for specifying interactive simulationist storyworlds. An action consists of a sifting pattern; an optional weighting function that decides how likely it is that this action should be performed, given a set of bindings for the logic variables defined in the sifting pattern; and a function that constructs an *event object* representing this action, which will be added to the database if this action is chosen to be performed. A minimal event object contains an autogenerated *timestamp*, which can be compared with the timestamps of other events to determine which happened first; a short string identifying its *event type*; and a *template string* into which the values of bound logic variables are substituted to produce a human-readable description of the event. It may also contain zero or more *effects*, which describe any other updates that must be made to the database if this event is accepted as part of the history of the storyworld, and possibly other properties on a case-by-case basis, such as the ID of an earlier event that was a direct cause of this event. Because actions are added to the database as events, Felt's story sifting features can be used to run sifting patterns over the history of everything that the simulated characters have said and done.

```
(eventSequence ?eventA ?eventB)
[?eventA "eventType" "betray"] [?eventA "actor" ?char]
[?eventB "eventType" "betray"] [?eventB "actor" ?char]
[?char "trait" "impulsive"]
(not-join [?char ?eventA ?eventB]
  (eventSequence ?eventA ?eventMid ?eventB) [?eventMid "actor" ?char])
```

Fig. 1. A moderately complicated Felt sifting pattern that will match a sequence of two betrayals perpetrated by the same impulsive character, with no other actions perpetrated by the same character (but arbitrarily many other events) in between.

By convention, in the projects we describe here as case studies, actions come in two flavors. *Internal* (or *reflection*) actions describe a character reflecting on past events. These actions typically generate "intent tokens" or "motive tokens", which represent a character's intent to act on a particular interpretation of these events in the future. *External* actions describe a character acting on a previously formed intent. These actions typically consume intent tokens and update the state of the world in some outwardly visible way. This separation ensures that intent tokens can be both produced and consumed in multiple different ways: many possible actions that produce the same type of intent token can serve as the motivation for many possible actions that all consume the same type of intent token, opening up the space of possible cause/effect relationships between

events. Additionally, in an emergent narrative system where actions are the player's primary window into what is happening in the simulated world, separate reflection actions help make it clear to the player that sifting patterns are at work behind the scenes, and that character behavior is meaningfully influenced by the history of past simulation events—sometimes in complex or sophisticated ways. This is one way in which we hope to address another of Ryan's four design challenges for interactive emergent narrative [34], namely that of *story support*: once a storyful sequence of events has been recognized, how should this be surfaced to the player? It also helps to ensure that we do not fall victim to the *Tale-Spin effect* [42] by failing to surface the interesting technical capabilities of our interactive narrative system to the player in a compelling way.

Internally, Felt uses the DataScript library [7] to store and query simulation state, including the history of events that have transpired within the simulated world. Felt sifting patterns translate directly into queries against a DataScript database, and are written in a minimal query language that desugars to a subset of Datalog, a simple logic programming language. DataScript provides facilities for storing, updating, and querying state as a set of simple facts of the form `[e a v]`; each fact represents an assertion that the database entity with integer ID `e` has an attribute named `a` with value `v`. A DataScript database is an immutable value: all operations that "update" the database in fact create a fresh copy of the database with the desired modifications, leaving previously stored versions of the database intact and unchanged. This property can be leveraged to snapshot the complete Felt simulation state and run queries against these snapshots while allowing the main copy of the simulation state to continue evolving, which we have found helpful during debugging. It has also enabled us to implement several features, described in section 4.2, that rely on the ability to perform actions in a speculative mode and easily undo them if they lead to unwanted consequences.

DataScript also provides several other useful features that assist with the authoring of sifting patterns. `not-join` query clauses enable testing for the non-existence of an entity that meets a certain set of criteria; this feature is frequently used to specify sifting patterns in which two target events must not be separated by any interceding events involving the same protagonist. Rules bundle groups of query clauses that are commonly used together under a common name; for instance, an `(eventSequence ?eventA ?eventB)` rule may simultaneously specify that both `eventA` and `eventB` refer to event entities and mandate that `eventA` must precede `eventB`. Rules may also be recursive, allowing for the implementation of a `(causalRelationship ?eventA ?eventB)` rule that will match not just direct causes but also indirect causes (separated by one or more intermediate stages of causation) of `eventB`.

