# Planning Stories Neurally

Rachelyn Farrell[1] and Stephen G Ware[1]

[1]University of Kentucky

March 19, 2024



**Abstract**

Symbolic planning algorithms and large language models have different strengths and weaknesses for story generation, suggesting hybrid models might leverage advantages from both. Others have proposed using a language model in combination with a partial order planning style algorithm to avoid the need for a handwritten symbolic domain of actions. This paper offers a complementary approach. We use a state space planning algorithm to plan coherent multi-agent stories in symbolic domains, with a language model acting as a guide to estimate which events are worth exploring first. We evaluate an initial implementation of this method on a set of benchmark problems and find that the LLM's guidance is helpful to the planner in most domains.

# Planning Stories Neurally

Rachelyn Farrell and Stephen G. Ware
University of Kentucky

***Abstract*—Symbolic planning algorithms and large language models have different strengths and weaknesses for story generation, suggesting hybrid models might leverage advantages from both. Others have proposed using a language model in combination with a partial order planning style algorithm to avoid the need for a hand-written symbolic domain of actions. This paper offers a complementary approach. We use a state space planning algorithm to plan coherent multi-agent stories in symbolic domains, with a language model acting as a guide to estimate which events are worth exploring first. We evaluate an initial implementation of this method on a set of benchmark problems and find that the LLM's guidance is helpful to the planner in most domains.**

***Index Terms*—narrative planning, story generation, language models**

## I. Introduction and Related Work

Interactive narratives play an important role in many virtual environments for entertainment and education. Ensuring the narrative remains believable and well-structured as it adapts to player input is a challenge that artificial intelligence and machine learning techniques can help to address.

Early interactive narrative systems, like *Tale-Spin* [1], *Universe* [2], and *Façade* [3], used symbolic models of their story domains. Characters, places, objects, and states are typically represented in some form of predicate logic. Events have preconditions which must be true before they can happen and effects which change the narrative state. These systems are primarily reactive, meaning they choose and commit to a single action based on the narrative so far with minimal look-ahead. The reactive approach scales well, and continues to be a popular choice for modern interactive narratives like *Prom Week* [4], *Versu* [5], *Why Are We Like This?* [6], and many others.

There has also been considerable research on using AI planning to ensure the structure of interactive stories. Plans and planning offer a rich, formal, generative, symbolic model of storytelling [7] that also uses predicate logic and events with preconditions and effects. Planning algorithms have been used to create many interactive narratives [8]–[11]. There has also been considerable research on augmenting planning algorithms with models of story structure [12]–[14]. Planning-based systems differ from reactive systems by performing an extensive backtracking search over the space of all possible stories. This grants them a greater ability to ensure story quality and structure, but it comes at the cost of scalability. The space of all possible stories in a domain quickly becomes too large to explore efficiently, limiting the applicability of planning to large story worlds and prompting research into fast narrative planning [15], [16].

With the recent explosion of research on deep learning and large neural language models (LLMs), there have also been many applications of language generators to storytelling. One example among many is *AI Dungeon* [17], which mediates the player's natural language input to GPT-2 [18] to mimic role-playing games. LLMs are generally used reactively, in the sense that they do little or no search through the space of possible stories, but generate and commit to the next word of text based on the current context. While training LLMs is expensive, text generation is relatively cheap. Unlike symbolic reactive storytelling systems, LLMs can draw from a large training set of characters, places, objects, and events without requiring the user to define them by hand in advance. However, while generated text can be coherent within a sliding window, it is difficult to maintain global coherence or to generate a goal-directed story with a language model alone [19]. Several researches have worked to address these limitations with respect to story generation [20]–[23].

One interesting method of achieving global story coherence was proposed in *Neural Story Planning* [24]. It mimics a partial order planning algorithm [25] that works backwards from the end of the story. For the story goal, and for each event, the system determines the preconditions that must be met. For each precondition, it finds an existing event in the story or adds a new event to the story which has an effect that satisfies that precondition. The resulting story is a causal network of events that improves coherence [26]. Its key distinction from a traditional partial order planner is that, rather than relying on a hand-authored symbolic domain of events, the system prompts an LLM to determine preconditions and to generate events which can satisfy them. The approach thus combines the coherence of a planner with the on-demand content generation of an LLM.

