Skald: Minstrel Reconstructed

Brandon Tearse, Peter Mawhorter, Michael Mateas, and Noah Wardrip-Fruin

***Abstract*—Scott Turner's Minstrel is considered a landmark story-generation system, cited as an important system in our field's history for the quality of its output. Other influential systems such as Meehan's Tale-Spin have inspired modern successors, but although a few systems have followed Minstrel's case-based approach, none of them use its "imaginative recall" technique. This paper details Skald, a publicly-released rational reconstruction of Minstrel that enables exploration of Turner's work and discovery of new implications for future research. A key finding is a brittleness only hinted at in Turner's publications: the story library, story templates, and the recall system must be tailored to one another for Turner's original system to function. We show that this can be ameliorated through a number of techniques, however, from adding differential costs to transformations to removing the least-successful author-level actions. Another key finding is that Turner's original "boredom" system limited leverage of the story library. An alternative and its results are presented here. What emerges from this work is a different picture of the original Minstrel than that currently present in the literature, as well as a new system, Skald, that sets the stage for future research to explore Turner's ideas for story generation.**

***Index Terms*—Artificial intelligence, knowledge-based systems, open source software.**

I. Introduction

SCOTT Turner's influential 1993 Minstrel system is a graph-based story generator designed to mimic a human author's creative process [1]. As one of only a few systems that attempt to model human creativity, its unique approach is worthy of study and it is also often cited (e.g., [2]–[4]) along with systems such as Meehan's Tale-Spin [5] and Lebowitz's Universe [6] as a seminal story-generation system. Based on a model of human creativity that centers on imaginative recall, it generates stories by modifying and splicing together fragments from a library of stories in order to fit its own needs. Despite its influence and important ideas, Minstrel, like many other early AI projects, is not accessible today, and no current system has built on its core "imaginative recall" technique. This paper presents Skald, a rational reconstruction of Minstrel, and describes the lessons we have learned while reconstructing a system that is almost two decades old.

Our main motivation to reconstruct Minstrel was based on the quality of the stories that it generated. We wanted to figure out how they were produced and how general this process was: how much of Minstrel's code would have to change to produce stories in a different genre, for instance? Reproduced here is an example story that Turner presented in his dissertation [1], which demonstrates the full capabilities of his system:

The Mistaken Knight

It was the spring of 1089, and a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot had lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.

One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to the woods. Andrea was at the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot loved Andrea. Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick. Andrea loved Frederick. Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot wanted to be near Frederick. Lancelot moved to Frederick. Lancelot was near Frederick. Lancelot fought with Frederick. Frederick was dead.

Andrea wanted to be near Frederick. Andrea moved to Frederick. Andrea was near Frederick. Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings with Frederick. Lancelot wanted to take back that he wanted to kill Frederick. But he could not because Frederick was dead. Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.

MORAL: Done in haste is done forever.

This story has a coherent plot, reasonably dramatic action, and incorporates foreshadowing. The English is a bit stilted, but is not full of errors, and the story has an explicit moral that it bears out. Even in the context of modern computer-generated stories, it stands out as both interesting and coherent. But given Turner's description of Minstrel's internal systems, some questions remain difficult to answer. Exactly how many unique sto-

Manuscript received October 05, 2012; revised June 27, 2013; accepted August 23, 2013. Date of publication November 21, 2013; date of current version June 12, 2014. This work was supported by the National Science Foundation under Grants IIS-1048385 and IIS-1258305.

The authors are with the Department of Computer Science, University of California Santa Cruz, Santa Cruz, CA 95064 USA (e-mail: batman@soe.ucsc.edu; pmawhorter@soe.ucsc.edu; michaelm@soe.ucsc.edu; nwf@soe.ucsc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2013.2292313

ries like this can Minstrel generate? Could Minstrel be used to create stories incrementally allowing a user (or a player) to influence how they turned out? To answer these questions, we decided to undertake a rational reconstruction of Minstrel.

Rational reconstruction is the process of rebuilding a system based on a detailed description of how it works. Rational reconstruction gives a new perspective on acclaimed technical projects, and thus provides an opportunity to understand more about how technical research progresses [7]. Reconstructions are also opportunities to learn more about the original systems than even their authors knew. Places where the description of a system is unclear or contradictory can highlight difficult decisions or potential changes that can alter system performance. Additionally, because this approach does not rely on the source code, reconstruction can reveal choices that the original author was unaware of having made.

Our work reconstructing Minstrel suggests that the original system was focused on demonstrating creativity and that without a carefully tailored story library, the quality of its output decreases drastically. To make the imaginative recall algorithm more robust we have developed alternate implementations of Turner's TRAM selection and boredom systems which work better without a tailored library, and which spread the material available in the story library over a larger number of output stories. Our work also turned up places where the original system was unclear given the public documentation, and idiosyncrasies of the underlying representation schema that limit what kinds of stories can be represented. In both cases, we have implemented something as close as possible to the original system as well as our own extensions. Of course, another result of our work is a storytelling system written in a modern programming language (Scala) that is freely available for others to work with.[1]

Besides describing our new system and some lessons learned from its construction, this paper describes example output from our system and a prototype game experience that uses it in the hope that others might be interested in exploring the possibilities of Minstrel-style story generation.

II. Related Work

Minstrel is a story-generation system, but it is also a computational model of creativity. Both of these communities have created systems relevant to our work on Skald.

A. Story Generation

Computer programs that generate compelling stories are difficult to create, as evidenced by some 40 years of research on the subject. The earliest dedicated story-generation system was Klein *et al.*'s 1971 Automatic Novel Writer [8], [9], building out of his earlier work on natural language processing and linguistics [10]. The most influential early system, however, was Meehan's Tale-Spin [11], which came to be regarded as the foundation of story-generating systems. Tale-Spin used character-level planning to generate stories: each character in Tale-Spin would plan a series of actions to get out of some imposed predicament, and these actions became the story output. Tale-Spin inspired many other systems to use planning as their core assembly algorithm, but most later systems avoided purely character-driven stories.

After Tale-Spin, systems like Dehn's Author [12] and Lebowitz's Universe [6] introduced the concept of author-level reasoning. Dehn's work in particular used a method of memory organization (intended to mimic human memory) that Turner also used in Minstrel. Where Dehn focused on author-driven creation of story scenes, Lebowitz's system revolved around the properties of its characters, although the event creation mechanism functioned using author-level constraints (e.g., "now these two characters must fall in love") rather than through character-level planning as Tale-Spin had.

