Winnow: A Domain-Specific Language for Incremental Story Sifting

Max Kreminski, Melanie Dickinson, Michael Mateas
University of California, Santa Cruz
{mkremins, mldickin, mmateas}@ucsc.edu

Abstract

Story sifters attempt to automatically or semi-automatically extract nuggets of compelling narrative content from vast chronicles of game or simulation events. Though sifting has successfully been used to enable novel computational narrative play experiences, its utility is limited by the fundamentally *retrospective* nature of existing sifters, which can only recognize storyful event sequences once they have fully played out. To address this limitation, we introduce Winnow: a domain-specific language for specifying story sifting patterns that can be executed *incrementally* to detect potentially storyful event sequences while they are still playing out. We evaluate Winnow by applying it to several specific use cases and show that it is well-suited to the implementation of prospective as well as retrospective narrative intelligence.

Introduction

Story sifters (Ryan, Mateas, and Wardrip-Fruin 2015; Ryan 2018) are computational systems that attempt to automatically or semi-automatically extract nuggets of compelling narrative content from vast chronicles of game or simulation events. Sifters enable a *curationist* approach to emergent narrative (Louchart et al. 2015), in which the generation of narrative events is decoupled from the arrangement and presentation of these events within a particular narrative frame. Because story sifters can detect emergent microstories regardless of how they emerge, sifting-based narrativization of game events can be carried out without direct access to or modification of the underlying event-generating process. Sifting-based approaches are thus especially well-suited to the implementation of narrative intelligence atop existing game or simulation engines, even those that were not initially designed for it.

However, existing approaches to story sifting have several key weaknesses. First and foremost, most prior work on story sifting has assumed that sifting is *retrospective*: microstories can only be recognized once they have run to completion, and not while they are still in the process of playing out. Though a purely retrospective approach to sifting enables some forms of play experiences well (Samuel et al. 2016), it limits the capacity of systems based on story sifting to reason about and intelligently foreshadow future narrative possibilities. Anticipation of future narrative outcomes is essential to how humans engage with stories (Liveley 2017); in narrative generally, readers who anticipate future outcomes may come to desire or dread these outcomes, and in narrative games, players may act to increase or decrease the likelihood that anticipated outcomes will occur. If a sifting-based system is only able to match events against narrative frames retrospectively, its capacity to play into anticipation by procedurally foreshadowing possible outcomes will be limited, as will its ability to make sense of player actions that were performed with anticipated outcomes in mind.

Additionally, despite attempts to make story sifting patterns easier to write, the existing state-of-the-art language for specifying sifting patterns—Felt (Kreminski, Dickinson, and Wardrip-Fruin 2019)—can still be frustratingly low-level. Felt's closeness to raw Datalog makes it difficult for humans to write and maintain complex Felt sifting patterns, because the high-level structure of these patterns can be hard to see at a glance. Moreover, Felt's awkward `not-join` syntax for expressing complex negative constraints, which are used to rule out stories featuring certain events, has been raised as a significant pain point.

To address these issues, we present Winnow: a domain-specific language for story sifting that moves beyond the limitations of existing sifters by providing affordances for incremental sifting. Winnow introduces clear acceptor-based semantics around the maintenance of *partial* sifting pattern matches, which can be narrativized or exposed to a human user prior to their completion. Additionally, where incremental sifting is not required, Winnow sifting patterns can be compiled directly to Felt sifting patterns for Felt-equivalent performance on retrospective sifting tasks. This enables the use of Winnow as a more human-friendly syntax for the specification of Felt sifting patterns.

Related Work

Story sifting as an approach was first described by Ryan, Mateas, and Wardrip-Fruin (2015) under the label "story recognition". Ryan's dissertation (Ryan 2018) introduced the "story sifting" term, provided an expanded characterization of sifting as an approach, and presented a story sifting system—Sheldon—that made use of sifting patterns specified as chunks of procedural Python code to identify microstory structures within the output of a simulation of a small

American town.

Ideally, a story sifter should be able to recognize a diverse array of emergent microstory structures. To achieve this goal, sifters employ large numbers of human-authored *story sifting patterns*, each of which specifies a particular kind of microstory that the sifter can recognize. To ease the authoring of large numbers of sifting patterns, recent work in story sifting has introduced Felt (Kreminski, Dickinson, and Wardrip-Fruin 2019)—a declarative domain-specific query languages in which sets of interrelated events can be concisely described—and Synthesifter (Kreminski, Wardrip-Fruin, and Mateas 2020), an authoring support tool that leverages inductive logic programming to synthesize sifting patterns from user-provided example event sequences.

