Authoring for Story Sifters

Max Kreminski, Noah Wardrip-Fruin, and Michael Mateas

Abstract We discuss the issues of authoring for *story sifters*: systems that search for compelling emergent narrative content within the vast chronicles of events generated by interactive emergent narrative simulations. We describe several different approaches to the authoring of *sifting patterns* that specify how to locate particular kinds of narratively potent situations; address the relationship between sifters and the simulations they operate over from an authoring perspective; and sketch several possible approaches to the authoring of *sifting heuristics*, or high-level encodings of what makes for a compelling story that could be used to guide a sifter's behavior.

1 Introduction

Interactive emergent narrative (IEN) [15, 21, 31, 38] is an approach to interactive narrative design in which narrative is allowed to emerge organically from open-ended interactions between autonomous simulated characters, as well as the actions of the human player. Like many other approaches to interactive narrative design, IEN attempts to solve the *narrative paradox* of reconciling open-ended interactivity with the communication of a coherent story [19].

Most existing approaches to interactive narrative design take a *top-down* approach to the narrative paradox: they attempt to ensure narrative quality by allowing only events that follow a preordained high-level plot structure to occur. For example, in linear interactive storytelling (often employed in many commercial story games), the player is able to interact within and between story scenes (plot points) but with no influence on their linear order. In branching interactive storytelling, the space

M. Kreminski (✉) · N. Wardrip-Fruin · M. Mateas
University of California, Santa Cruz, 1156 High St, Santa Cruz, CA, USA
e-mail: mkremins@ucsc.edu

N. Wardrip-Fruin
e-mail: nwardrip@ucsc.edu

M. Mateas
e-mail: mmateas@ucsc.edu

C. Hargood et al. (eds.), *The Authoring Problem*, Human–Computer Interaction Series,
https://doi.org/10.1007/978-3-031-05214-9_13

of all possible story traces is pre-authored as a graph structure, often with choice points explicitly presented to the player. And in *strong story* generative narrative approaches [22, 27] such as story planning [26, 39], the system reasons about story structure to generate linear or branching stories with a focus on story-centric characteristics such as causality.

IEN, in contrast, takes a *bottom-up* approach to the resolution of the narrative paradox, sacrificing fine-grained authorial control over plot structure in exchange for a greater degree of novelty and responsiveness to player action. In IEN, because the player and the simulated characters are free to take actions that don't line up with a preordained plot structure, the actions they take can vary significantly from one playthrough to the next, and the player-perceived narrative outcomes of this open-ended interaction can often surprise even the people who created the simulation.

Canonical works of IEN (such as *Dwarf Fortress* [1], *The Sims* [2, 6, 24], and *Stellaris* [17]) are known not only for their propensity to generate compelling and unexpected stories but also for their tendency to overwhelm players with the sheer volume of narrative content that they produce. Many of these works present players with complicated user interfaces that allow them to access a great deal of detailed information about the simulated storyworld, but at the cost of requiring users to spend a great deal of time learning to use this interface before they can reliably get compelling stories to emerge [16]. From a narrative design perspective, the central problem with IEN is one of unpredictability: because there is no central plot thread in relation to which the importance of individual events can be gauged, the system has no way to reliably determine which of the many events that take place within the storyworld are likely to hold particular narrative significance for the player. As a result, the most common failure condition for IEN play experiences involves the dissolution of the player-perceived story into a structureless mess, breaking the perception of narrativity [32] and causing players to understand the events of play not as a story but as "just one damn thing after another" [29, p. 4].

Story sifting [29, 31] attempts to address the problems of overwhelm and structurelessness in works of IEN by augmenting the underlying simulation (which is responsible for generating narrative events) with an additional technical system: the *story sifter*, which aims to detect narrative events or event sequences that make for compelling *narrative material*. Sifting thus allows the adoption of an 'overgenerate and test' approach to storyworld simulation, in which simulations are allowed to generate a wide variety of surprising juxtapositions; sifters are tuned to detect and surface the most interesting narrative situations that emerge from the simulation; and the overwhelmingly vast amounts of uninteresting or nonsensical material *also* generated by the simulation along the way can be downplayed or dismissed, allowing for a coherent story to solidify. James Ryan (who introduced the term 'story sifting') refers to this IEN design strategy as the *curationist* approach [29, p. 6].

