Designing a Modular, Scalable Benchmark for Narrative Experience Management

Molly Siler[1], Stephen G. Ware[1], Gage Birchmeier[1,†], Mira Fisher[1,†] and Lasantha Senanayake[1,†]

[1]Narrative Intelligence Lab, Department of Computer Science, University of Kentucky, Lexington, KY, USA 40506

Abstract

We discuss the ongoing development of MicroTales, a collection of elements that can be combined to generate interactive narrative environments of varying size and complexity. MicroTales aims to fill a gap among existing AI benchmarks by featuring active non-player characters, a wide variety of actions, and the possibility to soft-lock the problem. Its purpose is to provide a clearly defined environment to compare different experience management algorithms without enforcing any one definition of what makes a good story. We present design goals and a sketch of an initial design, and invite community feedback to help make our benchmark reusable by other narrative intelligence researchers.

1. Introduction

An *experience manager* is an agent that dynamically adapts a game environment to the player. We consider experience managers in an interactive narrative that assume control of the non-player characters (NPCs). Compared to a manually-authored interactive narrative, an experience-managed interactive narrative can include vastly more branching paths without a proportional increase in designer effort. However, it is challenging to make an experience manager replicate the kind of experience that a skilled human author can craft—one that guides the player through a cohesive narrative with believable NPCs to a satisfying end, while making the player feel they had a meaningful role in that narrative.

How do we test how well an experience manager meets this challenge? It is common to evaluate an experience manager using a custom-built interactive narrative environment [1]. The practice of evaluating with only one environment is understandable, as it is a time-consuming process to engineer a game or even to repurpose one so it showcases the features of a new experience management architecture. However, this practice limits the generalizability of results. We have identified a need for more benchmarks with parameterized generation of new problem instances, similar to those commonly used in other AI research areas like classical planning [2].

This paper proposes MicroTales, a collection of elements that can be combined to generate interactive narrative environments of varying size and complexity. MicroTales problems are highly modular, with different elements becoming available only when certain *extensions* are enabled. For example, the *Monarchy* extension adds castle locations, noble characters, a crown item, and an action for crowning a monarch. When the *Monarchy* extension is enabled, these elements can be used in a problem.

Along with the story world and characters, a problem provides *story goals* for an experience manager and *character goals* as directives for what the individual characters want. However, the experience manager is not restricted in how it makes characters act, and evaluation metrics are deliberately left open. We want MicroTales to aid exploration of different theories about what makes a good story.

MicroTales was conceived based on our own lab's research needs, but we hope to make it easily reusable by other researchers. We are presenting it during its ongoing development with an interest in

Joint AIIDE Workshop on Experimental Artificial Intelligence in Games and Intelligent Narrative Technologies, November 10-11, 2025, Edmonton, Canada.

[†]These authors contributed equally.

molly.siler@uky.edu (M. Siler); sgware@cs.uky.edu (S. G. Ware); gage.birchmeier@uky.edu (G. Birchmeier); mira.fisher@uky.edu (M. Fisher); lasantha.senanayake@uky.edu (L. Senanayake)

community feedback. The remainder of this paper discusses our objectives and presents a tentative design along with some working examples.

2. Design Objectives

This section describes the desiderata that MicroTales is meant to fulfill. As we discuss in Section 3, there are existing AI benchmarks that share some of these traits, but none that reflect all of them.

Game-Like Storytelling Environment: MicroTales is designed to imitate a simple game environment where a player controls one character, and a human game master or an automated experience manager controls the other characters and the environment. Our intention for MicroTales is to reflect the kinds of problems that an experience manager would need to solve interactively, meaning that new solutions are generated during play to adapt to unexpected player actions. At the same time, the problems will be versatile enough to benchmark offline planning approaches.

Action-Level Storytelling: Stories can be generated at many levels of granularity. The basic unit of story content in MicroTales is an action, which would typically correspond to a sentence of a written story with a subject, verb, and possibly one or two objects. Early intelligent narrative systems like Universe [3] and Façade [4] create stories one scene at a time, and an atomic piece of storytelling content corresponds to roughly one paragraph of a written story. Systems based on text generation, like AI Dungeon [5], generate stories on a token level. Action-level storytelling exists between these two poles.

Scalable Along Several Dimensions: It should be possible to generate a wide variety of problems of varying size and difficulty, from trivially easy to nearly impossible. MicroTales can vary in the size of the map, the number of characters, the number of items, the types of actions that are available, and the number and complexity of goals that need to be achieved. The ability to generate a wide variety of problems has several advantages. By generating problems of different sizes and with different features turned on or off, we can make a controlled study of what makes experience management problems easy or hard to solve. Having many problems to train on also encourages learning-based approaches to generalize rather than overfitting to a few specific problems.

Active NPCs: One purpose of MicroTales is to fill a gap we perceive in the kinds of games available to test computational storytelling approaches, namely story games with active non-player characters (NPCs). Many story games, like the interactive fiction *Zork* [6] and its descendants, rely on the player character to take most of the actions in the story. Actions by other characters are often short and direct reactions to the player. An ideal solution to a MicroTales problem will feature NPCs taking actions motivated by their goals, ideally in ways that are predictable or at least believable to the player, allowing the player to plan around anticipated NPC actions.

Following Genre Tropes: While MicroTales is meant to allow a wide variety of problems, it is also attempting to follow the genre tropes of simple medieval computer role-playing games. This serves two purposes. First, it allows players of an interactive MicroTales game to leverage existing genre knowledge and minimize the amount of onboarding required to understand the game's rules. Second, MicroTales is intentionally attempting to mimic a spread of "naturally occurring" problems from human-designed interactive narratives, complete with their irregularities. MicroTales is not meant to represent every conceivable situation or action. It is not meant to be the only type of benchmark problem that experience managers should be tested on. We are attempting to add to, not restrict, the variety of storytelling problems available for testing.