Building on these early systems, Turner's 1993 Minstrel [1] worked at the author level, but it used case-based reasoning (CBR) to fill in story fragments. CBR works by consulting a library of problem and solution pairs whenever a new problem is encountered. Where story generation is concerned, problems often look like fill-in-the-blank questions (e.g., "A _____ walks into a bar. . ." with possible solutions from the library yielding options: man, child, cow, rope, etc.) but the technique is not constrained to story generation and thus problems and solutions could take on any structured and searchable format. Our reconstruction operates almost identically to the original; full details of the architecture are described in Section III.

Since Minstrel, several other approaches have emerged. Some systems have continued to use planning-based approaches (e.g., [13] and [14]), while others have used agent-based algorithms [15], rule systems [16], or even genetic algorithms [17].

One line of work similar to Minstrel's case-based approach is Zhu and Ontanón's 2010 Riu system [18], which uses a computational analogy algorithm to fill in parts of a story. A few other systems have also used CBR, including Pérez y Pérez's 1999 Mexica [19], Fairclough and Cunningham's 2003 system [20] and Gervás *et al.*'s 2005 ProtoPropp [21]. These systems all derive world knowledge from a library of stories rather than encoding it explicitly, while also being able to use near-matches to assist in recalling solutions to novel problems.

Of these CBR-based systems, Riu's computational analogy algorithm is most similar to Minstrel's imaginative recall approach. Like Minstrel, Riu operates at the level of an author, modifying stories by filling in scenes sequentially. Although computational analogy and case-based approaches have a lot in common (notably both rely on a story library), computational analogy uses a very specific matching and adaptation algorithm that differs from Minstrel's transform–recall–adapt methods (TRAMs). The other three notable CBR-based story generators function differently: Mexica uses a Tale-Spin-like approach, telling a story based on character-level simulation, while both Fairclough and Cunningham's system and Proto-Propp are based on Propp's work on story grammars [22]. Propp was a Russian narratologist who proposed a grammar-based model of Russian folktales. His model is attractive because it is computationally tractable, but it is often applied outside the domain of the folktales that it was intended to describe.

Although these CBR-based story-generation systems are similar to Minstrel, none of them use the imaginative approach that

[1]Skald can be downloaded at: http://minstrel.soe.ucsc.edu/

Turner did. The ability of Turner's TRAM system to find creative matches, along with the capacity for using domain-specific TRAMs to control this matching process, is unique to Minstrel.

B. Creativity

Turner's original thesis claimed that Minstrel was a computer model of human creativity during story construction [1]. Creativity, however, especially computational creativity, is an extremely difficult topic to pin down. Most academics are willing to agree that a suitably complex system is required for creativity to arise, that the output must be suitably unique and have some amount of quality or value in order for it to be creative, and that what exactly creativity is and how it is measured is still a very open question. In our previous work, we have tried to measure how creative Minstrel was using notions of variability and quantity of output [23], but more advanced techniques could be applied.

Margaret Boden's idea of p-creativity is one starting point for a more formal definition of creativity. The output of a system is p-creative if it is unique with regards to the input to and previous output of that system [24]. This idea coupled with the constraint that the output must be consistent with the context in order to qualify as creative (i.e., producing unique gibberish is not creative) is the basis of a number of creativity frameworks (e.g., [24] and [25]). Although Turner's original definition [26] agrees with much of the thinking in the field and we have used variability and quality of output to measure the creativity of the system, other more complex metrics exist and should be tried at some future point (e.g., [27] and [28]).

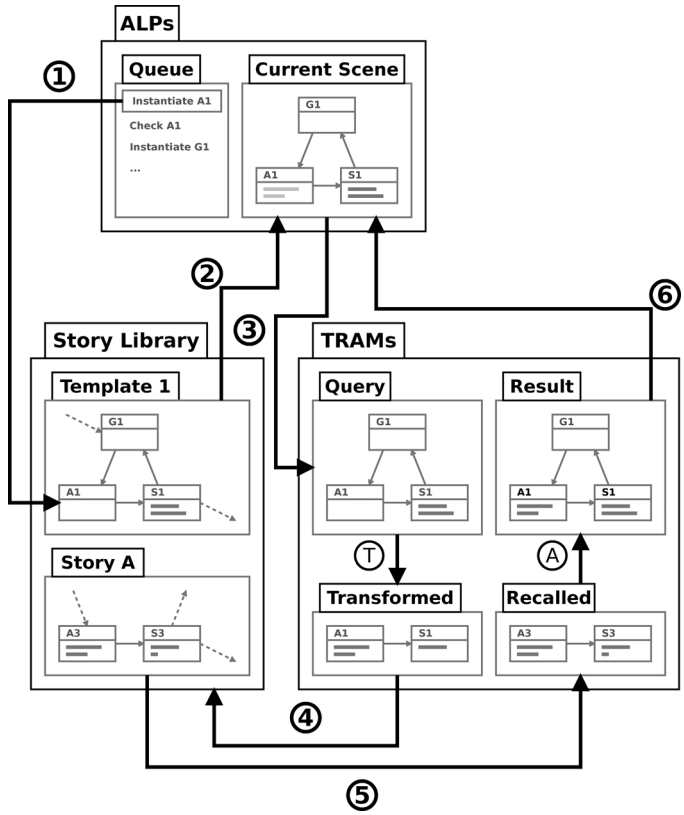

Fig. 1. Interactions between the ALP system, the TRAM system, and the story library during the instantiation of a single frame: 1–2: An ALP extracts a scene from the active story template in the library. 3: The ALP sends that scene to the TRAM system as a query. 4: Transformed queries are sent to the story library. 5: Eventually, a match is sent back to the TRAM system. 6: After adaptation, the TRAM system sends the result back to the ALP system, which will update the template in the library (not shown).

III. Skald

Our new system Skald is a reimplementation of Minstrel. First, we produced a close copy of Minstrel, called simply "Minstrel Remixed," and then we started tuning the system to improve results, calling our newer version "Skald." Previous work has described and measured Skald/Minstrel Remixed [29]–[32], analyzed its potential for interactive narrative [33], and investigated its creative subsystems [23]. This paper provides a thorough overview of the entire system and describes concrete lessons learned during the project. This section provides a brief overview of the core components common to both Skald and Minstrel, and it points out where they differ.