However, beyond the known issues of authoring for IEN [20], story sifting introduces new authoring challenges of its own. In particular, current approaches to story sifting are heavily reliant on human-authored *story sifting patterns*: short blocks of code that a sifter can execute to detect instances of a particular type of narratively potent situation that have emerged within the storyworld. Additionally, sifting also

has implications for simulation authoring, particularly around the need to keep track of causality relationships between events at the simulation level and the possibility of integrating sifting into simulation design. And finally, although there has been little concrete research in this direction to date, sifting could also be augmented by *sifting heuristics*. These are higher level, more generic descriptions of what makes emergent narrative content potentially compelling. Such heuristics could be used to prioritize some sifting pattern matches over others when deciding what narrative material to highlight, though identifying these heuristics is still an open research problem.

In this chapter, we discuss these three key authoring issues. First, we discuss the challenge of sifting pattern authoring and present a brief history of attempts to improve the ergonomics of writing sifting patterns. Second, we consider the issues of simulation design for curationist IEN experiences and the need to construct simulations in sifting-compatible ways. And third, we briefly discuss the possibility of developing higher level sifting heuristics that could further improve the authorial leverage [5] of story sifting as an approach.

2 Authoring Sifting Patterns

Modern story sifters make extensive use of *story sifting patterns* to detect emergent narrative content that might be worth incorporating into a story. A sifting pattern is a block of code that specifies how to find instances of a particular kind of narratively potent situation that might emerge within a storyworld, for instance, an escalating cycle of revenge between two characters; a character who is consistently unable to hold down a job; or a sequence of events in which a social contract (such as the expectation that hosts do not harm their guests) is betrayed. These 'nuggets' of potentially interesting narrative content can then be woven—either by a human interactor, a computational system, or both working together—into a coherent story.

The more sifting patterns a sifter has at its disposal, the wider the range of emergent microstories that it can detect and reason about, and the better its ability to respond to the unexpected consequences of player interaction. Consequently, a number of efforts have recently been made to improve the efficiency of sifting pattern authoring. In this section, we briefly recount the history of these efforts.

2.1 Procedural Sifting Patterns

The term 'story sifting' was first employed to describe the role of the *wizard* (performed by a member of the design team) in the simulation-driven interactive theater experience *Bad News* [33]. The wizard is responsible for manually searching for interesting narrative material in a Talk of the Town [30] simulation. To perform this search, they make use of the *wizard console*, a Python REPL equipped with a number of predefined functions for conveniently executing specific types of queries against

the full simulation state. Attempts to automate *Bad News*'s wizard role resulted in the Sheldon sifter [29, p. 657], which executes sifting patterns specified as chunks of procedural Python code against a Talk of the Town-like simulation state to identify sets of interrelated storyworld entities (such as events and characters) that meet certain criteria. Below is an example of a Sheldon sifting pattern, which is executed against many possible `candidate` events to find those representing the enactment of an *arson revenge scheme* (in which a character who has been harmed by another character burns down a building belonging to that character as a means of getting revenge) and bundle them with some relevant context for narration:

```
self.match = (
  candidate.name == "set-fire" and candidate.find_ancestor(
    name="hatch-revenge-scheme",
    initiator=candidate.initiator
  )
)
if self.match:
  self.set_fire = candidate
  self.hatch_scheme = (
    candidate.find_ancestor(
      name="hatch-revenge-scheme",
      initiator=self.set_fire.initiator
    )
  )
  self.arsonist = self.hatch_scheme.binding("arsonist")
  self.target = self.hatch_scheme.binding("target")
```

Though this example is relatively readable for an experienced programmer, it also highlights some of the weaknesses of the procedural (as opposed to declarative) approach to specifying sifting patterns. In particular, it makes heavy use of chained object graph traversal to access event sequences and properties of matched events, limiting the ability of sifting patterns to flexibly traverse the graph 'in reverse'. The `find_ancestor` method on event data structures represents a particularly thorny part of the Sheldon API, since it forces all event sequence access to begin at the last event in sequence unless the simulation authors also define a mirrored `find_descendant` function (thereby increasing the authoring burden on the simulation side). In general, this example illustrates how the procedural (non-declarative) approach to writing sifting patterns ties the pattern strongly to the implementation details of the simulation. Ideally, we would like to be able to specify sifting patterns independently of these implementation details. Additionally, because Sheldon patterns are expressed in plain Python code, potential authors of Sheldon patterns must learn the syntax and semantics of general-purpose Python language constructs (such as method calls, boolean operators, and `if` statements) before they can write patterns effectively. This reduces the approachability of pattern authoring to those with limited programming experience.