Wide Variety of Actions: Many planning and reinforcement learning benchmark problems have a limited variety of action types available. For example, the classic *Blocks World* planning problem [7] has only one type of action—moving a block onto another block or onto the table. While it might have a large number of possible actions when we consider all blocks, it has only one *type* of action. Similarly, the *Cart Pole* reinforcement learning problem [8] has only two actions: push the cart left or right. The action can vary in the force applied, but still there is only one or two types of action. We find that storytelling problems tend to involve a large number of action types. MicroTales has many types of actions with different kinds of preconditions and effects.

Soft Locks are Possible: A soft lock means that a solvable problem has been put into a state from which no solutions are possible. The possibility to soft-lock a game or puzzle makes it challenging because solvers must plan ahead. Solvers that rely on performing random actions—like Monte Carlo random rollouts or random exploration in MDP solvers—will need to account for soft locks. Many of the benchmark problems used to test classical planning algorithms do not create the possibility for soft locks. Stories often feature irreversible actions that require careful planning, which is why the possibility of soft locks is allowed, and even preferable, in MicroTales.

Deterministic Actions: Actions in MicroTales are deterministic, i.e., the exact resulting state is known before the action is executed. Determinism allows a wide variety of search and planning techniques to be applied. Real-world problems are not deterministic, but MicroTales is not designed to simulate a real-world problem; it is designed to simulate a game in a virtual environment where the experience manager has full control over the virtual world. While a MicroTales problem in isolation is defined deterministically, the larger problem of experience management in an interactive game may be viewed as nondeterministic since the player may take unexpected actions.

Complex Multi-Agent Interactions: Most problems will feature more than one character. Many existing AI benchmarks feature only a single agent or assume agents are strictly cooperative or competitive. In MicroTales, each character has their own goals, and they should aid or thwart one another to the extent that their goals align or diverge. One of the central challenges of storytelling is ensuring that each character behaves realistically while maintaining a story's central structure. MicroTales provides an environment to compare emergent storytelling via multi-agent systems to centralized storytelling via planning.

No Model of Storytelling Enforced: MicroTales enables the study of what makes an interactive narrative believable, interesting, and fun. It tries to avoid enforcing any particular model of storytelling in its rules. Action preconditions are meant to be minimal requirements on when an action would be "physically possible" in the virtual world. For example, one character can only *give* an item to another if they are in the same location and if the giver has the item. The preconditions do not require that the receiver wants the item; it is up to a model of storytelling to determine whether this action makes sense. To study a model of good storytelling, it must be possible to generate bad stories. Many AI benchmark problems define success as reaching a clear goal via a shortest or lowest cost path, but many MicroTales problems will have "solutions" that are bad stories—that is, sequences of actions that achieve the story goals but are made of actions that are not believable, interesting, or fun. The quality of a MicroTales solution should not be judged solely on whether it achieves goals or on the number of actions it contains. A solution must be evaluated as a story, but MicroTales does not define what makes a good story. In other words, MicroTales defines a set of objects, actions, and goals so that two researchers can be said to be working in the same domain, but it does not define success in that domain.

3. Related Work

Riedl and Bulitko [9] attribute the start of experience management to Bates [10], who proposed that interactive drama be framed adversarially—the aesthetic goals for the player's experience versus anything the player might do that derails those goals—suggesting a computational model similar to a chess algorithm. The term "experience management" itself was coined by Riedl et al. [11], who proposed that techniques for dramatic systems could also apply to intelligent training and tutoring systems. Riedl et al.'s approach models the future trajectory of the narrative as a plan structure, adapting to player derailments via plan repair and replanning. Balancing this preservation of designer goals with player agency has been identified as a major challenge of interactive narrative as a whole [12]. Our project is motivated in part by evaluating our experience managers' progress on this challenge.

Another challenge is the fact that the choices most convenient for an experience manager's goals do not always align with the choices that make sense for the NPCs involved [13]. Riedl and Young [14] proposed *narrative planning*, which uses a single agent to plan for all characters but constrains the actions that can be selected for the plan based on character believability. Later research has refined plan-based models to reason about more features of plot (e.g., foreshadowing and suspense) and character (e.g., social dynamics) [15]. Our characterization of experience management is most heavily influenced by plan-based approaches.

Ramirez et al. [16] noted that to date, there had been few empirical evaluations of autonomous experience managers. More recently, Mori et al. [1] observe that many experience management approaches are tested only in custom-built environments. There have been recent efforts, including those Mori et al. themselves, Thue and Bulitko [17], and Clerc et al. [18] to develop a unified framework for comparison of experience managers, but so far none has achieved widespread community adoption. Rather than trying to encompass experience management as a whole, we accept a degree of specificity while still allowing a diversity of algorithms to be exposed to a variety of situations.

One of our inspirations is the International Planning Competition [2], in which AI planners developed by the community on a common set of problems in a standardized format. Another is Gymnasium [8], which provides an API to a collection of Markov decision process (MDP) environments for testing reinforcement learning algorithms. Many benchmarks within these have relevant features like action-level decision-making and scalability, but when benchmarks are game-like they typically evaluate only decision-making from a player perspective, and they are built around rigid metrics for success like solve time or cumulative reward whereas we are interested in story-centric evaluation criteria.

TextWorld [19] provides a generator for *Zork*-style [6] text-based adventure games and an interface for AI agents to play them; however, the game world lacks active NPCs and rather only reacts to player actions. Conversely, our previous works [20, 21] emphasize online experience management with active NPCs, but each one uses a single predesigned world rather than supporting scalable problem generation.

4. Design Description

This section describes the design of MicroTales, centered around an example. A fuller set of definitions is in Appendix A. As we discuss in Section 5, the framework is a work-in-progress and we are interested in incorporating community feedback from the workshop.

At a low level, a running MicroTales environment is a deterministic state transition system. At any given time, the story world is in a state, which is defined by a complete assignment of values to a set of variables. There is a set of actions that, when taken, determine the world's next state. An action has preconditions that limit when it can occur, defined by a proposition over the variables. An action also has effects that take place when it occurs, defined in terms of new value assignments to variables.