A. Overview

Skald is written in Scala 2.9 (a relative of the Java programming language) and it includes the original components of Minstrel described in Turner's dissertation [1], as well as a few changes (see Section IV for the rationale behind these differences). At a high level, Skald creates a story by simulating the actions of a human author. Some modules are low level simulations of problem solving processes (akin to finding a replacement for absent jelly when one wants a peanut butter and jelly sandwich) while others simulate higher level authorial goals such as working from a given story outline or trying to highlight tragedy or suspense. Skald puts these modules to work on an input story, ranging from empty (generate a story from scratch) to all-but-complete; it decides on an area to work on, attempts to flesh it out, and then moves on to another area. Once a story is completed and all of the various modules have checked it over to ensure that it is finished and makes sense from their perspective, it will store the story in its library for later reuse and then present it to the user.

B. The Minstrel Architecture

Minstrel is a complex architecture including many subsystems and components. These are organized largely into three separate subsystems with a fourth organizational system to help them interact: the story library, TRAMs, author-level plans (ALPs), and the Bard, respectively. The TRAM system refers to a collection of TRAMs and accompanying support mechanisms that are used to do problem solving and generate the fine details of a story. ALPs are used to direct the broader themes of a story as well as being used to enforce a variety of consistency constraints. Together, the planning of the ALP system and the case-based reasoning of the TRAM system allow for complete stories to be pieced together. Finally, the Bard is responsible for managing communication between the components and initiating the story-generation process.

Fig. 1 shows the three primary systems in action. The details of steps 1–6 are explained in Section III-G (after the components have been described individually).

C. The Story Library

1) Functionality: Stories are represented as graphs. Each node is a map of key-value pairs describing a state, goal, action, or belief, and the edges describe how the nodes relate to one another. The key-value mappings are derived from Roger Schank's conceptual dependency [34], so each node type (state, goal, action, and belief) has a fixed set of keys that it uses. Goal nodes, for example, have type, actor, object, and value slots.

If a character plans to heal another character, this is represented as a goal with type "C-Health" (change health) and value "healthy." The actor and object of the goal are filled in with the character who has the goal and the character that they want to heal, respectively. Edges then represent the flow of the story. If the character acts on their goal, there will be a "plans" link from that goal node to an act node, and then perhaps an "intends" link from the act node to a state node, with an "achieves" link from the state back to the goal. In fact, most of the nodes in Minstrel stories are organized into these goal–act–state (or GAS) trios, with "motivates" links from the state node of one trio to the goal node of the next.

Using these graphs as its basis, the story library starts out with preloaded stories as a basis for generation, but it also incorporates any new stories that are generated. This particular system is the simplest of the four core systems since it is little more than a data storage and retrieval system.

2) Changes: Although the functionality of Skald's story library is the same as that of the original Minstrel, Turner's thesis did not include the complete set of stories that he used for his library. As a result, our story library is different, including a few stories that were described in his paper, but mostly made up of stories that we created ourselves. This is one of the most significant changes in terms of its impact on story output, although it does not affect the system architecture. The majority of our story library is a set of stories that we authored aimed at providing a diverse set of raw material for generation centered around a roughly medieval theme. Although this seemed like what Minstrel called for, we have found that a much more focused library may be necessary to support the quality of stories that Turner was able to produce. Section IV includes further discussion of this point.

D. TRAMs

1) Functionality: TRAMs support a modified form of CBR. In Minstrel, many small fragments of stories are created over the course of creating a whole story and this is done by the TRAMs. The TRAM system is called with a search query which is generally a partially defined story fragment such as "someone does something to a dragon which kills it." The system includes a number of TRAMs which can be recursively applied in any order to find a match for a given query.

The first step for each query is to attempt to recall fragments out of the story library which match without any transformation. Failing this, each TRAM has a specific transformation that it applies to the current query before recursing. Effectively, the TRAM system searches through a space of queries (connected via TRAM applications) to find a linear sequence of TRAMs which lead from the original query to a query that directly matches a story fragment in the library. When a result is found, each TRAM in the recursive stack applies its own adaptation code, adapting the result into a form applicable to the original query. The benefit of this method is that it can use precise transformations and adaptations in sequence to effectively find cases that are quite different from the original query. Of course, the TRAMs are not perfect, and the more TRAMs used, the higher the risk of a result which is incoherent with the rest of the story. Thus, the TRAM system relies on a story library which contains examples that are fairly close to the queries it attempts to answer.

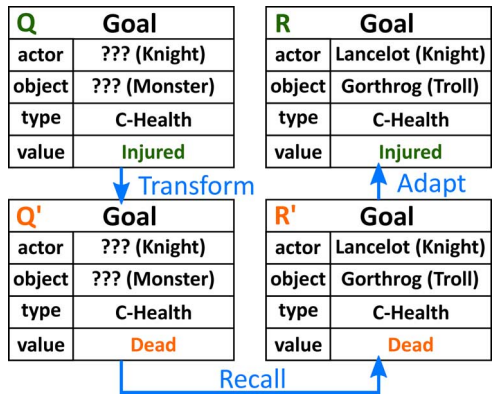

Fig. 2. The TRAM system in action.

In Fig. 2, an example query Q is shown in which a knight has a goal of injuring a monster but no stories are available in the library which can be used. As a result, a TRAM converts the query to one in which the monster is killed (Q') and this new query retrieves a result R' from the case library about Lancelot planning to kill Gorthrog the troll. This result is finally adapted back to a fragment R in which Lancelot plans to injure Gorthrog, satisfying all of the original requirements.

Fig. 3 illustrates a TRAM search tree with three potential TRAMs. The query passed in is the top node in which a knight fights something. The three children of the top node are all possible through different TRAM applications. The left child generalizes the action, the middle child replaces the actor type with a more general actor type, and the right child completely generalizes the actor. In the diagram, the middle child has been picked, transforming the query from a knight fighting something (top row) to a person fighting something (middle of second row). At this point Minstrel might recall a peasant fighting a troll, a constable fighting a bandit, or other similar situations. When adapted back, a story about a knight fighting a troll or bandit makes sense. If the recall were to fail however (if there are no stories in the library about people fighting), there are two further children (the bottom row in the diagram): relaxing the constraint that there is a person involved or releasing the constraint of a fight being in the matching story. Without the fight involved, Minstrel might match a story about a princess eating a berry, yielding a much stranger story after adaptation: a knight fighting a berry.

Skald contains 16 TRAMs, and each of these can be applied in any order and often with multiple possible targets within a given query. TRAMs operate inside a backtracking search tree, allowing for many different sets of TRAMs to be tried in order

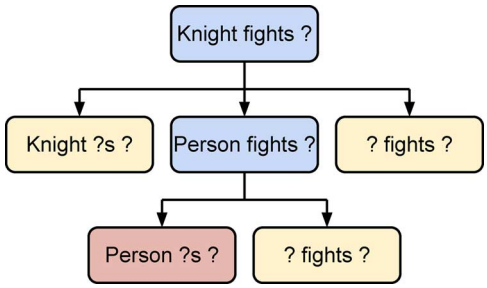

Fig. 3. TRAM search space.

to eventually find a match for a given query. Because this search space often has a branching factor in the hundreds (each way to apply each TRAM is a branch in the search space), there are optional depth and exploration limits that can be used to keep the searches from taking too long.