2.2 Declarative Sifting Patterns

Felt [14] attempts to alleviate the difficulty of writing procedural sifting patterns by instead applying a *declarative* approach to sifting pattern specification. Felt patterns specify what to find instead of how to find it, and are expressed in a small domain-specific query language that compiles down to a subset of Datalog instead of a Turing-complete programming language. Consequently, they are often more concise than equivalent Sheldon sifting patterns; can perform bidirectional traversal of the entity graph without any extra authoring effort on the simulation side; and can be authored by people with less programming experience, since the surface area of Felt as a language is much smaller than that of Python or a similar scripting language.

Felt sifting patterns look like the following:

```
(eventSequence ?e1 ?e2)
[?e1 eventType hatchRevengeScheme] [?e2 eventType setFire]
(contributingCause ?e1 ?e2)
[?e1 actor ?arsonist] [?e2 actor ?arsonist] [?e2 target ?target]
```

Like the example Sheldon sifting pattern listed above, this pattern locates instances of an arson revenge event sequence in which an `?arsonist` character burns down a building belonging to another character as part of a revenge scheme against them. Identifiers preceded by a `?` character represent *logic variables*, which are bound to concrete values when an instance of the pattern is successfully found. Square-bracketed clauses (such as `[?e1 actor ?arsonist]`) represent assertions that the entity on the left-hand side (here, `?e1`, or the first event in the matched sequence) has an attribute with the name in the middle (`actor`) whose value is the entity or constant on the right (`?arsonist`, or the character responsible for the arson scheme). Equality checks are often handled by *unification*: here, we specify that the actor for the first and second events in the sequence must be the same character by assigning both of them to the same logic variable, `?arsonist`, so that only matches in which both events have the same actor will succeed. Meanwhile, clauses surrounded by parentheses (such as `(contributingCause ?e1 ?e2)`) invoke simulation-specific inference rules that can be used to make judgments about the relationships between entities—here, to judge whether the first event in sequence (`?e1`) is causally related to the second (`?e2`).

A small authoring study of Felt [14] found that relatively programming-inexperienced users (four high school-aged research interns) were successfully able to use Felt to write working sifting patterns after one day of training. However, they used only a minimal subset of the Felt language constructs available to them and did not make full use of the available simulation domain constructs, suggesting that further guidance in exploring the space of possible sifting patterns would be necessary to assist novice programmers in making full use of story sifting affordances.

In addition to the approach taken by Felt, inspiration for future declarative approaches to story sifting may be found in the approaches taken by Playspecs [25], which apply regular expressions to the recognition of patterns (sometimes narrative)

in gameplay traces but are limited in expressiveness by their inability to capture variable bindings; by prior work on plan recognition in narrative domains [3], some approaches to which closely resemble story sifting from a technical perspective; and by the use of story intention graphs for analogy search between plot structures [7], which could be leveraged for sifting via the analogical comparison of simulation outputs against structural patterns extracted from known-good stories.

2.3 Sifting Pattern Authoring Tools

A small ecosystem of authoring tools and higher level domain-specific languages based on Felt have emerged, with each presenting a slightly different form of assistance to users in the definition of Felt sifting patterns.

Synthesifter [18] (Fig. 1) aims to support the authoring of Felt sifting patterns by presenting users with an example-based interface for pattern specification. Once users provide a small number of concrete example event sequences matching their intended sifting pattern, Synthesifter uses inductive logic programming [23] to automatically synthesize a sifting pattern capable of matching these sequences, and presents the user with further possible matches of this pattern against a corpus of test events. Users can then refine the synthesized sifting pattern by marking these additional matches as positive or negative examples, or modify the synthesized pattern directly to get live feedback on which event sequences are matched by their modified pattern. By obviating the initial need to create new sifting patterns by writing code from scratch and using program synthesis to introduce new syntactic and semantic concepts in the sifting pattern language to the user, Synthesifter provides the user with well-formed concrete examples of how to use potentially unfamiliar parts of the Felt language and/or simulation domain, and thereby aims to mitigate the tendency of novice Felt users to use only a limited subset of the available constructs.