This generic framework does not assume any particular implementation. We want it to be possible to implement MicroTales in everything from the Planning Domain Definition Language (PDDL) [22], used to define classical planning problems, to a Gymnasium environment [8], used to define an MDP for reinforcement learning, to even non-digital implementations, like a board game.

Figure 1: A mockup of the Robin Hood problem in a graphical interface (sprite credit: 16-Bit Fantasy collection by Oryx Design Lab).

Our running example, illustrated visually in Figure 1, is based on the tale of Robin Hood. Figure 2 shows a problem definition in YAML, the primary input format we are considering; we will explain its components throughout this section. The sets of variables and actions are not specified directly in a problem file, but rather implied by higher-level elements. Among other components, a problem includes a map of *locations*, their types, and the *paths* between them, as well as a set of *characters* and their types. For example, in Figure 2, *Sherwood* is a location of type *forest* with paths to all other locations, and *Robin* is a character of type *bandit*.

Most variables are associated with a character, such as the character's current location and held items. Most actions are also associated with a character; the basic actions available in every MicroTales problem are to *walk* between locations that have a path, *give* an item to or *trade* items with another character, and *drop* or *pickup* an item at the current location. For example, the Figure 2 problem implies a variable $location(Robin)$ an action $walk(Robin, Abbey)$ with precondition $location(Robin) = Sherwood$ and effect $location(Robin) = Abbey$. The problem specification does not include an explicit notion of player character—this is specific to the use case—but typically, an interactive narrative will assign a player to choose actions for one character while an experience manager will control the other characters as NPCs.

A core feature of MicroTales is the ability to specify any number of *extensions* in a problem. An extension is a group of related problem elements that become available when an extension is included in a problem. Extensions are designed to allow additional types of actions in a problem, but doing so means enabling the location and character types that would make those actions possible and relevant.

For example, in the Figure 2 problem definition, the inclusion of the *Theft* extension implicitly adds the action *take* to the list of available actions, allowing characters with a *Sword* item to take an item from another character. Characters of the *bandit* type have the special ability to bypass the *Sword* requirement and *take* at any time. The *bandit* character type and *Sword* item are illegal in a default MicroTales problem and specifically made legal by the *Theft* extension, as their relationship to the *take* action is what gives them their unique functionality. Similarly, the problem can include *chapel* locations where the *marry* action can occur because the *Marriage* extension is enabled, and so on.

The initial state of the world is another largely implicit element except where overridden in the problem definition. In Figure 2, Robin's starting location is set to Sherwood explicitly, but John's, Marian's, and the Sheriff's do not need to be specified because the *noble* and *knight* types default to the first *castle* in the locations list, in this case Nottingham, when one is available.

Items in this case are also created from defaults; in this case, as a *knight* the Sheriff starts with a *Sword* and as *noble*s John and Marian start with *Coin*s. Items are not first-class elements in the way that characters and locations are; instead, they exist as variable values such as $left(John) = Coin$ denoting a *Coin* existing in John's left hand. If Robin *take*s the *Coin* from John, the effect is $left(John) = Null$ and $left(Robin) = Coin$, or $right(Robin) = Coin$ if Robin's left hand is full. The advantage of this

```
name: "Robin Hood"
version: 1.0
extensions:
- Crime     # Adds jail; lawful goal; assault and theft now crimes
- Marriage  # Adds chapel; wed goal; marry action
- Monarchy  # Adds castle and Crown; ruling goal; enthrone action
- Theft     # Adds bandit, Coin, and Sword; allows take action
- Violence  # Adds knight and Sword; avenged goal; attack action
locations:
- Sherwood : forest    # Sherwood Forest
- Nottingham : castle  # Nottingham Castle
- Jail : jail          # The Jail
- Abbey : chapel       # Fountains Abbey
paths: # Sherwood Forest connects all locations
- [Sherwood, Nottingham]
- [Sherwood, Jail]
- [Sherwood, Abbey]
characters:
  Robin : bandit    # Robin Hood is a bandit
  John : noble      # Prince John is a noble
  Sheriff : knight  # The Sheriff of Nottingham is a knight
  Marian : noble    # Maid Marian is a noble
initial: # Only need to list values if they are not defaults
- [location, Robin, Sherwood]  # Robin starts in Sherwood Forest
- [right, John, Crown]         # Prince John starts with a crown
goals:
- [Robin, rich, Robin]         # Robin wants coins
- [Robin, wed, Robin, Marian]  # Robin wants to marry Marian
- [John, ruling, John]         # John wants to be or marry the monarch
- [John, wed, John, Marian]    # John wants to marry Marian
- [John, avenged, John]        # John wants his enemies dead
- [Sheriff, ruling, John]      # The Sheriff wants John to be or marry the monarch
- [Sheriff, lawful]            # The Sheriff wants no criminals
- [Marian, ruling, Marian]     # Marian wants to be or marry the monarch
- [wed, Robin, Marian]         # Story goal: Robin marries Marian
```

Figure 2: An example MicroTales problem in YAML format.

scheme is compatibility with an implementation that must declare a finite set of objects at the beginning, such as a PDDL planning domain; the items do not need to have their own individual identities and can be "created" or "destroyed" at will, such as by crafting actions.

A component of the problem definition we have not yet discussed is *goals*. Unlike the other components, goals do not modify states, variables, or actions. Instead, they suggest objectives the individual characters should try to achieve, in the case of *character goals*, or that the story overall should try to achieve, in the case of *story goals*. A goal is declared with these components: the character who has the goal (omitted for story goals); a goal type; and a list of arguments.