2) Changes: The original Minstrel applied TRAMs randomly, but Skald supports multiple techniques for directing its TRAM searches. Included are deterministic and fully random search methods for testing purposes, as well as a weighted random method for optimizing TRAM results both in terms of speed and quality. The weights associated with each TRAM (assigned by hand) correspond roughly to how much it modifies the original query. TRAMs are then selected randomly with probability inversely proportional to their weights, so that TRAMs which have more drastic effects are chosen less frequently. By searching more intensely in the part of the search space closest to the original query (in terms of the strength rather than the number of TRAMs), this technique provides quick results which generally make sense even when multiple TRAMs are required to get a result. In a previous analysis of Skald's creativity [23], we discuss the tradeoff being negotiated here between increased sanity and reduced creativity. Because a fully random search method is implemented within Skald, we were able to compare Minstrel's original design with our own and see how this subsystem affected the overall system.

E. Author-Level Planning

1) Functionality: The author-level planning system pursues author-level goals (ALGs) by retrieving and executing ALPs that serve specific functions in generating a story. High level goals consider the story as a whole, and represent tasks such as deciding on a theme for the story or checking the story for opportunities to insert foreshadowing. At the same time, lower level goals concern things such as filling out the details of a particular state or act within the story, or checking that a particular story node is consistent and properly motivated. Some of the ALPs encode microtheories about storytelling (theories of consistent motivation, for example) that help Minstrel produce consistent output. Other ALPs rely on the story library to act as a model of a well-formed story, using the TRAM system to fill in pieces of the story under construction with appropriate material.

Skald does not simply generate free-form story structures. Instead, it relies on templates to structure its stories. These story templates contain rough specifications for important parts of the plot. Skald generates stories by selecting an appropriate template and then filling in the details of that template, adding extra scenes to the story as necessary along the way.

Once a goal is selected and a set of plans is found for it, plans are tried one by one until one succeeds. If all available plans fail, the system re-enqueues the current goal with half of its original priority and drops it entirely if this would put it below a minimum priority threshold. This convention allows goals to interact: if one goal cannot be solved initially, other goals are attempted in the hopes that they will alter the story configuration and make the initial goal solvable. Once the goal queue is empty, story generation is done.

2) Changes: One change to the ALP system is that Skald uses generic story templates (GSTs) instead of what Turner called planning advice themes (PATs). These two constructs share the same structure but differ thematically. Turner asserted that the stories Minstrel would generate should all revolve around "planning advice": they would all have some moral that could be taken as a kind of advice encoded in a story. He believed that focusing on these types of stories was important, and so he called his story templates PATs and populated his template library exclusively with this type of content. The template used to generate "The Mistaken Knight," for example, consists of a character rashly doing something which they later regret, but which they are unable to take back. We have retained some stories that fall into this category, but we want to apply the Minstrel architecture more generally, and so in our system templates are just called GSTs, and we have no ideological position on how they should be constructed.

Besides this change in the types of templates used, Skald also uses a different set of ALPs than Minstrel did. After implementing the original set of ALPs based on Turner's descriptions, we found that some of them seemed prone to disrupting the story. For example, an ALP designed to ensure that all story actions are motivated often added nonsensical motivations. This was likely caused by the fact that our story library differed from Turner's: the results of trying to add a particular type of node and fill it in with TRAMs depend heavily on the content of the story library. We disabled the ALPs that were prone to poor results, and in addition, we added an ALP because of an issue we noticed: actors that were killed in the story would continue to initiate actions later on. We cannot be sure how Turner handled this issue (if it came up) but felt the inclusion in Skald was appropriate. All of the ALPs included in Skald can be enabled or disabled, allowing comparisons between the original set and our own.

F. Boredom

1) Functionality: Minstrel has a notion of boredom which is crucial to the system's creativity when the TRAM system is used repeatedly. To add variability to the system, Minstrel is programmed to get bored with one solution to a problem and, as a result, to search for novel solutions. This is implemented in Skald as a table of query/solution signatures coupled with a boredom value. Every time a query is given to the TRAM system and a solution returned, the boredom value of that signature is incremented. High boredom query/solutions will not

be used, so as the boredom value for a pair rises, other solutions must be found for given queries. Without this method of enforcing variation, Minstrel would often generate duplicate stories, though the random nature of the TRAM searches contributes to variation even without boredom.

2) Changes: Skald's boredom system is different from Minstrel's boredom system. Originally, Minstrel only incremented boredom, and when a solution had been used twice, it would never be used again. While useful for forcing interesting results, this had the side effect of quickly exhausting the story library, resulting in increasingly incoherent stories after only a few had been generated. Our modified system does not produce quite as interesting results as quickly, but instead distributes variation across dozens or hundreds of stories.

Skald's boredom system fractionally decrements the values of all signatures in the boredom table with each call to the TRAM system, gradually moving them below the boredom threshold. Functionally, this means that signatures refresh over time, allowing them to be reused in subsequent stories. Using this system, Skald establishes a cyclical pattern of boredom, and the effect, once the pattern has been established, is similar to generating each new story using a random subset of the story library, always avoiding recent results. This randomization produces a sustainable loop that distributes the creative potential of the system evenly over a large number of stories.

G. Instantiating a Story Node

Fig. 1 shows the sequence of communication between the ALPs, the story library, and the TRAM system during the execution of a "GeneralInstantiate" ALP. When the ALP system selects and begins executing this plan, it queries the story library (step 1) to get a copy of the template that it is working on. The plan will be targeting a particular node, and the ALP system will ask for only nodes within distance 1 of the target node: this is the "current scene" (step 2). The ALP system then sends the current scene to the TRAM system (step 3), asking it to use imaginative recall to fill in at least the target node (while matching as much of the scene as possible). The TRAM system then recursively attempts to transform the query, find a match, and adapt the result. In this example, only one transform step was required (step "T"), before the resulting query (step 4) matched a story in the story library (step 5). This match was then adapted (step "A") to produce the result, which the TRAM system returned to the ALP system (step 6). Finally, the ALP system will update the template in the story library (not shown in the figure). In this case information from "story A" was used to fill in a single node in the in-progress story "template 1." The ALP system then proceeds to its next task, which will generally be checking the newly instantiated node for defects.