Several existing systems not originally billed as story sifters bear some resemblance to sifters, and some have even been used in a prospective way. Playspecs (Osborn et al. 2015) are regular expressions that are used to match sequences of game states, though they are less expressive than Sheldon or Felt sifting patterns in that they cannot be parametrized by variables. Plan recognition techniques have been used in an interactive narrative context to anticipate future events (especially player actions) on the basis of past ones (Cardona-Rivera and Young 2015); though plan recognition tends to focus specifically on determining the goals of a particular storyworld agent, while story sifting has a much wider range of associated aims, this approach nevertheless resembles a prospective application of story sifting. *The Sims 2* matches past game events against a library of "story trees" and adjusts the probabilities of future game events to increase the likelihood of events that would advance partially matched story trees (Brown 2006; Nelson 2006); this may be the closest existing approach to ours, but details on this system are hard to come by.

Among purely retrospective systems that resemble sifters, *Caves of Qud* generates biographies for its historical figures by allowing them to perform actions randomly, then running sifting patterns over these action sequences to identify plausible motivations (Grinblat and Bucklew 2017). Several simulation-driven narrative games, including *Prom Week* (McCoy et al. 2013) and those based on the Versu engine (Evans and Short 2013), allow character actions to be predicated on whether a parametrized sequence of past events has unfolded in the game world. And the same is broadly true of systems that attempt to generate stories via planning; these systems have been surveyed elsewhere (Young et al. 2013; Porteous 2016) but are beyond the scope of this paper.

Motivation

Broadly speaking, prior attempts at story sifting share a major weakness when applied to the challenge of *prospective* sifting in the context of a still-running simulation. Because sifting patterns are conventionally written as descriptions of *complete* microstories, existing sifters have no way to anticipate (given a complete sifting pattern) whether that pattern is likely to be fulfilled in the future or not.

Given a library of complete sifting patterns, we could perhaps enable prospective sifting by creating several partial variants of every complete pattern, each of which matches a subset of events leading up to the complete pattern's realization. However, if these partial variants are hand-authored, then substantially more authoring effort (both to create and maintain partial variants) is required per pattern. The temporally unstructured nature of Sheldon and Felt sifting patterns makes it difficult to generate partial pattern variants automatically, because the precise details of when certain constraints must hold are not necessarily clear from a complete retrospective pattern: for instance, if a sifting pattern states that the perpetrator of a particular event must have the "selfish" trait, does that character need to retain the "selfish" trait throughout the entire event sequence, or is the character's personality allowed to change before or after certain pivotal events occur? And regardless of how partial pattern variants are created, all of these variants must be re-run repeatedly to detect new partial matches every time a new event occurs—necessitating the computationally expensive repeat evaluation of numerous pattern constraints.

Winnow represents our solution to these difficulties. By imposing a temporally divisible structure on sifting patterns, Winnow requires its users to clarify when exactly in a pattern's evaluation each constraint must hold. As a result, whenever a new event occurs, Winnow only needs to check whether the event *initiates* any new partial pattern matches (by matching the first temporal stage of a complete sifting pattern), *advances* any already-tracked partial pattern matches (by matching the *next* temporal stage of the relevant pattern), or *invalidates* any of these partial matches (by matching a negative constraint in the relevant pattern). This dramatically reduces the amount of computational work that needs to be performed per event, since only a small subset of each pattern's constraints must be evaluated when a new event occurs. Additionally, it fully relieves users from the authoring burden of creating and maintaining partial variants of complete sifting patterns by hand.

Winnow: An Incremental Sifting DSL

Winnow is an open source[1] domain-specific declarative query language for incremental story sifting. It is used to write sifting patterns. Each Winnow sifting pattern describes a sequence of interrelated events that constitute a potentially interesting microstory and enables the computer to detect instances of this microstory as they emerge.

Here is an example Winnow sifting pattern, modeled after the "violation of hospitality" Felt sifting pattern presented by Kreminski, Wardrip-Fruin, and Mateas (2020):

```
(pattern breakHospitality
  (event ?e1 where
    eventType: enterTown,
    actor: ?guest)
  (event ?e2 where
    eventType: showHospitality,
    actor: ?host,
    target: ?guest,
    ?host.value: communalism)
  (event ?e3 where
```

[1] https://github.com/mkremins/winnow

```
    tag: harm,
    actor: ?host,
    target: ?guest)
  (unless-event between ?e1 ?e3 where
    eventType: leaveTown,
    actor: ?guest))
```

This pattern attempts to match a sequence of events in which a `?guest` character enters a town; is shown hospitality by a `?host` character, who values communalism; but then is harmed in some way by the `?host` before the `?guest` has a chance to leave town.

As in Felt, identifiers prefixed by the `?` character represent *logic variables* that will be bound as the pattern is matched, and whose values must be consistent across the match as a whole. A single complete match for the `breakHospitality` pattern would include bindings for three event variables (`?e1`, `?e2` and `?e3`) and for the characters `?guest` and `?host`.