For example, in Figure 2, the first line of the goals list declares that Robin has the goal $rich(Robin)$. The *rich* goal type takes a character as a parameter—in this case, Robin himself—and states that the goal-possessing character wants the parameter character to have *Coins*. Meanwhile, the last line of the goals list declares a story goal of $wed(Robin, Marian)$ giving the experience manager itself an objective—namely, a *wed* goal is fulfilled by having the parameter characters be each other's *spouse*. Like character and location types, most goal types are enabled by extensions, such as *wed* having the *Marriage* extension as a prerequisite as it could not be achieved without *Marriage*-specific actions.

MicroTales does not define the relative importance of one goal to another, and when a goal can be achieved in multiple ways, it only defines ordinal preferences. For example, if a character has both *rich*

and *wed* goals, MicroTales does not specify whether it is more important to that character to have *Coin*s or marriage; this is up to the experience manager. Similarly, for *rich*, a character would prefer to have two *Coin*s over one *Coin*, but it is not necessarily twice as good to have two *Coin*s than it is to have one. Again, this is left up to the experience manager.

MicroTales tries to avoid imposing any particular model of what makes a good story. Some good stories may not achieve story goals, and stories that achieve story goals are not necessarily good stories. If goals can be ignored, why define them at all? First, MicroTales is meant to study the challenges of storytelling. Goals provide constraints that make planning a story challenging because the storyteller needs to look ahead to reason about what is and is not possible when deciding what actions to take. Second, in an interactive setting, goals are what will be communicated to players about what the characters want. Stories are meant to involve multiple active characters, so anticipating how non-player characters will act requires information about what they want.

4.1. Example Stories

Here are some example stories that could result from the Figure 2 problem. Recall that the story goal is for Robin Hood and Maid Marian to get married. We assume this problem is presented as an experience-managed interactive game where the player controls Robin Hood.

```
walk(Robin, Nottingham) # Robin goes to Nottingham Castle.
take(Robin, Coin, John) # Robin is rich, so he wants coins.
attack(Sheriff, Robin)  # The Sheriff drops Robin's health to "hurt".
attack(Sheriff, Robin)  # The Sheriff drops Robin's health to "dead".
```

This is a simple but unsuccessful story. Robin robs Prince John, but the Sheriff of Nottingham executes Robin for his crimes. This example does not achieve the story goal and in fact soft-locks the problem, but if the problem is presented as an interactive puzzle it might be the outcome of a reasonable attempt by the player to achieve Robin's goals. Suppose the player tries a different approach.

```
walk(Robin, Nottingham)      # Robin goes to Nottingham Castle.
take(Robin, Crown, John)     # Robins steals John's crown.
enthrone(Robin)              # Robin is crowned king.
arrest(Sheriff, Robin, Jail) # Lawful Sheriff throws Robin in jail.
walk(Marian, Sherwood)       # Marian on her way to release Robin.
release(Marian, Jail)        # Marian unlocks the jail.
walk(Robin, Jail, Sherwood)  # Robin on his way to marry Marian.
walk(Robin, Abbey)           # Robin goes to the chapel.
walk(Marian, Abbey)          # Marian goes to the chapel.
marry(Robin, Marian)         # Marian becomes queen by marriage.
```

This story is more interesting. Robin steals the crown and becomes king. The experience manager is faced with two possible responses from the Sheriff: attack like in the previous story, or arrest Robin and move him to the jail. Killing Robin would end the story before the goal is achieved, but arresting him leaves it open for a happy ending later. Robin can't do anything from the jail, but Marian comes to free him. From here, Marian can marry Robin to satisfy her *ruling* goal, which also satisfies the story goal. We consider this story more interesting because non-player characters take an active role.

While this makes a plausible story, it may not work when presented as a puzzle, since the player might wonder why Robin was allowed to take several actions in a row without any response from the Sheriff. Next, we will consider a story that enforces strict turn-taking, always allowing one player action after one NPC action.

```
enthrone(John, Nottingham)  # John is crowned king.
# The player chooses to wait, passing their turn.
```

```
walk(John, Sherwood)          # John on his way to marry Marian.
take(Robin, Coin, John)       # Robin steals John's money.
walk(Marian, Sherwood)        # Marian on her way to marry John.
take(Robin, Crown, John)      # Robin steals John's crown.
walk(John, Abbey)             # John on his way to marry Marian.
walk(Robin, Nottingham)       # Robin on his way to be crowned.
walk(Marian, Abbey)           # Marian on her way to marry John.
enthrone(Robin)               # Robin is crowned king.
take(Sheriff, Crown, Robin) # Sheriff steals Robin's crown.
walk(Robin, Sherwood)         # Robin on his way to marry Marian.
walk(Sheriff, Sherwood)       # Sheriff on his way to John.
walk(Robin, Abbey)            # Robin on his way to marry Marian.
walk(John, Sherwood)          # John on his way to the Sheriff.
marry(Robin, Marian)          # Marian becomes queen by marriage.
```

John starts by achieving his ambition to become king. He and Marian then set off in the direction of Fountains Abbey for their wedding. On the way, John gets robbed by Robin Hood. The player waiting for John to pass by is a prime example of a player anticipating active NPCs. Robin then goes to Nottingham Castle and crowns himself king before the wedding can take place. The Sheriff takes the crown from Robin and goes to Sherwood Forest hoping to give the crown to John. Robin makes his way to Fountains Abbey and marries Marian, achieving the story goal.

In this example, we assume characters only witness actions that occur in their location (except *enthrone*, which gets announced to everyone throughout the kingdom). The Sheriff is not aware of Robin's crimes, since he was not in Sherwood Forest during the robbery. This model of limited observability is not required. MicroTales does not model character beliefs; that is left to the experience manager, which is free to use this or any other model. For this problem, we think it makes the characters seem more realistic, and it adds a new dimension to the puzzle by encouraging Robin to avoid witnesses.