IV. Rational Reconstruction

Our work on Minstrel Remixed was a rational reconstruction project: we reconstructed the original Minstrel system as a means of understanding it. The differences between Minstrel, Minstrel Remixed (our reconstruction), and Skald (which moves further away from the original) are shown in Fig. 4. All of the changes described in Section III are shown in Fig. 4, as well as some other details that vary between the systems. Whereas Minstrel Remixed is a very close reconstruction of Minstrel, Skald includes outright modifications that we have been working on which help make it more robust and which make it better at generating story fragments for interactive contexts. Note that for the most significant changes in Minstrel Remixed (the TRAM selection algorithm and the boredom algorithm), our code can be recompiled to mimic the original Minstrel for the purposes of comparing between the two implementation choices.

Feature	Minstrel	M. Remix	Skald
Architecture	Monolithic	Monolithic	Modular
Language	Lisp	Scala 2.8	Scala 2.9
Source Code	No	Yes	Yes
Library Lang.	??	XML	Formulaic English
Interface	CLI	Web, CLI	Web, CLI, API
Story Count	?	14	14
Template Count	6	6	5
ALP Count	33*	27	25
TRAM Count	25*	13	13
Global Nouns	??	Yes	Yes
State Changes	Explicit	Explicit	Inferred
TRAM Method	Random	Weighted	Weighted
TRAM Weights	Equal	Weighted	Weighted
Boredom	Static	Dynamic	Dynamic

* Turner did not use all of the ALPs or TRAMs that he described in his dissertation; some turned out to be impractical or produced nonsense.

Fig. 4. Minstrel, Minstrel Remixed, and Skald.

A. Alterations

Minstrel was constructed both to test and showcase the idea of imaginative recall. To that end it was a success but Turner specifically did not create Minstrel as a general-purpose story generator (see the quote on [4, pp. 195–196]). One of the goals of our work is to discover alterations that could significantly improve its robustness and thus its usefulness as a story generator within a larger system rather than as a demonstration of a specific model of creativity.

One such alteration is the boredom mechanic: Turner's original mechanism strictly limited how many stories could be generated from a given library by rapidly invalidating all stories in the library (Turner gives an example of having his system run out of good material after about five stories [1]). This is fine for demonstrating that imaginative recall works, but we modified this boredom mechanism to "use up" the story library more slowly (see [32] for more details).

We also altered the TRAM selection algorithm to deal with the issue of library reliance. Although the original Minstrel produced interesting output, we found that without a library tailored to match both its templates and its transformation methods, it would often produce nonsense. To show that this reliance on a

strong library could be ameliorated, we ran an experiment comparing our modified version (using weighted random TRAM searches and our updated boredom algorithm) to a version of our code without those changes. Using each version, we generated 75 stories from a single template, recording the outcome of each call to the TRAM system. Each case resulted in about 500 total TRAM calls, which was enough to compare them with some accuracy.

Compared to the strict reconstruction, our modified version reduced the failure rate of tram searches from almost 19% to only 3.5%, reduced the average number of TRAMs tried per search from 58 to 16, and reduced the average number of TRAMs used when no direct match was found from 2.4 to 1.4 (full details of this study are in [32]). The data also spoke to the tuning of our library compared to Turner's: he reported in [1] that 88% of TRAM searches in his original system found a result from the story library without using any transformations (i.e., they matched some part of the library directly). Using his boredom algorithm with our library, we found that only 59% of our searches matched directly, and using our modified boredom, 72% of our searches matched. Even with our modifications to boredom, this corresponds to more than a twofold increase (12% versus 28%) in the use of transformations for retrieving cases: Turner's story library matched his templates (and the nodes added by his ALPs) much more closely than ours does.

Our story library is clearly a significant factor in the difference in quality between Minstrel's original stories and the ones we can generate, but this very fact reveals properties of the imaginative recall algorithm. Rather than tailoring our library to the templates that we use in order to increase direct recall, we have focused on changing the system to make better use of the story library that we have, since this latter approach promises improvements which would be useful to other people using the system.

The changes to both the boredom system and the TRAM selection mechanism are motivated by our plans to work with Skald as a component of a larger system. However, by experimenting with both our modified version and a version that mimics Turner's more closely, we can take design choices that Turner made and subject them to experimental analysis. These alterations are opportunities to understand more about how Turner's theories of imaginative recall can be put into practice.

B. Unknowns

Besides trying to alter Minstrel to serve our purposes, we also had to make some design decisions without knowing what Turner did. Turner's 900-page dissertation includes many specifics, including example traces, but even with these references some things remained unclear. For example, when we started creating story graphs to be stored in the library (and generating story graphs as output) we were not sure what scope the nouns should have. Should the Knight Lancelot be treated as the same entity even when he appears in different stories? Or should each story get its own instance? This choice has some subtle effects on how stories get built, especially when it influences graph matching. Turner never made a clear statement about this point in his dissertation and the examples that he gave did not resolve the issue. In this case, we tried both but decided that globally scoped nouns improved story quality. Perhaps that decision would be made differently in a different domain, but the domain that Turner used has a strong and consistent set of characters across many stories. Details like this likely seem obvious to the original author but upon reconstruction are revealed as important and potentially confusing choices.

Looking at the source code for Minstrel might have answered many of our questions. That being said, working from the original source code might also have made it difficult to discover authorial biases. As an example, had we used Turner's original story library, we might never have discovered how sensitive the system was to changes in the library. At one point we did contact Turner and learned that the source code for Minstrel was available, and that the tapes it was stored on could be shipped to us. Just like books, however, bits can decay and become unusable over time (a problem which digital arts historians have already started to grapple with [35], [36]). In this case, technological obsolescence at both the hardware and software levels stood in our way: to read Minstrel's source code, we would need to procure a tape reader along with hardware and software environments that could run the variant of Lisp that it was written in. In the end, we did not look at the original source code, both because of the difficulties involved and in order to avoid biasing our reconstruction.

C. Implications for Future Work

Ultimately, reconstructing Minstrel proved quite difficult. Missing details forced us to experiment with several systems, and some things, like the full contents of the original story library, we were simply unable to reconstruct. As much as some of these things are opportunities to learn more about how the implementation of imaginative recall influences its performance, in some cases, having more information would have been helpful. Simply saying that researchers should describe their systems more thoroughly or that they should spend more time making sure that their code and data are preserved is not productive, however; the pace of research is often dictated by outside terms, and even the most thorough author will miss things due to how close they are to the work. As the number and complexity of influential systems grows, spending the effort to look deeper into these systems to investigate these unknown design choices will become more important. Fortunately, open sourcing projects is becoming a more widely used practice and the more researchers and system designers adopt this practice, the more likely it is that source code and documentation will be easily available to help future rational reconstruction efforts.