However, content generation is a double-edged sword. The ability of LLMs to introduce new story elements can be an advantage or a liability depending on the application. When stories are told in text, as with *AI Dungeon*, there is little cost to introducing new elements. However, many interactive narratives take place in 3D virtual game-like environments where art assets, animations, and dialog are fixed. The ability of an LLM to introduce a dragon into a 3D game's story, however fitting it might be, is useless if the game does not have a rigged and animated dragon model. There is preliminary evidence that deep learning might eventually generate these art assets on demand as well [27], but this technology is in its infancy and raises significant ethical questions about attribution [28]. Regardless, some authors might want to limit a storytelling system to a fixed set of assets for aesthetic purposes, and commercial virtual environments may impose these limits to ease quality assurance.

We are interested in an approach complementary to *Neural Story Planning* that we call *Planning Stories Neurally*: Using

a narrative planner with a hand-written symbolic domain, we want to leverage the general knowledge and storytelling capabilities of an LLM to guide the planner toward solutions more efficiently. There are many ways we might achieve this. For a preliminary analysis of this approach, we define an LLM-based cost function for a uniform cost search. The planner ranks available actions at each step by how closely they match what the LLM says should happen next, given a description of the story so far. We evaluate in several different story domains whether the LLM's suggestions are helpful to the planner in two key ways: leading the planner towards the goal, and using actions that make sense from the perspectives of the characters who take them.

## II. Background

In forward state-space planning, actions are added onto the ends of plans until a goal state is reached. Classically, a plan that reaches a goal state is called a solution. Narrative planners however impose additional constraints on solutions to ensure that they meet certain desired narrative criteria, such as for characters to act believably. Toward this end, a planner can define *characters* as special kinds of entities that have goals and can only participate in actions that contribute to achieving their goals. We use the Sabre narrative planner [29], which additionally models character beliefs, and ensures that the plans or *explanations* used to justify a character's actions are consistent with their beliefs under limited observability.

For Sabre, a solution is a plan that 1) reaches a goal state and 2) contains only actions that are *explained*. A full definition of action explainability in Sabre is given in the above cited paper, but to summarize: An action is explained in general when it is explained for each of its consenting characters—the characters who are willing participants in the action. An action is explained for a character when it is the next step in some plan that achieves that character's goals, according to their beliefs. The existence of such a plan indicates that the character will appear to be taking the action for a reason, thus the action is explained for them. The planner tracks this information during search by identifying when character goals are achieved and propagating explanations backward, marking that character's prior actions as explained if they contributed to the goal achievement.

To plan efficiently in large spaces, a planner needs to consider possible actions in an intelligent order, prioritizing those that are likely to lead to a solution. Classical planning search heuristics estimate how close a given state is to a goal state, and can be used to search classical planning problems efficiently. But for narrative planning problems, reaching the goal state is only one of the solution requirements, the other being that actions are explained. A good narrative planning heuristic, in addition to estimating the distance to the goal, needs to estimate whether characters are acting believably (at least according to the planner's constraints). The Glaive heuristic [15] does this by estimating distance to character goals in addition to the story goal. However, it assumes full observability, since Glaive does not model character beliefs, and therefore does not perform well when Sabre's limited observability constraints are applied.

Sabre's belief model and narrative planning in general pose interesting challenges for search, and work toward more efficient search strategies is ongoing [31]. We ultimately envision an LLM-based heuristic used in combination with an intelligent search algorithm, but at this stage we are not attempting to implement a highly efficient search. Instead we focus on evaluating whether an LLM has the capabilities needed to be successful in a heuristic, and in what kinds of domains. We define a cost function that uses LLM suggestions, and compare it to a baseline cost function in a uniform-cost search, because this provides the data to demonstrate how helpful the suggestions are throughout a large part of the problem space.

We aim to see whether an LLM, given everything the planner knows about the story at a given point in a search, can provide good suggestions for what should happen next. A good suggestion, from the planner's point of view, is an action that can lead to a goal state and can be explained for its characters. In the sections below, we define a cost function that ranks actions using suggestions from an LLM, and then examine in multiple story domains how effective the suggestions are in terms of goal achievement (the number of nodes visited to reach the first solution) and explainedness (the ratio of explained to unexplained nodes visited during search). We begin by summarizing the formal definitions of a Sabre problem [29] below.