Centrifuge [9] (Fig. 2) is a visual editor for Felt sifting patterns that uses a node-graph model to make the Felt syntax more approachable. Elements of the Felt syntax and the simulation domain are represented as nodes, and connections between these nodes indicate the relationships between pattern-relevant simulation domain entities. This approach helps users avoid low-level syntax errors and view the pattern as a whole graphically, with the goal of making the connections between entities clearer—especially in complex patterns containing many interrelated entities. It also provides a palette of constructs that can be added to a pattern, allowing users to more readily explore the space of possible patterns.

And finally, **Winnow** [11] is a higher level domain-specific query language for story sifting that aims to save authoring effort by asking users to write a smaller number of explicitly staged sifting patterns, which can be executed *incrementally* to identify partial instances of desired microstories (e.g. the first few events of an arson revenge event sequence) before the sequence has run to completion and without

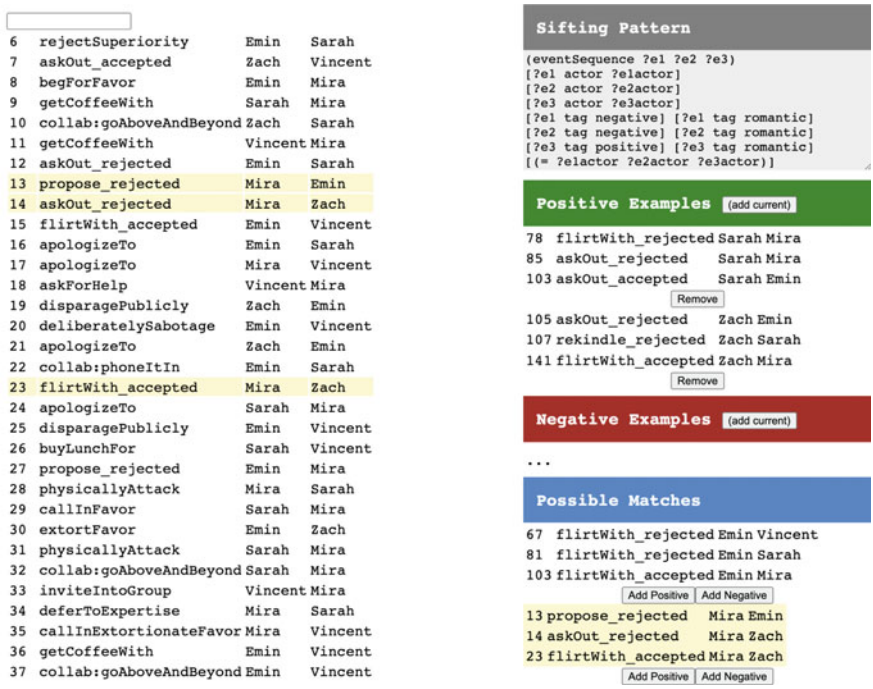

Fig. 1 Screenshot of the Synthesifter user interface (taken from [18]). On the left sits a scrolling, filterable log of all events that have occurred in the storyworld so far, allowing the user to select event sequences to use as examples. On the right sits an editable view of the current synthesized sifting pattern; the sets of positive and negative examples the user has provided; and the set of additional matches for the current candidate sifting pattern, which the user can add as positive or negative examples

any extra authoring effort. Consider the following Winnow translation of a slightly expanded `arsonRevenge` sifting pattern:

```
(pattern arsonRevenge
  (event ?harm where
    tag: harm, actor: ?victim, target: ?arsonist)
  (event ?scheme where
    eventType: hatch-revenge-scheme,
    actor: ?arsonist, target: ?victim,
    (ancestor ?harm ?scheme)),
  (event ?arson where
    eventType: set-fire, actor: ?arsonist, target: ?victim,
    (ancestor ?scheme ?arson)))
```