V. Examples

A. An Example Story

To demonstrate the differences between what we can produce and Turner's examples qualitatively, we include an example story here. Note that Minstrel originally used a natural-language-generation framework called Rhapsody to turn its graph structures into English sentences. Although we have reconstructed the core Minstrel components, we have not integrated our version with a natural-language-generation framework yet, and so this story was translated from a graph structure

into English by hand, with the idea of mimicking the style of Turner's original output.

SMAUG, MASON, AND LANCELOT

Once upon a time there was a dragon named Smaug. Smaug wanted a knight named Lancelot to be dead. One day, Smaug found a king named Mason and fed him poisoned lamb, causing Mason to become poisonous. Later, Smaug presented Lancelot with a present: Mason. As planned, Lancelot was hungry and ate Mason. Later on because of this, Lancelot died. Smaug felt good.

This story was picked both to show the general subject and depth of our stories and to demonstrate some particular issues that we have run into. Although we have not collected statistics on the overall quality of the stories our system can produce, we do frequently run into stories that have quirks similar to those this one has.

It is immediately apparent that this story is much shorter than Turner's example "The Mistaken Knight." The main reason for this is that we had disabled some of Turner's ALPs because of the frequency with which they produced bogus content. In "The Mistaken Knight," for example, an ALP which tries to add justifications for what it considers inconsistent events added the entire scene where Lancelot and Andrea met and fell in love, starting just from a node representing that Lancelot loved Andrea. When we turn this ALP on in our system, it adds such scenes (in this example it might add a scene explaining why Smaug wanted Lancelot to be dead) but they are usually nonsensical. This is an example of how the story library needs to be tuned to match both the template(s) used and the specifics of the recall system: by adding stories to our library containing reasons for dragons wanting to kill knights, we would be able to use the aforementioned ALP with this template without trouble. However, this would not be of much use when using another template which did not involve a similar situation, so our approach was to turn off the offending ALPs rather than expand the story library to support each one.

This story further illustrates the brittleness of Skald: it does not make sense that Lancelot would eat a king. This story is based on a template where someone poisons a piece of food in order to kill their enemy. In this version, the main character is a dragon, and its enemy is a knight. So far this is an interesting instantiation of the template, but then the poisoned food becomes a poisoned king, which does not make much sense. Of course, Skald does not have the commonsense knowledge to realize its mistake: it can only assume that the transformations and adaptations applied by the TRAMs are small enough to keep things sensible. In this case, Skald found a story in its library about a dragon eating a king. Because of this, it assumed that a king was something that could be eaten (by anyone) and thus the king is used as the poisoned food in this story.

This shows how Skald's story library must be matched well to its story templates and the TRAMs even when ALPs are not involved. If the library had contained a scene where a knight ate something, then when searching for a food item to substitute, it would likely have found that scene instead of the scene where the dragon eats a king, since the original query in this case is "Lancelot eats something," and Lancelot is a knight. Of course, finding that scene first depends on the set of TRAMs used: here a TRAM which generalizes actor types allows "Lancelot" to become "a knight" in fewer steps than it takes for "Lancelot" to become "any actor" (which would be able to match "Smaug the dragon" in "Smaug the dragon eats a king"). At the same time, the TRAM weights that we introduced in Skald allow the generalize actor TRAM to "cost less" than the generalize constraint TRAM, which would turn "Lancelot eats something" directly into "any actor eats something." So in order to avoid this kind of incoherent situation, a balanced approach considering the story library, the story templates, and the TRAM set is required. This kind of issue is what makes Minstrel (and the current version of Skald) a brittle system.

B. *Conspiracy Forever*

Conspiracy Forever is a prototype video game that was designed and implemented using Skald (then Minstrel Remixed) as an engine. Game play involved a dozen or so players using their web browsers to explore a 1990s hacker drama. Each player was an individual conspiracy theorist who could connect to other servers (which were either randomly seeded or the "home" servers of the other players) and manipulate the files found therein. Each file was a fragment of a story that Skald had generated and then used TRAMs on to obfuscate and morph (see Fig. 5). The goal of the game was for players to piece together a story from the fragments that they thought best fit together. Players enjoyed swapping the most outrageous fragments under increasingly evocative names. *Conspiracy Forever* demonstrates a different way to use story generation in game play: Generate many partial story pieces and prompt humans to search for narratives among them.

VI. EVALUATING MINSTREL

Having reconstructed Turner's Minstrel, what can we say about the original system? Does it live up to the promise that inspired this project? What are its properties as a story-generation system (as opposed to an implementation of a model of creativity)? Rebuilding the system has enabled us to address these questions. As we noted in Section IV, our reconstruction revealed that Turner's original system was dependent on an ecosystem of stories, templates, ALPs, and TRAMs that all functioned together. Getting it to produce stories in a new domain requires not just authoring stories and templates in that domain, but also tuning those things to work with the given TRAMs and ALPs, as well as producing some new TRAMs and ALPs specific to that domain. Minstrel's imaginative recall algorithm does show some potential as a tool for producing dynamic stories, but it certainly is not a one-size-fits-all solution to the problem of generating narrative.

Having worked extensively with the system, there are other noteworthy properties that are not apparent at first glance. One is that the system has no way of implementing templates where states, rather than just nouns, have to be consistent (but variable) between different parts of the template, for example, a template where someone feels an unspecified emotion and then

another character knows that they felt that emotion is impossible. This example is not unreasonable, because the TRAM system should be able to dynamically fill in the emotion based on the story library. However, because Minstrel does not have a way of guaranteeing that a specified string will remain consistent across nodes, such a template is likely to be rendered inconsistent during generation (e.g., "Lancelot felt happy. Andrea knew that Lancelot felt sad."). While noun objects can be generic, and thus a single recall might affect the identity of an actor throughout the story, no equivalent mechanism exists for the strings used to represent state values. In Skald, we have added a system of string references to address this, but without reconstructing Minstrel, this kind of system dynamic would not have been easily observable.

Another idiosyncrasy is that Minstrel has no way of representing adjectives or adverbs. They could be encoded into the set of actions used (i.e., have several distinct actions like "fight quickly," "fight fiercely," "fight desperately") but this strategy would greatly increase the requirements for the story library, as each separate action would need to be used several times. In general, Minstrel's graph representation is quite flexible, but the lack of adjectives and adverbs limits the nuances it can represent. These could be added during a graph-to-English conversion, but to do this intelligently would require either annotations in Minstrel's graphs or a smart system for deciding which adjectives/adverbs to use where.