### *Problem Definition*

A Sabre problem is defined as $\langle C, F, A, T, U, s_0 \rangle$, where:

- $C$ is a set of characters: special entities which should appear to have beliefs and intentions.
- $F$ is a set of state fluents: Boolean, nominal, or numeric properties whose values can change over time. Sabre defines a host of logical literals and operations over these fluents, including nested epistemic fluents representing complex character beliefs.
- $A$ is a set of actions: STRIPS-like [30] operators that specify preconditions for when they can occur and effects that change the world state afterward. Sabre actions also define a set of consenting characters—those who are participating in the action intentionally—and an observation function to determine which characters are aware of the action when it occurs.
- $T$ is a set of triggers, which are like actions except that they happen automatically. Whenever their preconditions are satisfied in a state, their effects apply immediately. Triggers do not have consenting characters or an observation function.
- $U$ is a set of utility functions: expressions that define preferences over states. This includes the *author utility function*, which represents the states the planner should attempt to reach (the story goal), and one utility function for each character to represent their goals.
- $s_0$ is the initial state, an assignment of a starting value for every fluent in $F$ and any wrong beliefs that characters initially hold.

Given a problem so defined, the planner's task is to find a plan, or sequence of actions, that transforms the world from the

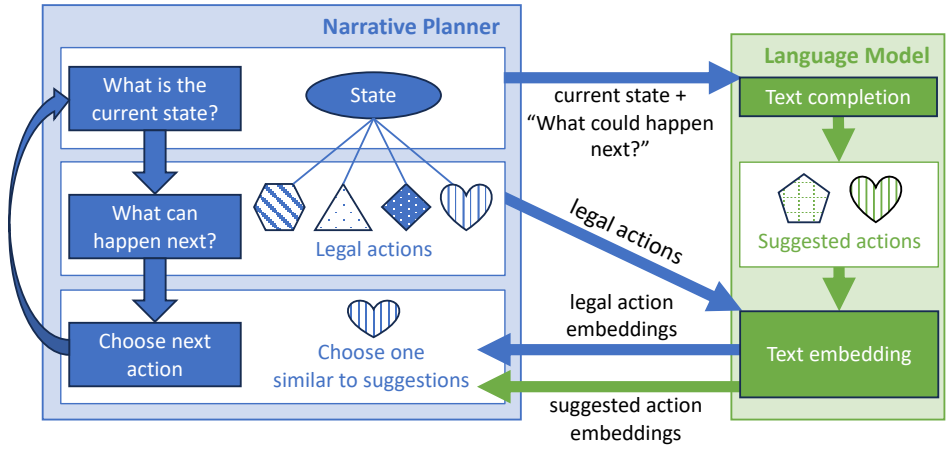


Fig. 1. The narrative planner assigns costs to available actions based on suggestions generated with a large language model.

initial state into one in which 1) the author utility is improved (the story goal achieved) and 2) all actions are explained for their consenting characters.

## III. Method

Our search process is illustrated in Figure 1. At each step in the search, the planner describes the current state to a language model, prompts it for what should happen next, and then prioritizes actions that are more similar to the LLM's suggestions.

### A. Describing the Current State

Each time the planner visits a node (a unique plan or story in progress, starting with the empty plan), it begins by describing the current state of the story world to an LLM and asking it what the next step might be. In Figure 1, the top rectangle on the left side represents this phase.

To describe the story's current state in natural language, the system must translate domain elements into natural language sentences. We created a simple handwritten grammar for each of the domains in Table I to accomplish this. For example, in the Grandma domain, the fluent assignment $location(Hero) = Cottage$ is converted into the sentence *"The hero is at the cottage."*, and the action $walk(Hero, Cottage, Crossroads)$ into *"The hero walks to the crossroads."*

Using these grammars, the planner builds a description of the current state that comprises each of the elements listed below. (The full text of this example is given in the Appendix.)

1) The initial state: "The hero is alive. The merchant is alive. The guard is alive..."
2) Each character's goals: "The hero wants to bring the medicine to the cottage. The bandit wants to collect valuable items..."
3) The names of the actions in the domain: "Characters can walk, buy, loot, attack, rob, report, and take."
4) The story goal: "The story must end with the hero either being attacked or having the medicine at the cottage."
5) The actions in the plan so far (unless this is the first): "The story begins with the following steps: The merchant walks from the market to the crossroads." (page 2 of Appendix)

We combine this description with a prompt to suggest followup actions, and send it to the LLM (first arrow in Figure 1).