4 Case Studies

4.1 *Starfreighter*

Starfreighter [14] is an in-progress procedural narrative game in which the player captains a small starship in a procedurally generated galaxy, completing odd jobs

to make a living while managing the needs of a small crew. The primary intent of this game was to test whether *parametrized storylets* [18]—atomic units of narrative content that, like Felt actions, are equipped with slots, preconditions, and effects—could be used to produce compelling emergent story arcs for procedurally generated characters.

It was while working alongside the developers of this game that we began to develop the earliest version of Felt. Like Felt, *Starfreighter* stores a chronicle of past events (framed as a sequence of "memories" accessible to the characters who participated in each event) and provides features for architecting storylets that refer directly to sequences of past events that meet certain criteria. As a result, *Starfreighter* storylets can contain instances of characters reflecting on sequences of past events, such as the circumstances that led them to leave their home planet or the evolution of their ongoing relationship with another character. Whenever the player completes a storylet, *Starfreighter* evaluates the sifting patterns of all other storylets to identify which ones it would currently make sense to present to the player, then chooses from this pool via simple weighted random selection—essentially using story sifting to implement a form of what Short terms *salience-based narrative* [39].

The early version of Felt used in *Starfreighter* differs significantly from the version we present in this paper. Most importantly, sifting patterns in this early version of Felt were not authored in terms of a true query language, but in terms of an ad-hoc collection of functions that retrieved entities from the game state in specific predefined ways. One notable consequence of this design decision was that, although storylets were equipped with sifting patterns that could bind a set of logic variables to appropriate values, the system would make no attempt to *unify* these variables with one another, meaning that there was no guarantee of being able to find all of the possible instantiations of a sifting pattern at any given time. Additionally, the authoring of new content became bottlenecked on the development of new functions that enabled the authors of sifting patterns to ask specific questions about the game state, forcing content authors to either learn how to write these often-complicated functions themselves (requiring deep knowledge of how the game state was structured) or else wait for the game's lead developer to implement the functions they had requested. Finally, because there was no straightforward way to get all of the possible instantiations of a sifting pattern in the context of the current game state, debugging was consistently difficult; in particular, if a sifting pattern was repeatedly failing to match a set of values for which it ought to succeed, the nondeterministic nature of sifting pattern resolution made it difficult to determine why.

Due to these issues, development of *Starfreighter* was temporarily suspended, with the intent to return to it in the future. Much of the existing *Starfreighter* content is now being rewritten using a modern version of Felt, which has helped to alleviate each of these issues.

4.2 *Cozy Mystery Construction Kit*

Cozy Mystery Construction Kit (*CMCK*) [15] is an in-progress AI-supported collaborative storytelling play experience (inspired by collaborative storytelling tabletop games like *Microscope* [32] and *The Quiet Year* [2]) in which two players collaborate with a computational system to write a mystery story about a small cast of simulated characters. *CMCK* uses Felt as a simulation engine for characters that sometimes perform actions autonomously and sometimes are directed to perform certain specific actions by players. It also uses Felt to help players locate and build on sites of narrative interest, such as a growing jealousy or resentment between two characters or a building conflict between two values—for instance, comfort and survival.

Of the case studies presented here, *CMCK* is the most explicitly focused on using story sifting to provide creativity support by recognizing emerging story structures as they unfold and suggesting elaborations on emergent patterns and themes. Several Felt features are especially useful in this context. Clear separation of actions that produce and consume "intent tokens" or "motivation tokens" enables players to ask the system questions about character motivation: for instance, "Who had a motive to harm this character?", or "What motives might this character currently want to act on?" This can be particularly useful when writing mystery stories. Because DataScript query evaluation is highly optimized, many Felt sifting patterns can be run over the database at once to provide players with a wide variety of suggestions as to what characters might reasonably do next. Additionally, because sifting patterns provide explicit slots for the characters and events they concern, *CMCK* can give players an interface that lets them filter action suggestions by specifying the values of one or more variables in advance. This enables players to (for instance) get a list of actions that a particular character might currently want to perform, or a list of actions that any character might want to perform in response to a particular past event.