The sequence of `event` clauses present in a sifting pattern's body constitutes the pattern's *kernel*. A pattern seeks to match an ordered sequence of events corresponding to its kernel, disregarding events that take place between kernel events unless they match one of the pattern's `unless-event` clauses (which can be used to invalidate pattern matches if an event with certain characteristics intervenes between a specified pair of kernel events).

Like Felt, Winnow is implemented in JavaScript and uses the DataScript library[2] as its backend for data storage and Datalog query execution. In the DataScript dialect of Datalog, facts are represented as RDF-like triples of the form `[entity attribute value]`; each triple can be read as an assertion that the `entity` on the left (usually a numeric ID) has an `attribute` whose name is given in the middle and whose `value` is given on the right. All Winnow sifting patterns ultimately operate over facts of this form. Many simulation engines store events and other data as graphs of related entities, with each entity structured as a JSON-like set of key/value pairs; these JSON-like data formats can often be straightforwardly translated into sets of DataScript facts, so we use these representations interchangeably in this paper.

Incremental Execution

Winnow's incremental execution mode revolves around the maintenance of a pool of *partial matches*. Each Winnow sifting pattern is compiled to an acceptor that, given a partial match (i.e., a set of logic variable bindings for the first N events of this sifting pattern) and a new event to consider, performs one of several possible actions:

- **Dies** if this event matches an active `unless-event` constraint on this event sequence.
- **Forks** and **accepts** if this event matches the specification of kernel event $N + 1$ in the sifting pattern. The "parent" partial match is left unchanged (in case an alternative means of advancing this partial match is later discovered), while the "child" match has the new event (and associated logic variable bindings) pushed onto it.
- **Passes** (i.e., remains unchanged) otherwise.

When a new event occurs, it is added to the database and checked against all the active partial matches. Where applicable, these partial matches are then updated based on the new event, and dead and completed partial matches are removed from the pool.

Figure 1 shows an example of incremental sifting pattern execution featuring the `breakHospitality` pattern defined previously. This example demonstrates all the core features of Winnow's incremental execution mode:

- Events that advance partial matches (1, 3, 4, and 5) are accepted onto forks of the matches they advance.
- Irrelevant events (2) do not change the pool of partial matches in any way.
- Events that match active `unless-event` constraints on partial matches (6) cause those partial matches to be marked as dead and removed.
- When all of a partial match's logic variables are bound (`breakHospitality_134`), the match is marked as complete and removed from the pool.
- Partial matches (e.g., `breakHospitality_1`) remain in the pool after they are advanced once (3) so that they can later be advanced again (5) in a different way.
- Events that might have advanced a partial match (7) are ignored if the partial match that they might have advanced was previously removed from the pool.

To determine whether an event matches an `event` or `unless-event` clause in the context of a particular partial match, Winnow translates the clause to a Datalog query and runs the query against the database. For instance, in Figure 1, to determine whether event 3 matches the `?e2` event clause in the context of partial match `breakHospitality_1`, Winnow executes the following query:

```
[3 "eventType" "showHospitality"]
[3 "actor" ?host] [3 "target" "Yann"]
[?host "value" "communalism"]
```

The results of this query are then used to establish bindings for the `?e2` and `?host` logic variables in the new partial match `breakHospitality_13`.

When maintaining a large pool of partial matches, Winnow runs many of these small queries per event—at least one per partial match, and more than one for each partial match against a sifting pattern with active `unless-event` constraints. However, the results of these queries are often very fast to compute, because the values of most involved logic variables (including the ID of the event being tested and the values of any previously-bound logic variables stored in the partial match) are already known. Therefore, only a small number of facts need to be checked to establish possible bindings for the variables that remain unbound.

Use Cases

Autonomous Incremental Sifting

Marie-Laure Ryan's characterization of how baseball commentators narrativize gameplay in real time (Ryan 1993)