### B. Prompting for Suggestions

For the experiments discussed in this paper we used the prompt text: "List $n$ different actions that could happen next. State each as a short sentence on its own line." It is important that we ask for more than one suggestion ($n > 1$) because when the LLM is instructed to give only one suggestion, it usually suggests something that would lead to a favorable outcome for the protagonist. This has the result of always guiding the planner in the same direction, and branches of the space that are important get overlooked. Asking for two different suggestions allows the LLM to suggest both good and bad options. We use $n = 2$ as the default; we also tested $n = \{3, 4\}$ in three example domains and did not observe any significant difference in performance compared to $n = 2$ (see Section IV-B).

For our text completion model we used OpenAI's *gpt-3.5-turbo* (May 2023). We used a temperature of 0.3, and limited responses to 25 tokens by default (increased when $n > 2$). We did not fine-tune the model, but since our prompt ends with a list of prior actions in the format we hope to generate, this is essentially few-shot prompting (for all but the first node in the search, when there are no prior events in the story). We did not test our method with other models, although we hope to do so in the future. We acknowledge that this will surely involve some prompt re-engineering, as will attempting to use any more recent version of *gpt-3.5-turbo*.

The response from the LLM is parsed into $n$ suggestion strings. For Step 1 of the Grandma Lose example (Appendix), the LLM's two suggestions are "The hero walks to the market" and "The merchant buys the medicine".

### C. Comparing Actions

At this point the planner identifies all the actions that are actually possible in the current state, translates them into natural language sentences using the grammars discussed above, and sends each sentence to an LLM to get its embedding vector (the second and third arrows in Figure 1). For all text embeddings we used OpenAI's *text-embedding-ada-002* model. This was the only model for which OpenAI provided access to text embeddings at the time.

The final step is to compare the text embeddings of the legal actions with those of the suggested actions (the fourth arrow). The actions that are good matches to any of the LLM's suggestions are the ones we want the planner to visit first. We therefore assign lower costs to actions that are similar to a suggestion, and higher costs to those that are not.

Given the embeddings of the legal actions and suggestions, the planner calculates, for each action, a distance value that measures its proximity to the suggestion closest to it. Let $a$ be an action and $S$ a set of suggestions from the LLM output. The minimum distance between $a$ and $S$ is calculated as:

$$\textsc{dist}(a, S) \leftarrow \min_{s \in S} \left(1 - \frac{1 + \textsc{CoSim}(\textsc{Em}(a.\textsc{nl}), \textsc{Em}(s))}{2}\right)$$

where $a.\textsc{nl}$ is the natural language translation of the action $a$ (discussed in Section III-A) and the function $\textsc{em}()$ returns the text embedding of the given string. The cosine similarity $\textsc{CoSim}()$ between each pair of embeddings is inverted and scaled between 0 and 1 to avoid negative costs, and the smallest distance value is returned. During search (Algorithm 1) this value is calculated for each of the possible actions in a given state and actions with smaller minimum distances are assigned lower step costs, causing the planner to prioritize them.

Continuing with our previous example (Appendix), there are 12 legal actions, including some that are explained and necessary for most solutions (e.g. The hero walks from the cottage to the crossroads), some that would lead to an ending but are never going to be explained (e.g. The hero attacks himself), some that can be explained but are irrelevant for the story (e.g. The merchant reports seeing the bandit at the camp to the guard) and many in between. As it happens, neither of the suggested actions is directly possible at this moment: The hero cannot walk straight to the market without first walking to the crossroads, and the merchant cannot buy the medicine because she already has it.[1]

Since none of the possible actions are a perfect match, none will be given a cost of 0. In this case, the lowest cost is 4, which is appropriately assigned to the very important action, "The hero walks from the cottage to the crossroads", due to its similarity to the suggestion "The hero walks to the market". Another action which is also helpful (it is explained and leads to some solutions) is tied for the same cost: "The merchant walks from the market to the crossroads", based on its similarity to "The merchant buys the medicine". So even though the LLM's suggestions are not directly applicable in this state, the planner does its best to follow the lead: The hero walks somewhere, the merchant does something, etc. The unhelpful actions mentioned above are assigned higher costs (8 and 9) and will not be considered until much later in the search, if at all.