Since the DataScript database in which Felt stores simulation state is an immutable value, *CMCK* can allow players to perform actions in a speculative mode, run queries against the updated database to decide whether they like the effects of these actions, and easily roll back to a previous database state if they do not. Immutability may also be leveraged to facilitate a sort of planning or goal-directed search over actions: because Felt actions (like planning operators) are defined in terms of preconditions and effects, search over Felt actions can be used to locate speculative future worlds where some specified set of conditions holds true. This can then be used to present users with an interface in which they specify a scenario they would like to bring about in the storyworld, and the system searches for a sequence of actions that might realize this scenario. The developers of *CMCK* plan to implement this feature in the future.

Another potential Felt-enabled feature that may be implemented in *CMCK* involves the automatic surfacing of "almost actions": dramatic actions that are *almost* possible, but currently invalid due to a small number of unmet preconditions. This is directly inspired by the feature with the same name in *Writing*

Buddy [37], and leverages Felt's per-action weighting functions to judge how dramatically significant a given action would be if performed.

CMCK is also notable for its use of story sifting to highlight character personality and subjectivity through sifting-driven reinterpretation of events. Each *CMCK* character holds several randomly selected *values* drawn from a pool of eight possible values, and these values are used in sifting patterns to influence how characters will interpret certain event sequences. Consider, for instance, a sequence of events in which a character forbids anyone from using the kitchen until a crime that took place there has been thoroughly investigated. A character who values comfort above all else may evaluate this sequence of events very differently than a character who values safety. Much as *Terminal Time* [24] narratively spins historical events to cater to the audience's ideological biases, and *Caves of Qud*'s biography generation system [13] retroactively decides how to interpret the motivations behind a character's randomly generated actions, *CMCK* characters engage in retroactive interpretation of events through the sifting of their own stories, one another's stories, and the stories of the world around them. Moreover, in *CMCK*, the differential interpretations that result from this process of sifting serve as the main driver of character conflict. In this sense, *CMCK* could be viewed as an instance of AI-based game design [9, 41] in which the AI process at the heart of the play experience is a story sifting engine.

4.3 *Diarytown*

Diarytown is an in-progress game in which players craft diary entries about their real life and watch as the described experiences are creatively enacted, extrapolated on, and respun through the lens of a simulated, personalized town. It is currently in active development and prototyping alongside the most recent version of Felt, and subject to design changes.

In the current version of the game, Felt is primarily used to recognize story patterns in a player's diary entries over time, allowing *Diarytown* to surface to the player different possible interpretations of things that have happened within their life. This underpins one of the project's primary design goals, of facilitating playful, generative reflection on one's life. Whenever a story pattern is recognized, *Diarytown* surfaces it to the player as a new scenery object within the simulated town. The player can then interact with this scenery object to view the recognized story, and can choose to edit the object's appearance and placement, or even to remove it from the town entirely if they reject the interpretation of their life's events that it represents. Multiple recognized story patterns that share many of the same attributes (for instance, a common focal character) might be collapsed over time into a single, larger scenery object, and these objects may thus gradually take on the role of symbols of larger patterns within the player's life (for instance, a monument to the player's ongoing relationship with a particular friend).

Players craft diary entries by composing terms from a symbolic action library consisting of actions, connectors, and modifiers designed to reflect common actions in a person's everyday life. Some are optionally parametrized with character

names, places, and other reference nouns, which the player defines during play. The parametrized nature of Felt actions make it ideal for representing elements of these complex diary entries as simulation actions.

Felt is also being used to simulate autonomous town activities and background characters that are partially conditioned on player-entered actions and character definitions. This integration of player-defined and autonomous actions allows us to playfully extrapolate on a player's account of their daily life, and leverage the expressive affordances of emergent narrative (which generally requires a large number of events to sift through) even when there are relatively few player diary entries.

In the context of the *Diarytown* project, Felt was introduced to four high-school-aged research interns, three of whom had some prior programming experience (primarily in Java) and one of whom had none. At the end of a single day of instruction, all four interns were able to author new actions (including sifting patterns) on their own. Within a week, they had authored 85 new actions without expert intervention.