Given the Sheriff does not know about Robin's crimes, it may seem odd that he would take the crown. The Sheriff thinks he is stealing, which makes him a criminal and goes against his *lawful* goal which calls for minimizing criminals who walk free. The Sheriff takes the crown because he hopes to return it to John so that John can again take the throne. After returning the crown, the Sheriff plans to arrest himself, satisfying all his goals, though the story ends before he can complete his plan. Does it make sense for the Sheriff to arrest himself? This is up to the experience manager.

Notice also how the Sheriff and John meet in Sherwood Forest. According to our earlier assumptions about belief, the Sheriff saw John leave for the forest, but did not see John continue to the Abbey, so the Sheriff wrongly believes John is in the forest. John did not see the Sheriff go to Sherwood, so John wrongly believes the Sheriff is at Nottingham. Luckily, these two meet by chance in the forest despite their wrong beliefs thanks to some coordination by the experience manager.

5. Discussion Points

Our intention in presenting this early-stage work to the Intelligent Narrative Technologies community is to elicit discussion on how we can make the final benchmark relevant to a sufficient breadth of experience management approaches and reusable for other narrative AI researchers. The following are some topics that we are especially interested in presenting for feedback.

One point of discussion is the appropriateness to the target audience. The design commitments of MicroTales preclude its relevance to certain types of narrative generation and experience management: Some experience management approaches focus on tailoring a more reactive game environment to player preferences [23], contrasting with our emphasis on active NPCs. Systems emphasizing highly open-ended story generation with natural language [24] may find the predetermined action space too constraining. Storylet-based systems [25] are potentially but not necessarily compatible with our framework, as they are designed to assemble narratives on a higher level than action. Given

our assumptions of multi-character storytelling within constrained action-based game mechanics, we nonetheless hope to make our benchmark accessible to a diversity of approaches.

A design challenge we have faced is the tension between the objective to avoid enforcing a model of storytelling, and the need to articulate character goals. Although they do not appear in action preconditions and in theory can be ignored at an experience manager's convenience, there are some variables in our state models that represent social rather than physical properties. For problem, in order to define the *avenged* character goal, which says that a character wishes death upon someone's enemies, we needed to introduce a *relationship* variable that tracks friendship and enmity and an explicit definition of what causes those relationships to change. We are looking to find the right balance between defining benchmarks that can clearly express social goals, and ensuring that different narrative approaches can override these definitions with their own models of phenomena like interpersonal relationships [26], personality and affect [27], and social norms [28].

A notable exclusion from our design so far is actions that exist solely for the purpose of communication between characters. Fully open-ended dialog is out of our scope; it would make the action space vast or infinite and sometimes trivialize the situations we want to study where players and experience managers must infer each other's intentions from their decisions [29]. We are considering whether to include possible communication approaches ranging from a simple "ping", e.g., "[character1] tells [character2] about [entity]", to a more precise set of speech acts.

6. Future Work

Our next step after this workshop will be to leverage community discussions to reach a stable version of MicroTales. We will then create tools implementing the framework. One tool will be a procedural generator that can create MicroTales problems scaled to a desired number of locations, characters, and extensions. Others will include an API for running a generated problem with arbitrary player and experience manager agents; sample implementations of random and human-controllable agents as a template for implementing new agents; and a graphical interface to facilitate experiments with human participants, similar to Figure 1. Our latest work will be available on the project webpage at http://cs.uky.edu/~sgware/projects/microtales/.

In the longer term, we will use our benchmark to study story generation and experience management approaches from a variety of angles. In offline benchmarking, we can compare the scalability of narrative planning algorithms and heuristics by varying the size and complexity of problems. In interactive experiments with human participants, we can change the game over multiple within-subjects trials so a player reasons about new situations instead of memorized ones. We will use these experiments to investigate what makes an experience manager effective at balancing properties like cohesive story structure, achievement of story goals, player agency, and character believability.

Acknowledgments

This material is based upon work supported by the U.S. National Science Foundation under Grant No. 2145153 and the U.S. Army Research Office under Grant No. W911NF-24-1-0195. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Army Research Office.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] G. Mori, D. Thue, S. Schiffel, EM-Glue: A platform for decoupling experience managers and environments, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 18, 2022, pp. 266–274.

[2] A. Taitler, R. Alford, J. Espasa, G. Behnke, D. Fišer, M. Gimelfarb, F. Pommerening, S. Sanner, E. Scala, D. Schreiber, J. Segovia-Aguas, J. Seipp, The 2023 International Planning Competition, AI Magazine 45 (2024) 280–296.

[3] M. Lebowitz, Story-telling as planning and learning, Poetics 14 (1985) 483–502.

[4] M. Mateas, A. Stern, Structuring content in the Façade interactive drama architecture, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 1, 2005.

[5] N. Walton, AI Dungeon, https://aidungeon.com/, 2019.

[6] Infocom, Zork I: The Great Underground Empire, 1980.

[7] J. Slaney, S. Thiébaux, Blocks World revisited, Artificial Intelligence 125 (2001) 119–153.

[8] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, O. G. Younis, Gymnasium: A standard interface for reinforcement learning environments, 2024. URL: https://arxiv.org/abs/2407.17032. arXiv:2407.17032.

[9] M. O. Riedl, V. Bulitko, Interactive narrative: An intelligent systems approach, AI Magazine 34 (2013) 67–77.

[10] J. Bates, Virtual reality, art, and entertainment, Presence: Teleoperators & Virtual Environments 1 (1992) 133–138.

[11] M. O. Riedl, A. Stern, D. Dini, J. Alderman, Dynamic experience management in virtual worlds for entertainment, education, and training, International Transactions on Systems Science and Applications 4 (2008) 23–42.

[12] E. W. Adams, Interactive narratives revisited: Ten years of research, in: Games Developers Conference, 2005.

[13] M. Mateas, A. Stern, Towards integrating plot and character for interactive drama, in: Socially Intelligent Agents: Creating Relationships with Computers and Robots, Springer, 2002, pp. 221–228.

[14] M. O. Riedl, R. M. Young, Narrative planning: Balancing plot and character, Journal of Artificial Intelligence Research 39 (2010) 217–268.