These properties of Minstrel help define its niche within the ecosystem of story-generation systems. Discovering them was not the result of a close reading of Turner's dissertation, but rather the result of an attempt to use our reconstructed system to actually generate content for a game. The pragmatic constraints imposed by an attempt to create a media experience are different from those imposed by a scientific study that tests some well-defined hypothesis, and working to meet those constraints sheds light on practical properties of the system that may not seem relevant at a purely theoretical level.

VII. Conclusion

Our reconstruction of Scott Turner's 1993 Minstrel system [1] has allowed us to test the boundaries of Minstrel's capabilities. By creating Skald based on the blueprints laid out in Turner's dissertation we were able to tease out design decisions and system dynamics which allowed us to learn more about how Turner's idea of imaginative recall can be applied.

We discovered that the original Minstrel relied heavily on a well-tailored story library to produce quality output. To make Skald more flexible, we modified the TRAM system to use weighted search instead of purely random search. In addition, we found that the boredom mechanism exacerbated the problem of overreliance on a tailored library: it banned using any single result more than twice. To deal with this, Skald slowly reduces the boredom value for all results, which allows it to generate dozens of stories with sustained variation. Our modifications both reduced the number of transforms required in the TRAM system and increased the rate of direct matches to the story library.

Our rational reconstruction of Minstrel has produced not only lessons about the original system, but also a working new system called Skald which we have made publicly available.[2] Although Skald is in some senses still a brittle system, relying on several components being tuned together to avoid incoherent output, by playing to its strengths, we have been able to create a prototype game system that uses Skald to construct conspiracy fragments. Now that Skald is complete, we are interested in applying it to procedurally generate story content for games and other interactive artworks.

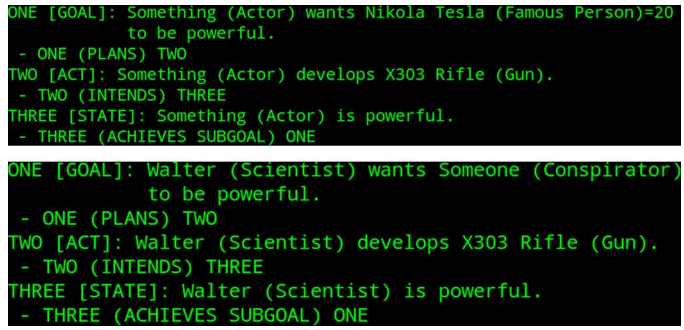

Fig. 5. The same initial story fragment altered by TRAMs in two different ways and distributed on different servers for players to find.

Besides using Skald for content generation, more work could be done to evaluate its creativity, and building systems to make its core imaginative recall system more robust would also be useful. Alternately, building tools for authoring domains might present a solution to Skald's brittleness: perhaps the necessary tuning of the various components can be semi-automated to provide a simpler interface for authors.

Throughout this document we have pointed out various constraints and restrictions that limit Minstrel's applicability. On the whole, however, we think that Skald is a powerful story-generation system. Despite requiring some care to avoid incoherent output, it scales naturally with increasing story library size, and because the core operates on single story scenes at a time, it can easily be adapted to produce content that fits into a larger system. We hope that by releasing it to the community we can inspire others to experiment with Minstrel-style story generation.

References

[1] S. Turner, "MINSTREL: A computer model of creativity and storytelling," Ph.D. dissertation, Dept. Comput. Sci., Univ. California Los Angeles, Los Angeles, CA, USA, 1993.
[2] P. Gervas, "Computational approaches to storytelling and creativity," *AI Mag.*, vol. 30, no. 3, pp. 49–62, 2009.
[3] M. Sharples, "Storytelling by computer," *Digital Creativity*, vol. 8, no. 1, pp. 20–29, 1997.
[4] N. Wardrip-Fruin, *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. Cambridge, MA, USA: MIT Press, 2009.
[5] J. Meehan, "TALE-SPIN," in *Inside Computer Understanding: Five Programs plus Miniatures*, R. Schank and C. Riesbeck, Eds. Hillsdale, NJ, USA: Lawrence Erlbaum, 1981, ch. TALE-SPIN.
[6] M. Lebowitz, "Creating characters in a story-telling universe," *Poetics*, vol. 13, no. 3, pp. 171–194, 1984.
[7] M. A. Musen, J. H. Gennari, and W. W. Wong, "A rational reconstruction of internist-I using protege-II," in *Proc. Annu. Symp. Comput. Appl. Med. Care Amer. Med. Inf. Assoc.*, 1995, pp. 289–293.