5 Discussion

5.1 Authoring Sifting Patterns

When adopting an approach to interactive narrative that makes integral use of story sifting, the design and development of sifting patterns becomes part of the content authoring pipeline. As such, we made it one of our design goals for Felt to make the authoring of sifting patterns as easy and approachable as possible. As a result of this focus on approachability, we initially intended to provide sifting pattern authors with a large library of preauthored functions for accessing the database in certain specific ways, and thereby to avoid creating a situation in which sifting pattern authors had to learn how to interact directly with the complicated network of relationships between game entities.

In practice, however, we soon found that it was very difficult to anticipate in advance the full range of questions that a sifting pattern author might want to be able to ask about the game state. This made it near-impossible for us to create an adequate library of preauthored functions. As a result, we found ourselves turning instead toward the path of giving sifting pattern authors access to a "real" query language. Query languages are designed for flexibility, enabling the user to ask a wide variety of questions about the game state on an as-needed basis—including questions that no one specifically anticipated ahead of time.

It may, at first glance, seem counterintuitive that authoring can be made more approachable by presenting content authors with a query language they must learn. However, as argued by Nardi [28] and evidenced by the widespread success of the Tracery language [6] among users with little or no prior programming experience, people are generally quite good at learning simple formal languages when the language is tied to a task they want to perform. This is especially the case when a gentle on-ramp to query authoring is available: novice content

authors may start off using pre-composed sifting patterns without modification, graduate to making slight modifications of these pre-composed patterns, and eventually gain sufficient facility with the query language to author their own sifting patterns from scratch. Our success with having high-school-aged research interns on the *Diarytown* project write sifting patterns with little training supports the hypothesis that users can learn to write sifting patterns in a simple query language fairly quickly when they are provided with a robust library of examples to copy, paste, and modify.

5.2 Debugging Story Sifters

Another advantage of using a database with a full query language to store game state is that it greatly simplifies the process of debugging, enabling developers and content authors to write and run queries against the live database at any point. This stands in sharp contrast to the debugging experience in *Starfreighter*, where the opacity of the ad-hoc game state data structures made it difficult to explore the game state when trying to track down the reason for a sifting pattern's failure or misbehavior—especially for content authors, who had particular difficulty learning how different parts of the game state related to one another.

DataScript query evaluation is computationally inexpensive. This makes it tractable to get a list of all sifting patterns that are currently succeeding, including all possible sets of variable bindings that they could use, simply by running all of the available sifting patterns against the database in quick succession. This can significantly speed up debugging by making it visible at a glance whether or not a particular instantiation of a sifting pattern is currently possible, saving a substantial amount of time that a developer might otherwise have to spend manually testing sifting patterns they are attempting to debug.

Moreover, the DataScript queries that underlie Felt sifting patterns are partitioned into distinct clauses, which can be evaluated against the database individually or in subgroups as well as in the context of a complete query. We took advantage of the structured nature of our sifting patterns to implement a debugging helper function we refer to as `whyNot`. This function takes a sifting pattern as an argument, and can optionally also be supplied with a partial set of variable bindings for the pattern's logic variables. It then tests each clause of the sifting pattern in isolation, then each subgroup of clauses, until it identifies the set of clauses that are currently causing the pattern to fail. This information can then be reported to the pattern's developer, potentially saving them the work of manually stepping through the pattern line by line to identify why it is not succeeding when it ought to be.

5.3 Coupling Sifting and Simulation

Felt is a sifting engine coupled with a simulation engine. Strictly speaking, it is possible to make full use of Felt's story sifting features without making any use of its simulation engine. Sequences of events can be generated by an external process and then added to the database in a Felt-compatible form, enabling

the authoring of sifting patterns that operate over these externally generated events. However, in practice, it is often desirable to make use of sifting patterns within the definition of simulation actions, as this enables the straightforward authoring of character actions that involve characters reflecting on, interpreting, and responding to events that have transpired in the past. Therefore, in every project to date that has made use of Felt's story sifting features, Felt's simulation features have also been employed.

6 Conclusions and Future Work

One top priority for future work on Felt involves the design and development of a more sophisticated domain-specific query language for story sifting, with features that enable more concise expression of common concepts within sifting patterns. Currently, complicated Felt sifting patterns can be quite long and unwieldy. A more sophisticated query language could help ameliorate this, ideally without adding so much additional complexity that content authoring becomes bottlenecked on the development of expertise as a user of the query language.