[15] R. M. Young, S. G. Ware, B. A. Cassell, J. Robertson, 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, 37 (2013) 41–64.

[16] A. Ramirez, V. Bulitko, M. Spetch, Evaluating planning-based experience managers for agency and fun in text-based interactive narrative, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 9, 2013, pp. 65–71.

[17] D. Thue, V. Bulitko, Toward a unified understanding of experience management, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 14, 2018, pp. 130–136.

[18] J. Clerc, D. Lourdeaux, M. Sallak, J. Barbier, M. Ravaine, Modeling interactive narrative systems: A formal approach, in: International Workshop on Computational Models of Narrative, volume 8, 2025.

[19] M.-A. Côté, Á. Kádár, X. Yuan, Q. Kybartas, T. Barnes, E. Fine, J. Moore, M. Hausknecht, L. El Asri, M. Adada, W. Tay, A. Trischler, TextWorld: A learning environment for text-based games, in: Computer Games, Springer, 2019, pp. 41–75.

[20] M. Siler, S. G. Ware, Structure, agency, and intent: Preliminary data collection, in: Intelligent Narrative Technologies Workshop, volume 14, 2024.

[21] M. Fisher, M. Siler, S. G. Ware, Structure, agency, and improvisation in human-led digital interactive narrative exercises, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 21, 2025.

[22] M. Ghallab, A. Howe, D. McDermott, A. Ram, M. Veloso, D. Weld, D. Wilkins, PDDL—the Planning

Domain Definition Language, 1998.
[23] K. K. Yu, M. Guzdial, N. R. Sturtevant, Evaluating the effects of AI directors for quest selection, in: AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, volume 20, 2024, pp. 245–252.
[24] L. J. Martin, B. Harrison, M. O. Riedl, Improvisational computational storytelling in open worlds, in: International Conference on Interactive Digital Storytelling, volume 9, 2016, pp. 73–84.
[25] M. Kreminski, N. Wardrip-Fruin, Sketching a map of the storylets design space, in: International Conference on Interactive Digital Storytelling, volume 11, 2018, pp. 160–164.
[26] J. Porteous, F. Charles, M. Cavazza, NetworkING: Using character relationships for interactive narrative generation, in: International Conference on Autonomous Agents and Multi-Agent Systems, volume 12, 2013, pp. 595–602.
[27] A. Shirvani, S. G. Ware, L. J. Baker, Personality and emotion in strong-story narrative planning, IEEE Transactions on Games 15 (2022) 669–682.
[28] R. Pérez y Pérez, Representing social common-sense knowledge in MEXICA, in: T. Veal, F. A. Cardoso (Eds.), Computational Creativity: The Philosophy and Engineering of Autonomously Creative Systems, Springer, 2019, pp. 255–274.
[29] R. Cardona-Rivera, R. Young, Symbolic plan recognition in interactive narrative environments, in: Intelligent Narrative Technologies Workshop, volume 8, 2015, pp. 16–22.

A. Draft Definitions

This appendix is a tentative listing of the full collection of MicroTales elements; refer back to Section 4 for how they appear in a problem file and how they are used to define gameplay. This version of the definitions is informal and compact to fit the page limits. A more detailed version can be found at https://github.com/sgware/microtales-doc, which will be updated as the project is developed. As shorthand, in definitions of parameterized elements—actions, goals, and variables—assume parameters starting with c are characters, i are item constants, and l are locations.

A.1. Extensions

Table 1 shows the names of the MicroTales extensions and the actions, variables, location types, character types, and goals introduced by each. Some elements are enabled by more than one extension.

Table 1: List of MicroTales extensions.

extension	**enables…**
`Alchemy`	`laboratory` locations, `Flower` items; $brew$ actions
`Commerce`	`market` locations; `merchant` characters; `Coin` items; $rich$ goals; $sell$ actions
`Crafting`	`workshop` locations; `Ore` items; $craft$ actions
`Crime`	`jail` locations; `knight` characters; $criminal$ variables; `Coin` items; $lawless$ and $lawful$ goals; $arrest$, $jailbreak$, and $release$ actions
`Enchantment`	`noble` and `sorcerer` characters; `CharmPotion` items; $charm$ actions
`Forgiveness`	`chapel` locations; `cleric` characters; $forgive$ and $repent$ actions
`Healing`	`cleric` and `sorcerer` characters; `HealPotion` items; $heal$ actions
`Marriage`	`chapel` locations; $spouse$ variables; `Flower` items; wed goals; $marry$ actions
`Monarchy`	`castle` locations; `noble` characters; $monarch$ variables; `Crown` items; `ruling` goals; $enthrone$ actions

`Sickness`	`sorcerer` characters; *health* variables; `CursePotion` items; *harmed* and *unharmed* goals; *curse*, *die*, *loot*, *recover* and *sicken* actions
`Stealth`	`camp` locations; `bandit` and `sorcerer` characters; *visible* variables; `HidePotion` items; *hide* actions
`Teleportation`	`sorcerer` characters; `TeleportPotion` items; *teleport* actions
`Theft`	`bandit` characters; `Coin` and `Sword` items; *rich* goals; *take* actions
`Undead`	`graveyard` locations; `cleric` and `sorcerer` characters; `BanishPotion` and `RaisePotion` items; *banish* and *curse*, *raise*, and *rise* actions
`Violence`	`knight` characters; *health* variables; `Sword` items; *harmed* and *unharmed* goals; *attack*, *die*, *loot*, and *recover* actions

A.2. Item Constants

MicroTales elements cannot be created or destroyed, but items are not first-class elements. Rather, they are a group of constants, special values that can be assigned to some variables. Each character has `left` and `right` hand variables, and the values that can be assigned are the item constants or `Null` to represent holding nothing. This means each character can hold zero to two items.