[2]Skald is available at: http://minstrel.soe.ucsc.edu/

[8] S. Klein, J. D. Oakley, D. J. Suurballe, and R. A. Ziesemer, "A program for generating reports on the status and history of stochastically modifiable semantic models of arbitrary universes," Comput. Sci. Dept., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep. TR142, 1971.
[9] S. Klein, J. F. Aeschiliman, D. Balsiger, S. Converse, C. Court, M. Foster, R. Lao, J. D. Oakley, and J. Smith, "Automatic novel writing: A status report," Comput. Sci. Dept., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep. TR186, 1973.
[10] S. Klein, "Control of style with a generative grammar," *Language*, vol. 41, pp. 619–631, 1965.
[11] J. Meehan, "The metanovel: Writing stories by computer," Ph.D. dissertation, Dept. Comput. Sci., Yale Univ., New Haven, CT, USA, 1976.
[12] N. Dehn, "Memory in story invention," in *Proc. 3rd Annu. Conf. Cogn. Sci. Soc.*, 1981, pp. 213–215.
[13] M. Riedl, "Narrative planning: Balancing plot and character," Ph.D. dissertation, Dept. Comput. Sci., North Carolina State Univ., Raleigh, NC, USA, 2004.
[14] M. Riedl and M. Young, "From linear story generation to branching story graphs," *IEEE Comput. Graph. Appl.*, vol. 26, no. 3, pp. 23–31, May-Jun. 2006.
[15] M. Theune, S. Faas, D. K. J. Heylen, and A. Nijholt, "The virtual storyteller: Story creation by intelligent agents," in *Proc. Technol. Interactive Digit. Storytelling Entertain. Conf.*, Darmstadt, Germany, Mar. 2003, pp. 204–215.
[16] N. M. Sgouros, "Dynamic generation, management and resolution of interactive plots," *Artif. Intell.*, vol. 107, no. 1, pp. 29–62, 1999.
[17] V. Bui, H. Abbbass, and A. Bender, "Evolving stories: Grammar evolution for automatic plot generation," in *Proc. IEEE Congr. IEEE Evol. Comput.*, 2010, DOI: 10.1109/CEC.2010.5585934.
[18] J. Zhu and S. Ontanón, "Towards analogy-based story generation," in *Proc. 1st Int. Conf. Comput. Creativity*, 2010, pp. 75–84.
[19] R. Pérez y Pérez, "MEXICA: A computer model of creativity in writing," Ph.D. dissertation, Schl. Cogn. Comput. Sci., Univ. Sussex, Brighton, U.K., 1999.
[20] C. Fairclough and P. Cunningham, "A multiplayer case based story engine," Dept. Comput. Sci., Trinity College Dublin, Dublin, Ireland, Tech. Rep., 2003 [Online]. Available: http://www.tara.tcd.ie/jspui/handle/2262/12599
[21] P. Gervas, B. Diazagudo, F. Peinado, and R. Hervas, "Story plot generation based on CBR," *Knowl.-Based Syst.*, vol. 18, no. 4-5, pp. 235–242, Aug. 2005.
[22] V. Propp, *Morphology of the Folktale*. Austin, TX, USA: Univ. Texas Press, 1968.
[23] B. R. Tearse, P. Mawhorter, N. Wardrip-Fruin, and M. Mateas, "Experimental results from a rational reconstruction of Minstrel," in *Proc. Int. Conf. Comput. Creativity*, 2011, pp. 54–59.
[24] M. Boden, *The Creative Mind*, 2nd ed. London, U.K.: Routledge, 2004.
[25] G. Ritchie, "Assessing creativity," in *Proc. Symp. AI Creativity Arts Sci.*, Heslington, U.K., 2001, pp. 3–11.
[26] S. R. Turner, *The Creative Process: A Computer Model of Storytelling And Creativity*. Hillsdale, NJ, USA: Lawrence Erlbaum, 1994.
[27] G. Wiggins, "A preliminary framework for description, analysis and comparison of creative systems," *J. Knowl. Based Syst.*, vol. 19, no. 7, pp. 449–458, 2006.
[28] S. Colton, J. Charnley, and A. Pease, "Computational creativity theory: The FACE and IDEA descriptive models," in *Proc. 2nd Int. Conf. Comput. Creativity*, 2011, pp. 90–95.
[29] B. Tearse, N. Wardrip-Fruin, and M. Mateas, "Minstrel remixed : Procedurally generating stories," in *Proc. 6th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, 2010, pp. 192–197.
[30] B. Tearse, P. Mawhorter, M. Mateas, and N. Wardrip-Fruin, "Minstrel remixed: User interface and demonstration," in *Proc. 7th Artif. Intell. Interactive Digit. Entertain. Conf.*, 2011, pp. 217–218.
[31] B. Tearse, "Minstrel remixed: A reconstruction and exploration," in *Proc. 6th Int. Conf. Found. Digit. Games*, 2011, pp. 253–255.
[32] B. Tearse, P. Mawhorter, M. Mateas, and N. Wardrip-Fruin, "Lessons learned from a rational reconstruction of Minstrel," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 249–255.
[33] B. Tearse, M. Mateas, and N. Wardrip-Fruin, "MINSTREL Remixed: A rational reconstruction," in *Proc. Intell. Narrative Technol. III Workshop*, Monterey, CA, USA, 2010, DOI: 10.1145/1822309.1822321.
[34] R. C. Schank, "Conceptual dependence: A theory of natural language understanding," *Cogn. Psychol.*, vol. 3, no. 4, pp. 552–631, 1972.
[35] N. Montfort and N. Wardrip-Fruin, "Acid-free bits: Recommendations for longer-lasting electronic literature from the ELO's preservation, archiving, and dissemination (PAD) project," 2004 [Online]. Available: http://eliterature.org/pad/afb.html
[36] J. Rothenberg, "Ensuring the Longevity of Digital Documents," *Sci. Amer.*, vol. 272, no. 1, pp. 42–47, 1995.

Brandon Tearse received the B.A. degree in psychology from Carleton College, Northfield, MN, USA, in 2004. Currently, he is working toward the Ph.D. degree, advised by N. Wardrip-Fruin, at the University of California Santa Cruz, Santa Cruz, CA, USA.

His current research focuses on automatic story-generation techniques, and he has previously worked in bioinformatics and network security.

Peter Mawhorter received the B.S. degree in computer science from Harvey Mudd College, Claremont, CA, USA, in 2008. Currently, he is working toward the Ph.D. degree, advised by M, Mateas, at the University of California Santa Cruz, Santa Cruz, CA, USA.

His current research focuses on the automatic generation of branching stories and the application of answer set programming to story generation. His past projects include work on robotics, sensor networks, and data streaming.

Michael Mateas received the B.S. degree in engineering physics from the University of the Pacific, Stockton, CA, USA, in 1989, the M.S. degree in computer science from Portland State University, Portland, OR, USA, in 1993, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, USA, in 2002.

He is a Professor of Computer Science, Co-Director of the Expressive Intelligence Studio, and Director of the Center for Games and Playable Media at the University of California Santa Cruz, Santa Cruz, CA, USA. His research focuses on expressive intelligence, which explores the intersection of artificial intelligence, art, and design. Along with A. Stern, he created Façade, which is the world' first fully produced, real-time, interactive story. His recent projects include Prom Week (2012 finalist at IndieCade and the Independent Games Festival) and Immerse [a Defense Advanced Research Projects Agency (DARPA)-funded project on teaching good stranger behavior through immersive full-body interaction with socially aware virtual characters].

Noah Wardrip-Fruin received the MFA degree in literary arts and the Ph.D. degree in special graduate studies from Brown University, Providence, RI, USA, in 2003 and 2006, respectively.

He is an Associate Professor of Computer Science, Director of the Digital Art and New Media MFA program, and Co-Director of the Expressive Intelligence Studio at the University of California Santa Cruz, Santa Cruz, CA, USA. He is the author of two books: *The New Media Reader* (Cambridge, MA, USA: MIT Press, 2003) and *Expressive Processing* (Cambridge, MA, USA: MIT Press, 2009). His recent projects include Prom Week (2012 finalist at IndieCade and the Independent Games Festival) and the 2012 Media Systems gathering [with the National Endowment for the Humanities (NEH), the National Endowment for the Arts (NEA), the National Science Foundation (NSF), Microsoft Studios, and Microsoft Research as partners].