By explicitly incorporating the initial `?harm` event that leads to the revenge scheme into the sifting pattern and dividing the pattern into three explicit stages (one per matched event), we enable Winnow to automatically detect instances in which

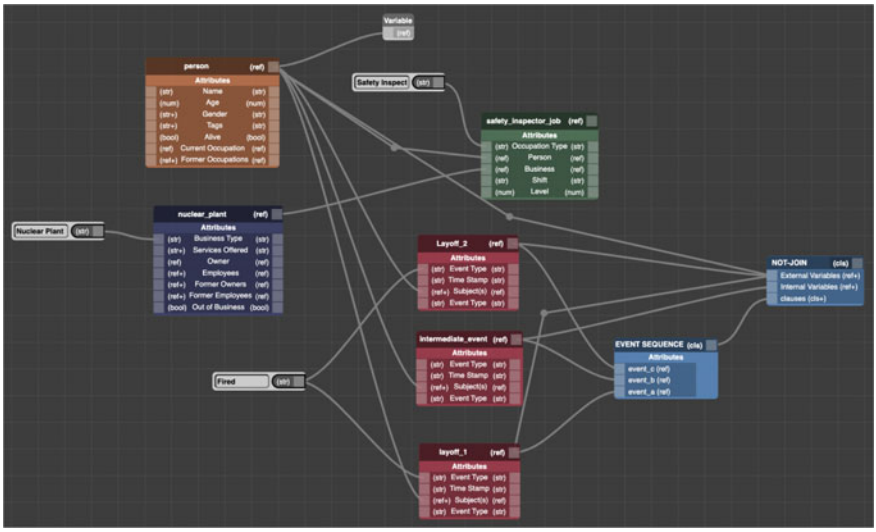

Fig. 2 Partial screenshot of the Centrifuge user interface, showing the graphical specification of a moderately complex sifting pattern. The depicted pattern is used to find instances of a nuclear plant safety inspector who has been fired twice in a short time period, without any other interceding life events

the first two events of the sequence (or any other prefix) have taken place, but the remaining events have not yet transpired. This allows for the procedural generation of foreshadowing for later events in the sequence; the suggestion or promotion of simulation actions that would advance this partially-formed microstory; and the capacity for avoidance of actions that would cut this microstory off before it has the chance to run to completion. To perform similar partial matching with Felt patterns alone would require pattern authors to maintain several partial variants of each pattern in parallel with the complete version; this increases the likelihood that errors will be introduced in the copying process, as well as the burden of synchronizing changes between the full pattern and its variants.

Though the tools and languages discussed in this section have introduced substantial subjective improvements to sifting pattern authoring processes from the authors' perspective, little evaluation of pattern authoring tools has been done, and none of these tools have been put through a formal user study at the time of this writing. Consequently, one potentially beneficial direction for future work in this area would be to perform a more thorough evaluation of the strengths and weaknesses of these authoring tools, particularly for less programming-experienced users.

3 Authoring Siftable Simulations

Beyond the authorship effort that is put into the construction of story sifting patterns appropriate for a particular emergent narrative domain, creators of IEN systems also have the option of crafting simulations with sifting in mind. This entails additional authoring effort at the simulation level, but can make it substantially easier to write sifting patterns that match relevant narrative situations. In this section, we describe three major levels of engagement with sifting at the level of simulation authoring.

3.1 Authoring Sifters for Existing Simulations

One advantage of story sifting as an approach is that it can be applied to the output of a simulation that was created without story sifting in mind. However, this often requires the construction of an adaptation layer that transforms the output of the simulation engine into a form that is more amenable to sifting—typically including what Ryan calls a *chronicler*, or a system that extracts a list (i.e. a 'chronicle') of all the potentially narratively significant events that have transpired in a storyworld's history [29, p. 236].

A number of chroniclers have been authored for existing IEN systems, including several distinct chroniclers (with slightly different aims) created to extract event sequences from the *Blaseball* simulation[1] and the Legends Viewer chronicler for *Dwarf Fortress*[2]. Legends Viewer is notable because it also provides some lightweight interactive sifting affordances on top of the extracted data, and because it has been used as a base for autonomous sifter development—for instance, by the Dwarf Grandpa project [8]. The creators of these chroniclers often need to exercise editorial judgment as to how the continuous output of an IEN system can best be quantized into discrete events: there is a balance to be struck in chronicler authoring between capturing enough data that a wide variety of expressive sifting patterns can be written over the data, and providing a sufficiently summarized view of the data that sifters do not get bogged down in considering many narrative-irrelevant events (e.g. movement events with little narrative content) when executing sifting patterns.