[2] https://github.com/tonsky/datascript

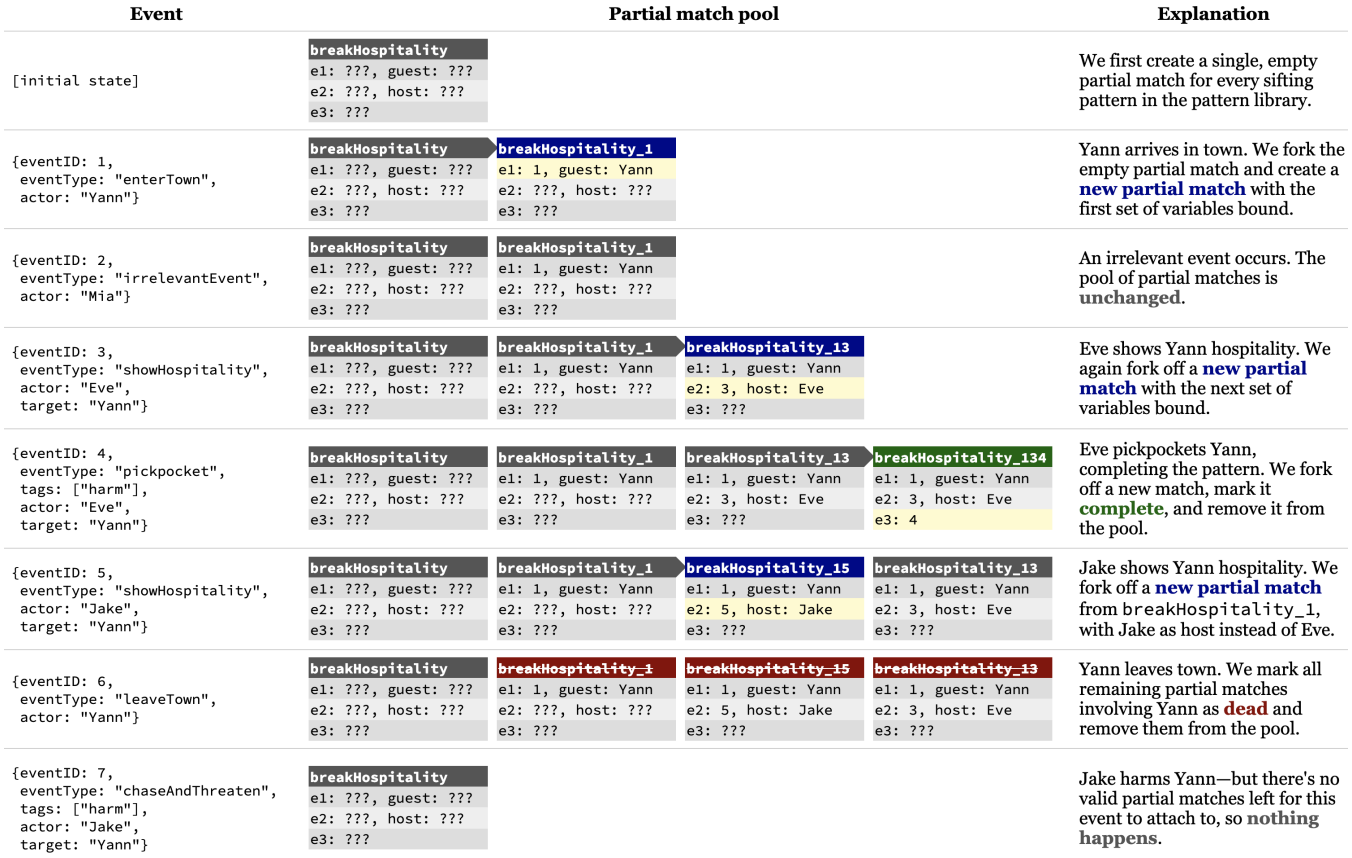

Figure 1: A visualization of Winnow incrementally executing the `breakHospitality` sifting pattern over a sequence of events in which a character **Yann** enters town and is first shown hospitality by, then harmed by, two other characters: **Eve** and **Jake**. As the structured events on the left are added to the database of storyworld state one by one, the pool of active partial pattern matches evolves as shown in the middle and explained on the right.

provides a strong motivating example for incremental sifting. In Ryan's analysis, commenting on a live baseball game involves a combination of looking back at what has already happened and looking forward at what might happen in the future, then weaving a narrative that seems coherent at the present moment while speculating about likely future outcomes to create suspense. A system that seeks to fully reproduce this kind of live gameplay commentary must therefore have some capability to speculate about the future, in addition to the capability to retrospectively interpret the past.

Consider Ryan's analysis of the FATAL ERROR theme in the baseball commentary she has selected for study. In a key moment, an outfielder forgets to flip down his sunglasses; as a result, he fails to track and catch a fly ball, allowing a runner from the opposing team to score. The broadcasters immediately narrativize this blunder as a FATAL ERROR by projecting their narration forward to a possible future in which the game is lost as a result. In Ryan's words:

> The emplotment of the game combines a retrospective interpretation with the prospective evocation of a possible outcome. A whole game is projected on the basis of the play just completed—a game in which the score stands as it is, and the future adds nothing to the story.

A system that uses purely retrospective story sifting to make sense of game events would not be able to generate this commentary live. A sifting pattern that identifies game losses resulting from blunders could be used to narrativize the blunder as a FATAL ERROR once the game has been completed; and a sifting pattern that identifies blunders would be able to recognize this event as a blunder immediately; but to link the blunder to the *possibility* of game loss before the game has actually concluded requires prospective in addition to retrospective narrative intelligence.