Felt already makes extensive use of sifting patterns, but we have as of yet made no attempt to implement what Ryan refers to as *sifting heuristics*: nonspecific, high-level computational models of an event sequence's storyfulness, which may be used to guide a story sifting system to prefer some event sequences over others. For this, we may be able to draw on general-purpose models of event relatedness, including Indexter [5]: a computational model of event relatedness based on event recall in human memory. Of the five major contributing factors to perceived event relatedness that the Indexter model describes, many existing Felt sifting patterns make use of at least three (namely searching for sequences of events that share a common protagonist, causal relatedness, and common intentionality), and Felt's explicit modeling of causality and intentionality may make it a good testbed for an Indexter-inspired set of sifting heuristics.

More generally, it is our hope that, by presenting this system, we will encourage the development of a wide variety of approaches to story sifting. The query language-based approach we explore here is only one of many possible approaches, and we have only presented a first step toward the realization of our own preferred approach. We also hope that the existence of a "reference" story sifter will inspire the design of new kinds of interactive narrative experiences based on story sifting technology—particularly experiences that use sifting to provide creativity support for the human interactor in a collaborative storytelling context.

Acknowledgements The authors would like to thank Megna Anand, Anish Kashyap, Daniel Man, and Akhil Vemuri for their assistance in testing, debugging, and authoring content for Felt and *Diarytown*.

References

1. Adams, T.: Emergent narrative in *Dwarf Fortress*. In: Procedural Storytelling in Game Design, pp. 149–158 (2019)
2. Alder, A.: The Quiet Year, https://buriedwithoutceremony.com/the-quiet-year. Last accessed 17 Jul 2019
3. Bay 12 Games: Slaves to Armok: God of Blood Chapter II: Dwarf Fortress (2006)
4. Brown, M.: The power of projection and mass hallucination: Practical AI in The Sims 2 and beyond. Presented at AIIDE (2006)
5. Cardona-Rivera, R.E., Cassell, B.A., Ware, S.G., Young, R.M.: Indexter: A computational model of the event-indexing situation model for characterizing narratives. In: Proc. Computational Models of Narrative, pp. 34–43 (2012)
6. Compton, K., Kybartas, B., Mateas, M.: Tracery: An author-focused generative text tool. In: Interactive Storytelling (2015)
7. DataScript, https://github.com/tonsky/datascript. Last accessed 15 Jul 2019
8. Eladhari, M.P. Re-tellings: The fourth layer of narrative as an instrument for critique. In: Interactive Storytelling (2018)
9. Eladhari, M.P., Sullivan, A., Smith, G., McCoy, J.: AI-based game design: Enabling new playable experiences. UC Santa Cruz Baskin School of Engineering (2011)
10. Evans, R., Short, E.: Versu: A simulationist storytelling system. IEEE Transactions on Computational Intelligence and AI in Games, **6**(2), 113–130 (2014)
11. Freehold Games: Caves of Qud (2020)
12. Garbe, J.: Simulation of history and recursive narrative scaffolding, http://project.jacobgarbe.com/simulation-of-history-and-recursive-narrative-scaffolding. Last accessed 15 Jul 2019
13. Grinblat, J., Bucklew, C.B.: Subverting historical cause & effect: Generation of mythic biographies in *Caves of Qud*. In: Proc. FDG (2017)
14. Kreminski, M.: Procedural narrative design with parametrized storylets. Presented at GDC (2019)
15. Kreminski, M., Acharya, D., Junius, N., Oliver, E., Compton, K., Dickinson, M., Focht, C., Mason, S., Mazeika, S., Wardrip-Fruin, N.: Cozy Mystery Construction Kit: Prototyping toward an AI-assisted collaborative storytelling mystery game. In: Proc. FDG (2019)
16. Kreminski, M., Samuel, B., Melcer, E., Wardrip-Fruin, N.: Evaluating AI-based games through retellings. In: Proc. AIIDE (2019)
17. Kreminski, M., Wardrip-Fruin, N.: Generative games as storytelling partners. In: Proc. FDG (2019)
18. Kreminski, M., Wardrip-Fruin, N.: Sketching a map of the storylets design space. In: Interactive Storytelling (2018)
19. Kybartas, B., Bidarra, R.: A survey on story generation techniques for authoring computational narratives. IEEE Transactions on Computational Intelligence and AI in Games, **9**(3), 239–253 (2017)
20. Kybartas, B., Verbrugge, C., Lessard, J.: Expressive range analysis of a possible worlds driven emergent narrative system. In: Interactive Storytelling (2018)
21. Liapis, A., Yannakakis, G.N., Alexopoulos, C., Lopes, P.: Can computers foster human users' creativity? Theory and praxis of mixed-initiative co-creativity. Digital Culture & Education **8**(2), 136–153 (2016)
22. Louchart, S., Truesdale, J., Suttie, N., Aylett, R.: Emergent narrative: Past, present and future of an interactive storytelling approach. In: Interactive Digital Narrative: History, Theory and Practice, pp. 185–199 (2015)