3.2 Co-designing a Simulation and Its Sifter

One difficulty of sifting the output of a simulation that was not designed for story sifting is that information about the causality relationships between events (which plays an important role in narrative) is not preserved or made retroactively available by most simulations. Consequently, Ryan argues that simulation authors who

[1] https://sibr.dev/apis.

[2] http://www.bay12forums.com/smf/index.php?topic=154617.0.

intend their simulations to be amenable to curation should ensure that the simulation performs *causal keeping* in its recording of events [29, p. 162], taking note of which events led to other events and making these causality relationships available alongside the records of the events themselves.

More broadly, in authoring simulation actions, it can be beneficial to include extra information alongside the events themselves that are useful in writing more abstract story sifting patterns. Rather than specifying only a single string to identify a simulation event's type, for instance, we have found that it can make authoring sifting patterns much easier if you also attach a variable-length list of string tags to each event. For example, an event representing asking someone out on a date and being turned down can be tagged with `romantic` and `failure`. This allows different sifting patterns (for example, some that are looking for looking for 'any romantic event', and some that are looking for 'any failure') to consider the same event for inclusion in matches. This *event polymorphism* increases the potential for narratively interesting emergent behavior to be captured by sifting.

When authoring both a simulation and the sifter meant to operate over that simulation in parallel, it is important not to create *only* the simulation actions that lead to satisfaction of your existing sifting patterns—this misses the point of IEN (increased novelty and emergence) and falls back into what Louchart and Aylett call 'plot-based authoring' [20]. Consequently, it may be advantageous to follow an iterative three-step process: first, author a number of simulation actions without considering the sifting patterns that they might be matched by; second, test the simulation to see what surprising new emergent microstories appear; and third, author sifting patterns to capture these new microstories. Alternating between simulation-focused authoring and sifting-focused authoring creates mental distance between the action sequences that you expect to occur and the action sequences that you are attempting to recognize, allowing emergent behavior to appear independent of attempts to recognize that behavior.

3.3 Designing Simulations That Incorporate Sifting

Beyond authoring a simulation and its sifter in parallel, it is also possible to incorporate sifting directly into the simulation—for instance, by enabling certain character actions within the simulation if and only if certain sifting patterns have been matched. Felt and Kismet [37] both play double duty as sifters and simulation engines by allowing incorporation of sifting patterns into the preconditions of simulation actions.

The co-creative IEN writing game *Why Are We Like This?* [12, 13], which uses Felt as its underlying simulation engine, employs this feature to implement character subjectivity. In addition to taking simulation actions that update the state of the outside world, individual simulated characters can also perform *introspection actions* in which they apply one of their own preferred sifting patterns to a sequence of past events and formulate a narrative *perception* of those events. This mechanism can be used to craft characters with distinct reactive procedural personalities [36] by giving them access to different sifting patterns: for instance, a melancholy character

might be assigned a pool of sifting patterns that allow most social interactions to be interpreted as indicative of hostility, causing the character's interpretations of the world to be biased systematically toward the negative.

Though it has not yet been attempted to the best of the authors' current knowledge, it is also possible to construct a sifting-based *drama manager* [28] that uses sifting to gather information about the current state of the storyworld, then makes targeted interventions at the simulation level to influence the development of emergent storylines based on the sifted information. This would likely represent a relatively light-handed approach to drama management, attempting to gently nudge emergent storylines toward completion (in much the manner of the 'narrative promotion' techniques employed in *The Sims 2* [2, 24]) rather than to impose a single overarching plot structure on the entirety of a storyworld's history.