Table 2 lists the item constants along with a summary of associated uses. Items listed as "potion" in the table can be created by *brew*, and "metal" by *craft*. See Section A.7 for full details about how actions interact with items. All items have extensions as prerequisites; refer back to Section A.1.

Table 2: List of MicroTales items.

item constant	description
`BanishPotion`	Potion. Used by *banish*.
`CharmPotion`	Potion. Used by *charm*.
`Coin`	Metal. Creatable by *sell*. Desired by *rich* goal.
`Crown`	Metal. Used by *enthrone*.
`CursePotion`	Potion. Used by *curse*.
`Flower`	Creatable by *grow*. Used by *brew*.
`HealPotion`	Potion. Used by *heal*.
`HidePotion`	Potion. Used by *hide*.
`Ore`	Metal.
`RaisePotion`	Potion. Used by *raise*.
`Sword`	Metal. Used by *arrest*, *attack*, and *take*.
`TeleportPotion`	Potion. Used by *teleport*.

A.3. Location Types

MicroTales problems take place on a map made of one or more locations. A location has a unique name and a type. A problem must define at least one location, and how locations are connected via paths. Paths are symmetric and are used to determine where characters can *walk*.

Table 3 describes each location type. The `cave`, `crossroads`, `forest`, and `village` types can be used in any problem; the rest are enabled by extensions as listed in Section A.1. Locations typically have actions associated with them; these are elaborated in the Section A.7 action definitions.

Table 3: List of MicroTales location types.

location type	description
`camp`	Used to *hide*. 1–2 paths.
`castle`	Used to *enthrone*. 1–4 paths.
`cave`	Used to *drop* and *pickup*. Starts with `ore` by default. 1–2 paths.

`chapel`	Used to *repent* and *marry*. 1–3 paths.
`crossroads`	Used to *drop* and *pickup*. 2–4 paths.
`forest`	Used to *drop*, *pickup*, and *grow*. 0–4 paths.
`graveyard`	Used to *raise* and *rise*. 0–2 paths.
`jail`	Used to *arrest*, *jailbreak*, and *release*. Has *locked* variable. Exactly 1 path.
`laboratory`	Used to *brew*. 1–2 paths.
`market`	Used to *sell*. 1–4 paths.
`village`	Used to *drop* and *pickup*. 0–4 paths.
`workshop`	Used to *craft*. 1–2 paths.

A.4. Character Types

A character has a unique name and a type. Table 4 defines the character types. Default locations initialize $location(c)$ to the first defined location of that type. Default items initialize $left(c)$ to that item constant. Some character types have special abilities that override the action preconditions in Section A.7. The *peasant* is available in all problems; the rest come from extensions in Section A.1.

Table 4: List of MicroTales character types.

character type	**description**
`bandit`	Starts at `camp` or `crossroads` by default. Can *take* without `Sword` and *hide* without `HidePotion`.
`cleric`	Starts at `chapel` by default. Can *heal* or *banish* without potions. Can *forgive*. *avenged* goal forbidden.
`knight`	Starts at `camp` or `crossroads` with `Sword` by default. Can *attack* without `Sword`.
`merchant`	Starts at `market` or `crossroads` with `Coin` by default. Can't *sell*. *liked* goal forbidden.
`noble`	Starts at `castle` or `chapel` with `Coin` by default. Can *charm* without `CharmPotion`.
`peasant`	Starts at `village` or `workshop` by default.
`sorcerer`	Starts at `laboratory` or `graveyard` by default. Can *brew* without `laboratory`.

A.5. Variables

A variable is identified by a name followed by an ordered list of arguments. Table 5 lists these variables. It describes the parameter types, the values that can be assigned to the variable, and default initial value. If the default value is an item constant that requires an extension and the problem does not enable that extension, the default value is `Null` instead. Some character definitions also override default values. Variables are used to define the preconditions and effects of actions, as well as goal fulfillment. More details are in the definitions for those elements. The *item*, $left$, *location*, *relationship*, and *right* variables exist in all problems; see Section A.1 for the extensions that enable the others.

Table 5: List of MicroTales variables.

variable	**description**
$criminal(c)$	maps character to Boolean, default `False`
$health(c)$	maps character to `Healthy` (default), `Hurt`, `Dead`, or `Ghost`
$item(l)$	maps a `cave`, `crossroads`, `forest`, or `village` to item, default `Null`
$left(c)$	maps character to item, default `Null`
$location(c)$	maps character to location, default by character type

$locked(l)$	maps a `jail` to Boolean, default `False`
$monarch()$	single character, default `Null`
$relationship(c1, c2)$	maps two characters (asymmetric) to `Friend`, `Null` (default), or `Enemy`
$right(c)$	maps character to item, default `Null`
$spouse(c)$	maps character to another character (symmetric)
$visible(c)$	maps character to Boolean, default `True`

A.6. Goals

Table 6 defines the possible goals. The first column is the goal with name and arguments. The second column describes the goal's objective. The $avenged$, at, $disliked$, $liked$, and $obliged$ goals may be added to any problem; the others come from extensions.

There are additional goals that always apply implicitly. First, if `Sickness` or `Violence` is enabled, characters c prefer $health(c) =$ `Healthy` over `Hurt`, `Hurt` over `Dead`, and `Ghost` over `Dead`. Second, if `Undead` is enabled, each c has a goal that behaves as $avenged(c)$ when $health(c) =$ `Ghost`.