[1]Informally, we noticed that the LLM often gives poor suggestions for the initial call (when the plan is empty), as this example illustrates. Having at least one step in the plan helps by prompting the LLM with some immediately relevant context and with the expected format for actions; so its suggestions for non-first steps tend to be more appropriate.

### D. Search

We now define an LLM-guided uniform cost search algorithm, which we label LLM-UCS. Uniform cost search (UCS) always visits the plan with the lowest cumulative action cost. Breadth-first search is an example of UCS in which all actions cost 1, so the cost of a plan is always equal to the plan's length. In LLM-UCS, the cost of a plan is the sum of the LLM-based costs of each action in the plan. Thus, for example, a 5-step plan which adheres closely to the LLM's suggestions may be visited sooner than a 3-step plan which does not.

**Algorithm 1** LLM-Guided Uniform Cost Search

1: Let $d$ be a depth limit, and $z$ a precision constant (100 or 1,000).
2: Let $\textsc{state}(\pi)$ denote the state reached by executing the plan $\pi$ starting from the initial state of $p$.
3: **function** LLM-UCS(problem $p$) ▷ Returns a solution, or failure
4: $\quad\pi \leftarrow$ an empty plan (sequence of actions)
5: $\quad q \leftarrow$ an empty min-priority queue of plans
6: $\quad q.\textsc{push}(\pi, \text{ cost} = 0)$
7: $\quad$**loop**
8: $\quad\quad\pi \leftarrow q.\textsc{pop}()$
9: $\quad\quad$**if** $\textsc{state}(\pi) \models p.\text{goal}$ **and** $\pi$ is explained **then**
10: $\quad\quad\quad$**return** $\pi$
11: $\quad\quad$**else if** $\textsc{length}(\pi) + 1 \leq d$ **then**
12: $\quad\quad\quad S \leftarrow \textsc{promptLLM}(p, \pi)$
13: $\quad\quad\quad$**for all** $a \in A$ **do**
14: $\quad\quad\quad\quad$**if** $\textsc{state}(\pi) \models a.\text{precondition}$ **then**
15: $\quad\quad\quad\quad\quad cost(a) \leftarrow \textsc{round}\big(\textsc{dist}(a, S) * z\big)$
16: $\quad\quad\quad\quad\quad q.\textsc{push}\big(\pi + a, cost(\pi) + cost(a)\big)$
17: $\quad\quad$**if** $q.\textsc{empty}()$ **then return** failure

The search process is formalized in Algorithm 1. Before search begins, the planner grounds and simplifies the problem. As a preprocessing step, we compute text embeddings for all the grounded actions in the domain and store them in a hash table along with the embeddings for all the suggestions received from the LLM.

Beginning with an empty plan that costs 0 (line 6): If the plan achieves the story goal and is explained, it is returned as a solution (line 10). Here, $p.\text{goal}$ is a disjunction of the author's goals—conditions in the author utility expression that evaluate to a higher utility than the initial state. If the plan is not a solution, the system prompts the LLM for suggestions about what should happen next (line 12) using the prompt text described earlier. Each action that is possible in the current

state (line 14) is then assigned a cost according to its minimum distance with respect to the suggestions (line 15). Each new plan ($\pi + a$) is added onto the search queue, with the cost of the new action being added to the cost of the plan (line 16). Search continues with the next lowest-cost plan in the queue, until either a solution is returned (success), or the queue is emptied (failure).

Normally, Sabre evaluates whether a node is explained (line 9) by checking if an explanation for that node has been found in the current search. However, this search itself would not necessarily find all the explanations needed, since it is only considering actions that are possible (line 14). A full epistemic search would also have to visit nodes that characters believed were possible, even if they weren't.[2]

To avoid missing crucial explanations, we instead evaluate the check on line 9 using a lookup table of pre-recorded explained nodes. These were logged during a separate, complete breadth-first search of each problem in our test suite (Table I) using the given epistemic limit, and with both depth limits increased by one in an effort to mark as many explained nodes as possible (+1 $atl$ and +1 $ctl$).