4 Toward Sifting Heuristics

The sifting patterns that are used in existing story sifters tend to be fairly low-level, concrete specifications of emergent story patterns that make for good narrative material. Patterns at this level, however, do not necessarily capture more generic notions of what makes for a good story, for instance, those that have been set out in cognitive narratology research. This raises the question of how a more generic sense of narrativity could be encoded into the machine, such that sifters can leverage this information to better understand the player-perceived story—for instance, by using abstract narrativity to gauge which of many viable sifting pattern matches are most likely to be important to the player-perceived narrative. In the story sifting literature, encodings of abstract narrativity are called *sifting heuristics* [29, p. 237].

Sifting heuristics may attempt to operationalize constructs from cognitive narratology, including story *interestingness* as defined by Schank [35] and *event salience* [10] (a proxy for story *memorability*) as operationalized in Indexter [4]. An operationalization of *surprise*—which is often treated as a key component of interestingness, and which may be detectable via statistical approaches such as anomaly detection—could also prove useful in sifting heuristics. Since surprise tends to trade off against narrative coherence, striking an appropriate balance between these dimensions is likely to be a central challenge in pursuing this approach.

Sifting heuristics might also be learned from data on how users interact with existing interactive story sifters, for instance, the *Bad News* 'wizard console' or the Legends Viewer interface for exploring *Dwarf Fortress* worlds. Samuel et al. have recently conducted an analysis of interaction trace data with the *Bad News* wizard console [34], revealing that certain sets of wizard console commands are often executed together. Recurring patterns of interaction with these lower-level sifting interfaces could potentially be abstracted into high-level sifting heuristics, since a human user's sense of what information is needed to identify a compelling narrative throughline for a whole *Bad News* play session (for instance) could be expected to serve as a good proxy for the information that a computational system would need to make similar determinations.

5 Conclusion

Story sifting presents a potential solution to one of the key difficulties of interactive emergent narrative: that of mitigating overwhelm and perceived narrative structurelessness while preserving responsiveness and the potential for surprising but compelling emergent narrative developments. However, sifting also introduces new authoring difficulties, particularly around the authoring of story sifting patterns; the construction of simulations that are amenable to sifting; and the definition of highly general sifting heuristics. Several technical and design problems remain to be solved if sifting is to become a more widely deployed solution to the difficulties of IEN.

References

1. Adams T (2019) Emergent narrative in Dwarf Fortress. In: Procedural storytelling in game design. AK Peters/CRC Press, pp 149–158
2. Brown M (2006) The power of projection and mass hallucination: practical AI in The Sims 2 and beyond. Invited talk at AIIDE 2006
3. Cardona-Rivera R, Young R (2015) Symbolic plan recognition in interactive narrative environments. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 11
4. Cardona-Rivera RE, Cassell KB, Ware SG, Young RM (2012) Indexter: a computational model of the event-indexing situation model for characterizing narratives. In: Proceedings of the 3rd workshop on computational models of narrative, pp 34–43
5. Chen S, Nelson M, Mateas M (2009) Evaluating the authorial leverage of drama management. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 4
6. Eladhari MP (2018) Re-tellings: the fourth layer of narrative as an instrument for critique. In: International conference on interactive digital storytelling. Springer, pp 65–78
7. Elson DK (2012) Detecting story analogies from annotations of time, action and agency. In: Proceedings of the LREC 2012 workshop on computational models of narrative, Istanbul, Turkey, pp 91–99
8. Garbe J (2018) Simulation of history and recursive narrative scaffolding. http://project.jacobgarbe.com/simulation-of-history-and-recursive-narrative-scaffolding
9. Johnson-Bey S, Mateas M (2021) Centrifuge: a visual tool for authoring sifting patterns for character-based simulationist story worlds. In: Proceedings of the AIIDE workshop on programming languages and interactive entertainment (PLIE)
10. Kives C, Ware S, Baker L (2015) Evaluating the pairwise event salience hypothesis in Indexter. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 11, pp 30–36
11. Kreminski M, Dickinson M, Mateas M (2021) Winnow: a domain-specific language for incremental story sifting. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 17, pp 156–163
12. Kreminski M, Dickinson M, Mateas M, Wardrip-Fruin N (2020) Why are we like this?: exploring writing mechanics for an AI-augmented storytelling game. In: Proceedings of the electronic literature organization conference
13. Kreminski M, Dickinson M, Mateas M, Wardrip-Fruin N (2020) Why are we like this?: the AI architecture of a co-creative storytelling game. In: Proceedings of the fifteenth international conference on the foundations of digital games (2020)