23. Martens, C.: Ceptre: A language for modeling generative interactive systems. In: Proc. AIIDE (2015)
24. Mateas, M., Domike, S., Vanouse, P.: Terminal Time: An ideologically-biased history machine. AISB Quarterly, Special Issue on Creativity in the Arts and Sciences, **102**, 36–43 (1999)
25. Maxis: The Sims 2. Electronic Arts (2004)
26. McCoy, J., Treanor, M., Reed, A.A., Mateas, M., Wardrip-Fruin, N.: Prom Week: Designing past the game/story dilemma. In: Proc. FDG (2013)
27. McCoy, J., Treanor, M., Samuel, B., Reed, A.A., Mateas, M., Wardrip-Fruin, N.: Social story worlds with Comme il Faut. IEEE Transactions on Computational Intelligence and AI in Games, **6**(2), 97–112 (2014)
28. Nardi, B.A.: A small matter of programming: Perspectives on end user computing. 1st edn. MIT Press, Cambridge, Mass. (1995)
29. Nelson, M.J.: Emergent narrative in The Sims 2, http://www.kmjn.org/notes/sims2_ai.html. Last accessed 18 Jul 2019
30. Osborn, J.C., Samuel, B., Mateas, M., Wardrip-Fruin, N.: Playspecs: Regular expressions for game play traces. In: Proc. AIIDE (2015)
31. Porteous, J.: Planning technologies for interactive storytelling. In: Handbook of Digital Games and Entertainment Technologies, pp. 393–413 (2017)
32. Robbins, B.: Microscope, http://www.lamemage.com/microscope. Last accessed 17 Jul 2019
33. Ryan, J.: Curating Simulated Storyworlds. University of California, Santa Cruz (2018)
34. Ryan, J.O., Mateas, M., Wardrip-Fruin, N.: Open design challenges for interactive emergent narrative. In: Interactive Storytelling (2015)
35. Ryan, J.O., Summerville, A., Mateas, M., Wardrip-Fruin, N.: Toward characters who observe, tell, misremember, and lie. In: Proc. AIIDE (2015)
36. Samuel, B. Crafting Stories Through Play. University of California, Santa Cruz (2016)
37. Samuel, B., Mateas, M., Wardrip-Fruin, N.: The design of *Writing Buddy*: A mixed-initiative approach towards computational story collaboration. In: Interactive Storytelling (2016)
38. Samuel, B., Ryan, J., Summerville, A.J., Mateas, M., Wardrip-Fruin, N.: *Bad News*: An experiment in computationally assisted performance. In: Interactive Storytelling (2016)
39. Short, E.: Beyond branching: Quality-based, salience-based, and waypoint narrative structures, https://emshort.blog/2016/04/12/beyond-branching-quality-based-and-salience-based-narrative-structures. Last accessed 17 Jul 2019
40. Smith, A.M., Mateas, M.: Computational caricatures: Probing the game design process with AI. In: Proc. AIIDE (2011)
41. Treanor, M., Zook, A., Eladhari, M.P., Togelius, J., Smith, G., Cook, M., Thompson, T., Magerko, B., Levine, J., Smith, A.: AI-based game design patterns. In: Proc. FDG (2015)
42. Wardrip-Fruin, N.: Expressive Processing: Digital fictions, computer games, and software studies. 1st edn. MIT Press, Cambridge, MA, USA (2009)
43. Young, R.M., Ware, S., Cassell, B., Robertson, J.: Plans and planning in narrative generation: A review of plan-based approaches to the generation of story, discourse and interactivity in narratives. Sprache und Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative **37**(1–2), 41–64 (2013)