The following Winnow sifting pattern expresses a variant of Ryan's FATAL ERROR theme—modified slightly to work with event data adapted from the *Blaseball* API[3] via

[3] *Blaseball* is a surrealist baseball simulation idlegame that mixes normal baseball game events with strange occurrences (e.g., players being incinerated by rogue umpires); we use it here as a convenient source of JSON-formatted simulated baseball gameplay data. Due to details of how *Blaseball* simulates baseball gameplay, we have switched the focus of this pattern to be on a batter rather than an outfielder who commits a blunder: *Blaseball* includes a humorous "strike out thinking" blunder that batters can perform, but doesn't simulate outfielders at sufficiently high fidelity to attribute a blunder to one outfielder in particular.

the fan-created chronicler tool Datablase[4], which allows for the easy retrieval of historical *Blaseball* simulation output.

```
(pattern fatalError
  (event ?gainLead where
    (leadChange ?gainLead),
    (teamHasLead ?gainLead ?team))
  (event ?blunder where
    event_text: ?text,
    (includes? ?text "strikes out thinking"),
    batter_id: ?blunderer,
    batter_team_id: ?team)
  (event ?loseLead where
    (leadChange ?loseLead),
    (not (teamHasLead ?loseLead ?team)))
  (event ?loseGame where
    event_type: GAME_OVER)
  (unless-event ?e between ?gainLead ?blunder
    where (leadChange ?e))
  (unless-event ?e between ?loseLead ?loseGame
    where (leadChange ?e)))
```

This pattern matches a sequence of events in which a team gains the lead over their opponents; commits a serious blunder; subsequently loses the lead; and then loses the game without ever regaining the lead. It employs several advanced Winnow features: two custom Datalog inference rules, `(leadChange ?event)` and `(teamHasLead ?event ?team)`, are used to identify events in which a particular team gained the lead over their opponents; the DataScript built-in function `includes?` is used to determine whether a string value contains a specific substring; and the `not` keyword is used to negate a constraint.

When this pattern is executed incrementally while the game is still going on, a partial match that has advanced as far as the `?loseLead` event can safely be narrated as a *likely* fatal error. Instead of waiting for the entire event sequence to be completed before we can comment on the FATAL ERROR theme, we can comment speculatively on the likelihood that the `?blunderer`'s mistake has cost their team the win.

Additionally, if we note in the event database that we have chosen to speculatively comment on this event as a fatal error, we can make use of this in later commentary to refer back to our own past commentary as mistaken. Suppose the team that committed the blunder ends up winning the game despite our prospective evocation of the FATAL ERROR theme. In this case, the final story of the game as produced by our incremental sifting-based commentary generator can dramatize the mistaken commentary as part of a larger ERROR AND REDEMPTION theme by retrospectively invoking the moment at which it appeared that the game would be lost due to the blunder. By computationalizing the "anticipation of retrospection" that characterizes prospective narrative intelligence (Brooks 1984), we can produce more human-like commentary that more effectively draws out a convincing story from gameplay.

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

Interactive Incremental Sifting

Several play experiences that center on narrative coauthorship between the player and an AI system have made use of story sifting to help the player comprehend the contents of a rich simulated storyworld. These include Writing Buddy (Samuel, Mateas, and Wardrip-Fruin 2016), *Cozy Mystery Construction Kit* (Kreminski et al. 2019), and *Why Are We Like This?* (Kreminski et al. 2020a,b). In these games, the ability to expose partial sifting pattern matches to the player introduces new potential affordances for play.

Consider a sifting pattern like the following, adapted from the `arson-revenge` Sheldon sifting pattern presented by Ryan (2018):

```
(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)))
```

In an interactive sifting context, when a pattern like this is available and a partial match with bindings for the first two events is present, the system can make use of this partial information (that a revenge scheme has been undertaken but not yet completed) in many ways. Beyond simply informing the player that this microstory is *possible*, it can accept suggestions from the player on whether the scheme should actually be carried out or not; suggest character actions that advance the scheme in various ways (e.g., having the `?arsonist` character purchase a can of gasoline); or even suggest alternative ways that the revenge scheme could be carried out, for instance if multiple revenge-oriented patterns that all begin with a similar set of events are simultaneously present.

Additionally, a game of this nature could allow the player to choose for themselves a sifting pattern that they would like to see take place. Once chosen, the player could track the progress of this sifting pattern as new events occur; decide whether or not to accept a particular event as advancing the sifting pattern; and receive early warning from the computer when an event has a chance of disrupting the story that the player is trying to create. Such an interface could enable the computer and player to explicitly collaborate on partially realized stories in exciting new ways.