14. Kreminski M, Dickinson M, Wardrip-Fruin N (2019) Felt: a simple story sifter. In: International conference on interactive digital storytelling. Springer, pp 267–281
15. Kreminski M, Mateas M (2021) A coauthorship-centric history of interactive emergent narrative. In: International conference on interactive digital storytelling. Springer, pp 222–235
16. Kreminski M, Mateas M (2021) Toward narrative instruments. In: International conference on interactive digital storytelling. Springer, pp 499–508
17. Kreminski M, Samuel B, Melcer E, Wardrip-Fruin N (2019) Evaluating AI-based games through retellings. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 15, pp 45–51
18. Kreminski M, Wardrip-Fruin N, Mateas M (2020) Toward example-driven program synthesis of story sifting patterns. In: Joint proceedings of the AIIDE 2020 workshops
19. Louchart S, Aylett R (2004) The emergent narrative theoretical investigation. In: Narrative and interactive learning environments conference, pp 21–28
20. Louchart S, Swartjes I, Kriegel M, Aylett R (2008) Purposeful authoring for emergent narrative. In: Joint international conference on interactive digital storytelling. Springer, pp 273–284
21. Louchart S, Truesdale J, Suttie N, Aylett R (2015) Emergent narrative, past, present and future of an interactive storytelling approach. In: Interactive digital narrative: history, theory and practice. Routledge, pp 185–199
22. Mateas M, Stern A (2000) Towards integrating plot and character for interactive drama. In: Working notes of the social intelligent agents: the human in the loop symposium. AAAI, pp 113–118
23. Muggleton S, De Raedt L (1994) Inductive logic programming: theory and methods. J Logic Program 19:629–679
24. Nelson MJ (2006) Emergent narrative in the sims 2 (2006). https://www.kmjn.org/notes/sims2_ai.html. Accessed 20 Aug 2021
25. Osborn J, Samuel B, Mateas M, Wardrip-Fruin N (2015) Playspecs: regular expressions for game play traces. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 11
26. Porteous J (2016) Planning technologies for interactive storytelling. In: Handbook of digital games and entertainment technologies. Springer
27. Riedl MO, Bulitko V (2013) Interactive narrative: an intelligent systems approach. AI Mag 34(1)
28. Roberts DL, Isbell CL (2008) A survey and qualitative analysis of recent advances in drama management. Int Trans Syst Sci Appl, Spec Issue Agent Based Syst Hum Learn 4(2):61–75
29. Ryan J (2018) Curating simulated storyworlds. PhD thesis, University of California, Santa Cruz
30. Ryan J, Mateas M (2019) Simulating character knowledge phenomena in talk of the town. In: Game AI Pro 360. CRC Press, pp 135–150
31. Ryan JO, Mateas M, Wardrip-Fruin N (2015) Open design challenges for interactive emergent narrative. In: International conference on interactive digital storytelling. Springer, pp 14–26
32. Ryan ML (1992) The modes of narrativity and their visual metaphors. Style 368–387
33. Samuel B, Ryan J, Summerville AJ, Mateas M, Wardrip-Fruin N (2016) Bad news: an experiment in computationally assisted performance. In: International conference on interactive digital storytelling. Springer, pp 108–120
34. Samuel B, Summerville A, Ryan J, England L (2021) A quantified analysis of Bad News for story sifting interfaces. In: International conference on interactive digital storytelling. Springer, pp 142–156
35. Schank RC (1979) Interestingness: controlling inferences. Artif Intell 12(3):273–297
36. Short T (2017) Designing stronger AI personalities. In: Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment, vol 13, pp 111–117
37. Summerville A, Samuel B (2020) Kismet: a small social simulation language. In: Joint workshops of the international conference on computational creativity
38. Walsh, R.: Emergent narrative in interactive media. Narrative 19(1):72–85 (2011).

39. Young RM, Ware SG, Cassell KB, Robertson J (2013) Plans and planning in narrative generation: a review of plan-based approaches to the generation of story, discourse and interactivity in narratives. Sprache Und Datenverarb, Spec Issue Form Comput Model Narrat 37(1–2):41–64