Table 6: List of MicroTales goals.

goal	**description**
$at(c, l)$	achieve $location(c) = l$
$avenged(c)$	minimize number of c' such that $relationship(c, c') =$ `Enemy` and $health(c') \neq$ `Dead`
$disliked(c)$	maximize number of c such that $relationship(c', c) =$ `Enemy`
$harmed(c)$	achieve $health(c) \neq$ `Healthy`
$lawful()$	minimize number of c such that $criminal(c)$, $health(c) =$ `Healthy` or `Hurt`, and $loc(c)$ is not a `jail`
$lawless()$	maximize number of c such that $criminal(c)$, $health(c) =$ `Healthy` or `Hurt`, and $loc(c)$ is not a `jail`
$liked(c)$	maximize number of c' such that $relationship(c', c) =$ `Friend`
$obliged(c)$	maximize number of c' such that $relationship(c, c') =$ `Friend` and $health(c') \neq$ `Dead`
$rich(c)$	maximize `Coin` values among $left(c)$ and $right(c)$
$ruling(c)$	achieve $monarch() = c$ or $monarch() = spouse(c)$
$unharmed(c)$	achieve $health(c) =$ `Healthy`
$wed(c1, c2)$	achieve $spouse(c1) = c2$

A.7. Actions

Actions have a name and arguments. Table 7 lists the actions and their preconditions and effects. We use shorthands as follows.

"Must" implies a precondition; other statements imply effects unless otherwise noted. For a character c to have an item i means $left(c) = i \vee right(c) = i$. To lose i means there is an effect of $left$ or $right$ no longer being i. To expend i must have and loses it. Available space means $left(c) =$ `Null` $\vee$ `right(c)` $=$ `Null`. To gain i means the $left$ value becomes i if it was `Null`, otherwise the $right$ value becomes i. Given a location l rather than a character, these terms apply similarly to $item(l)$.

We say a character or location "is" (for preconditions) or "becomes" (for effects) a constant, e.g., `Healthy`, when the relevant variable, e.g., $health(c)$, takes that value. We say c is "at" (precondition) or "moved to" (effect) l when $location(c) = l$. To raise or lower c's health is to change $health(c)$ by one stage along the ascending ranking `Dead`, `Hurt`, `Healthy` if possible, no change otherwise. Likewise, the "relationship of c to c'" raises or lowers $relationship(c, c')$ along `Enemy`, `Null`, `Friend`.

If a variable would be checked or modified by an action while not enabled by an extension, that part of the action description is ignored. Some character abilities override preconditions; see Section A.4.

Unless otherwise specified, assume actions require characters to be alive; actions with two characters require their $location$ to be the same, and the second character to be $visible$.

Table 7: List of MicroTales actions.

action	**description**
$arrest(c1, c2, l)$	$c1$ must have `Sword`. Moves $c2$ to `jail` l. l becomes locked. Lowers $c2$'s relationship to $c1$. $c1$ becomes criminal if $c2$ not criminal.
$attack(c1, c2)$	$c1$ must have `Sword`. Lowers $c2$ health and relationship to $c1$. $c1$ becomes criminal if $c2$ not criminal.
$banish(c1, c2)$	$c2$ must be `Ghost`. $c1$ expends `BanishPotion`. $c2$ becomes `Dead`. Lowers $c2$'s relationship to $c1$.
$brew(c, i)$	c must be at `laboratory`. i must be potion. c expends `Flower`, c gains i.
$charm(c1, c2)$	$c1$ expends `CharmPotion`. Raises $c2$'s relationship to $c1$.
$craft(c, i1, i2)$	c must be at `workshop`. $i1$ and $i2$ must be metal. c expends $i1$, c gains $i2$.
$curse(c1, c2)$	$c1$ must be `Ghost` or expends `CursePotion`. Lowers $c2$ health and relationship to $c1$. $c1$ becomes criminal if $c2$ not criminal.
$die(c)$	c must be `Hurt`. c becomes `Dead`.
$drop(c, i)$	c must be at `cave`, `crossroads`, `forest`, or `village`. c expends i. c's location loses item if any, gains i.
$enthrone(c)$	c must have `Crown` and be at `castle`. c becomes monarch.
$forgive(c1, c2)$	$c1$ must be `cleric`. $c2$ stops being criminal.
$give(c1, i, c2)$	$c2$ must have available space. $c1$ expends i. $c2$ gains i.
$grow(l)$	l must be `forest` l and have available space. l gains `Flower`.
$heal(c1, c2)$	$c1$ expends `HealPotion`. Raises $c2$'s health and relationship to $c1$.
$hide(c)$	c must be visible and at `camp`. c stops being visible.
$jailbreak(l)$	l must be locked `jail`. l stops being locked.
$loot(c1, i, c2)$	$c1$ must have available space. $c2$ must be `Dead`. $c2$ expends i. $c1$ gains i.
$marry(c1, c2)$	$c1$ and $c2$ must not have spouse. $c1$ and $c2$ become each other's spouse.
$pickup(c)$	c must have available space. c's location l must be `cave`, `crossroads`, `forest`, or `village`. l expends item i. c gains i.
$raise(c1, c2, l)$	$c1$'s location must be $c2$'s location or l. $c2$ must be `Dead`. l must be `graveyard`. $c1$ expends `RaisePotion`. $c2$ becomes `Ghost` l and loses all items. Moves $c2$ to l.
$recover(c)$	c must be `Hurt`. c becomes `Healthy`.
$release(c, l)$	l must be locked `jail`. c's location must have path to l. l becomes unlocked. c becomes criminal if character at l is criminal. Raises relationship of characters at l to c.
$repent(c)$	c must be at `chapel`. c stops being criminal.
$rise(c, l)$	c must be `Dead`. l must be `graveyard`. c becomes `Ghost` and loses all items. Moves c to l.
$sell(c, i)$	c must be at `market`. c expends i, gains `Coin`.
$sicken(c)$	c must be `Healthy`. c becomes `Hurt`.
$take(c1, i, c2)$	$c1$ must have `Sword` and available space. $c2$ expends i. $c1$ gains i. $c1$ becomes criminal if $c2$ not criminal. Lowers $c2$'s relationship to $c1$.
$teleport(c, l)$	c expends `TeleportPotion`. Moves c to l.
$trade(c1, i1, c2, i2)$	$c1$ expends $i1$, gains $i2$. $c2$ expends $i2$, gains $i1$.
$walk(c, l)$	c must be `Healthy` or `Ghost` at location with path to l. Moves c to l.