Retrospective Sifting

It is also possible to use Winnow as an alternative, more verbose but more human-friendly syntax for the specification of retrospective sifting patterns. For instance, Winnow can compile the previously defined `breakHospitality` sifting pattern to a roughly equivalent Felt pattern:

```
(eventSequence ?e1 ?e2 ?e3)
[?e1 eventType enterTown] [?e1 actor ?guest]
[?e2 eventType showHospitality]
```

```
[?e2 actor ?host] [?e2 target ?guest]
[?host value communalism]
[?e3 tag harm]
[?e3 actor ?host] [?e3 target ?guest]
(not-join [?e1 ?guest ?e3]
  (eventSequence ?e1 ?eMid ?e3)
  [?eMid eventType leaveTown]
  [?eMid actor ?guest])
```

One minor semantic difference between this compiled Felt pattern and the incrementally executed Winnow pattern lies in how the constraint `[?host value communalism]` (the Felt equivalent of Winnow's `?host.value: communalism`) is applied. Under retrospective execution of a Felt sifting pattern, this constraint is checked at the end of the event sequence, so the validity of the match hinges on whether the host character still holds communalism as a value after the whole sequence has played out. Under incremental execution, however, this constraint is associated specifically with the `?e2` execution stage, so we only check whether the host values communalism at the time of event `?e2`. This allows for incrementally executed patterns to make some subtle distinctions about when constraints hold that are not possible when executing sifting patterns in a purely retrospective mode.

Compilation allows for Winnow sifting patterns to be used as preconditions in Felt action definitions. It also ensures that Winnow's performance in a purely retrospective sifting context is never worse than Felt's. And even "timeless" Felt or Sheldon sifting patterns that are not formulated in terms of event sequences (e.g., a sifting pattern that finds instances of unrequited love between characters) can be expressed in Winnow as single-event patterns. Consequently, Winnow can often be used as a drop-in replacement for the sifting component of Felt.

Performance

Winnow is written in browser JavaScript, in a coding style that optimizes for clarity over performance. Nevertheless, it is fast enough to be useful for at least some real-world simulation-driven games.

To establish a performance baseline, we created an incremental sifting benchmark task involving a small simulated storyworld with 30 event types and 5 characters. We conducted several distinct runs of the benchmark; on each run, the partial match pool was initialized with a variable number of partial matches against the `breakHospitality` pattern, ranging from 10 up to 1000. One hundred random events (with an event type and tags randomly taken from the 30 available event types, and an actor and target character chosen randomly from the five available characters) were created and added to the database one by one, and the time it took Winnow to update the partial match pool on each event addition was recorded.

The benchmark was run in Firefox 87.0, on a 2019 MacBook Pro with a 2.6 GHz 6-Core Intel Core i7 processor and 16GB of RAM. Full benchmark results are available in Table 1. Notably, on the most difficult version of the benchmark task (in which 1000 partial pattern matches had to be checked per event), Winnow took on average 912ms per event to update the partial match pool. This is within the one-second response window suggested by usability experts as sufficient for maintaining the user's flow of thought in an interactive context (Nielsen 1993, Chapter 5); therefore, Winnow's performance is likely sufficient for the provision of near-immediate feedback on player-initiated game events. Additionally, in many games, events of potential narrative significance occur much less frequently than once per second, so it is likely that Winnow will be able to keep up with a wide variety of gameplay types. (Especially frequent events, such as movement events in action games, are often of little narrative significance and unlikely to even be logged as events in a narrative-focused chronicle of gameplay.)

Pool size	Min time	Max time	Avg time
10	9ms	40ms	13ms
50	41ms	110ms	50ms
100	78ms	227ms	93ms
500	398ms	577ms	460ms
1000	443ms	1097ms	912ms

Table 1: Benchmark results for various partial match pool sizes. Minimum, maximum, and average time taken to update the partial match pool per event is given for each iteration of the task.

Low-hanging fruit for further optimization is abundant. In particular, Winnow's use of the DataScript library for Datalog query execution imposes a string parsing overhead on every query that is run; this overhead could be reduced through tighter integration with a Datalog backend. Additionally, since Winnow may frequently find itself evaluating the same Datalog expression many times when checking an event against a large pool of partial matches, some form of expression-oriented evaluation cache may help to avoid redundant computation.

Discussion

Pool Management Strategies

Winnow makes no attempt to remove partial pattern matches from the pool, except when they are either completed or killed by `unless-event` constraint violations. As a result, additional application-specific pool management strategies may be useful in mitigating unbounded growth of the pool over time. Collectively, these strategies bear some resemblance to the "sifting heuristics" proposed by Ryan (2018): higher-level counterparts to sifting patterns that encode more generic facets of narrative tellabilty.

One easy-to-implement and fairly generic approach to pool management involves the automatic pruning of any partial match that has not accepted any of the last K events for some reasonably large K. The exact threshold to use here is likely dependent on the texture of the simulation with which you are working, and different thresholds may even be appropriate for different sifting patterns within the same application. For instance, a partial match against a "whirlwind romance" pattern can likely be safely pruned after a relatively short period of inactivity, whereas a match against a

pattern that encompasses the whole of a character's lifespan might usefully lie dormant for a much longer period.

Another strategy for mitigating pool growth involves replacing a partial match's default "fork and advance" behavior with a simpler "advance directly" behavior (i.e., a behavior that avoids growing the partial match pool) once the match is advanced past a certain point. For instance, a partial match that already has bindings for all of its non-event variables could be advanced directly without forking off duplicates. Since the non-event variables in some patterns (e.g., the identities of the `?host` and `?guest` characters in the `breakHospitality` pattern) seem much more strongly determinant of the microstory's player-perceived identity than the events themselves, a group of partial matches that are technically unique but vary only in event specifics might be perceived by the player as duplicates—a perception that would be mitigated if this strategy was employed.

Finally, narrower application-specific heuristics could be used to clean up partial matches when certain game events take place without requiring these events to be specified as `unless-event` clauses in every relevant sifting pattern. For instance, depending on the simulation domain, many emerging microstories might be invalidated by the premature death of an involved character. Therefore, instead of writing `unless-event` clauses into almost every pattern to hedge against character death, it might be easier from an authoring standpoint (and more performant from a work-minimizing standpoint) to automatically prune any partial matches involving a character that has just died—perhaps excluding matches against a smaller set of patterns that have been specifically marked as tolerant of character death.

Modeling Causality

As Ryan has argued (Ryan 2018), *causal bookkeeping*—the explicit modeling of causality relationships between simulation events—greatly aids the implementation of a curationist approach to emergent narrative. However, many of the simulation engines that are used in notable emergent narrative games today—for instance, Ryan's own Talk of the Town simulation engine (Ryan and Mateas 2019)—do not perform explicit causal bookkeeping, instead relying on human interactors to infer or invent causality relationships between events. In order to ensure that Winnow is able to reason over the output of a wide range of simulation-driven emergent narrative games, we wanted to avoid imposing a technical requirement that Winnow sifting patterns match only causally connected sequences of events. As a result, Winnow sifting patterns are by default written in a causality-agnostic way.

When working with the output of a simulation engine that performs explicit causal bookkeeping, we expect that causal relationships between events will be available as data within the event entities themselves: for instance, every event entity might contain an explicit pointer to the previous event or events that caused it. Therefore, Winnow sifting patterns can be written to explicitly reason about this causality information when it is present—for instance, to constrain pattern matches so that all of a pattern's kernel events must be causally related—or to ignore causality information when it is either absent or irrelevant to the context in which a particular sifting pattern will be used. We believe that this added flexibility is worth the slight additional authoring burden of having to manually assert causality constraints between events when these constraints are desired.

Conjunction and Disjunction

At the language level, Winnow does not provide any specific support for writing sifting patterns that match the conjunction of two or more other patterns. However, conjunction of patterns can straightforwardly be implemented through modification of the event chronicle. When a match (complete or partial) against a particular sifting pattern is first detected, the fact that this match has occurred can be added to the chronicle as a new event. Higher-level sifting patterns can then be written to look for instances of these match events and advance when the appropriate conjunction of lower-level pattern matches has occurred.

Within sifting patterns, disjunction (the ability for a single event clause to match *either* an event A or an event B, where A and B have different characteristics) can be implemented via Datalog inference rules. An inference rule with multiple disjoint bodies will hold if any one set of body conditions holds true; therefore, an event clause can match disjoint events by checking whether the event satisfies an inference rule that contains a disjunction.

Conclusion

We have introduced Winnow, a domain-specific language for incremental story sifting that improves on previous sifting technologies (particularly Felt) by enabling the implementation of *prospective* as well as retrospective narrative intelligence via sifting. Winnow is capable of expressing and incrementally executing a wide variety of realistic sifting patterns, including equivalents to existing Felt and Sheldon patterns and patterns that operate over the *Blaseball* simulation. Additionally, it is performant enough to run in an interactive context and can be used as a more human-friendly language for purely retrospective sifting as well.

At a high level, Winnow can be viewed as a narrative cognition engine that attempts to help the computer understand partial stories that might have arisen in the mind of a human spectator or user. It is from this perspective that story sifting appears to us as an especially exciting approach to narrative intelligence: if we can computationally model the way that humans make sense of emergent stories, we can build systems that are capable of understanding gameplay narratively, just as players do. We believe that future work on sifting should attempt to further explore the implications of this view.

Acknowledgements

We thank the anonymous reviewers for their thoughtful feedback. We also thank the Seabright Camerata (particularly Cat Manning, Jason Grinblat, and Jacob Garbe) for their participation in the discussions that led to Winnow's creation.

References

Brooks, P. 1984. *Reading for the Plot: Design and Intention in Narrative*. Harvard University Press.

Brown, M. 2006. The power of projection and mass hallucination: Practical AI in The Sims 2 and beyond. Invited talk at AIIDE 2006.

Cardona-Rivera, R.; and Young, R. 2015. Symbolic plan recognition in interactive narrative environments. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11.

Evans, R.; and Short, E. 2013. Versu—a simulationist storytelling system. *IEEE Transactions on Computational Intelligence and AI in Games* 6(2): 113–130.

Grinblat, J.; and Bucklew, C. B. 2017. Subverting historical cause & effect: generation of mythic biographies in Caves of Qud. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Kreminski, M.; Acharya, D.; Junius, N.; Oliver, E.; Compton, K.; Dickinson, M.; Focht, C.; Mason, S.; Mazeika, S.; and Wardrip-Fruin, N. 2019. Cozy Mystery Construction Kit: Prototyping toward an AI-assisted collaborative storytelling mystery game. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*.

Kreminski, M.; Dickinson, M.; Mateas, M.; and Wardrip-Fruin, N. 2020a. Why Are We Like This?: Exploring writing mechanics for an AI-augmented storytelling game. In *Proceedings of the Electronic Literature Organization Conference*.

Kreminski, M.; Dickinson, M.; Mateas, M.; and Wardrip-Fruin, N. 2020b. 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*.

Kreminski, M.; Dickinson, M.; and Wardrip-Fruin, N. 2019. Felt: a simple story sifter. In *International Conference on Interactive Digital Storytelling*, 267–281. Springer.

Kreminski, M.; Wardrip-Fruin, N.; and Mateas, M. 2020. Toward example-driven program synthesis of story sifting patterns. In *Joint Proceedings of the AIIDE 2020 Workshops*.

Liveley, G. 2017. Anticipation and narratology. In Poli, R., ed., *Handbook of Anticipation: Theoretical and Applied Aspects of the Use of Future in Decision Making*. Springer.

Louchart, S.; Truesdale, J.; Suttie, N.; and Aylett, R. 2015. Emergent narrative, past, present and future of an interactive storytelling approach. In *Interactive Digital Narrative: History, Theory and Practice*, 185–199. Routledge.

McCoy, J.; Treanor, M.; Samuel, B.; Reed, A. A.; Wardrip-Fruin, N.; and Mateas, M. 2013. Prom Week: designing past the game/story dilemma. In *Proceedings of the International Conference on the Foundations of Digital Games*.

Nelson, M. J. 2006. Emergent narrative in The Sims 2. https://www.kmjn.org/notes/sims2_ai.html.

Nielsen, J. 1993. *Usability Engineering*. Morgan Kaufmann.

Osborn, J.; Samuel, B.; Mateas, M.; and Wardrip-Fruin, N. 2015. Playspecs: Regular expressions for game play traces. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11.

Porteous, J. 2016. Planning technologies for interactive storytelling. In *Handbook of Digital Games and Entertainment Technologies*. Springer.

Ryan, J. 2018. *Curating Simulated Storyworlds*. Ph.D. thesis, University of California, Santa Cruz.

Ryan, J.; and Mateas, M. 2019. Simulating Character Knowledge Phenomena in Talk of the Town. In *Game AI Pro 360*, 135–150. CRC Press.

Ryan, J. O.; Mateas, M.; and Wardrip-Fruin, N. 2015. Open design challenges for interactive emergent narrative. In *International Conference on Interactive Digital Storytelling*, 14–26. Springer.

Ryan, M.-L. 1993. Narrative in real time: chronicle, mimesis and plot in the baseball broadcast. *Narrative* 1(2): 138–155.

Samuel, B.; Mateas, M.; and Wardrip-Fruin, N. 2016. The design of Writing Buddy: a mixed-initiative approach towards computational story collaboration. In *International Conference on Interactive Digital Storytelling*, 388–396. Springer.

Samuel, B.; Ryan, J.; Summerville, A. J.; Mateas, M.; and Wardrip-Fruin, N. 2016. Bad News: An experiment in computationally assisted performance. In *International Conference on Interactive Digital Storytelling*, 108–120. Springer.

Young, R. M.; Ware, S. G.; Cassell, K. B.; and 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 Datenverarbeitung, Special Issue on Formal and Computational Models of Narrative* 37(1-2): 41–64.