| Problem | Domain Size | | | | Configurations | | | |
|---|---|---|---|---|---|---|---|---|
| | $\|C\|$ | $\|F\|$ | $\|A\|$ | $\|T\|$ | $u$ | $atl$ | $ctl$ | $el$ |
| Basketball | 4 | 93 | 168 | 192 | 1 | 8 | 5 | 3 |
| Bribery | 3 | 25 | 27 | 0 | 1 | 2 | 2 | 1 |
| Deer Hunter | 3 | 42 | 28 | 76 | 1 | 10 | 5 | 1 |
| Fantasy | 4 | 85 | 76 | 141 | 2 | 9 | 3 | 3 |
| Grandma Lose | 4 | 96 | 800 | 952 | 1 | 5 | 4 | 2 |
| Grandma Win | 4 | 96 | 800 | 952 | 2 | 5 | 4 | 2 |
| Hospital | 4 | 67 | 102 | 196 | 1 | 10 | 6 | 3 |
| Jailbreak Lose | 3 | 46 | 106 | 54 | 1 | 7 | 7 | 2 |
| Jailbreak Win | 3 | 46 | 106 | 54 | 2 | 7 | 7 | 2 |
| Raiders | 3 | 29 | 35 | 66 | 1 | 7 | 4 | 2 |
| Secret Agent | 2 | 15 | 44 | 75 | 1 | 8 | 8 | 1 |
| Snakebite | 4 | 99 | 352 | 637 | 1 | 8 | 5 | 2 |
| Space Lose | 2 | 26 | 29 | 66 | 1 | 5 | 3 | 2 |
| Space Win | 2 | 26 | 29 | 66 | 4 | 5 | 3 | 2 |
| Treasure | 2 | 9 | 5 | 0 | 1 | 4 | 4 | 3 |

TABLE I
SABRE BENCHMARK PROBLEMS

A collection of benchmark problems for the Sabre narrative planner [32]

## IV. EVALUATION

We evaluate our method on a set of narrative planning benchmark problems collected from a variety of sources and adapted to Sabre's format [32]. Table I lists each problem along with its size information (characters, fluents, actions, and triggers after grounding) and the search configurations we used for it. The configurations were chosen manually for each problem based on its known solutions, as specified in the technical report.

The utility $u$ sets the problem goal. Some problems have multiple instances, such as Grandma Lose and Grandma Win, having different criteria for the ending. For example, in the "Win" variant of Grandma ($u = 2$), the hero must achieve his goal, whereas in "Lose" ($u = 1$), he can be thwarted which is a shorter plan. The author and character temporal limits, $atl$ and $ctl$, limit the length of plans that can be considered for the author and characters, respectively. Finally, the epistemic limit $el$ limits the depth of epistemic nesting.

We searched each problem using LLM-UCS, stopping either when the first solution is found or when the maximum node limit is reached (5,000 nodes), recording all nodes visited. We compare both the number of visited nodes, and the percent of these nodes that are explained, to their respective baselines: the node count and explained ratio of a complete breadth-first search of the same problem space. (Baseline searches were limited to 1 million nodes.) If the suggestions are generally helpful for a given problem, LLM-UCS will visit significantly fewer nodes on its way to the solution, and these will have a higher explained ratio than the full space.

### A. Results

Only two problems were affected by the node limits. Both planners reached their limits before solving Snakebite, and LLM-UCS reached its 5,000-node limit on Jailbreak Win (though a solution exists at 186,707 nodes). We omit these problems from our analysis. All other problems were solved by both planners, but three were too small to show interesting results: Bribery, Space Lose, and Treasure all had solutions within 40 nodes, and LLM-UCS found it without significant variation from the baseline on either metric. Table II shows the results for the remaining 10 problems.

| Problem | Nodes | Explained | Visited | Vis. Exp. |
|---|---|---|---|---|
| Deer Hunter | 163,109 | 10% | 2,300 ↓ | 19% ↑ |
| Grandma Win | 95,894 | 39% | 734 ↓ | 68% ↑ |
| Hospital | 56,105 | 52% | 532 ↓ | 75% ↑ |
| Fantasy | 20,307 | 66% | 640 ↓ | 81% ↑ |
| *Jailbreak Lose* | *1,349* | *35%* | *2,851 ↑* | *21% ↓* |
| Grandma Lose | 1,056 | 38% | 37 ↓ | 68% ↑ |
| Raiders | 1,051 | 37% | 141 ↓ | 39% ↑ |
| Basketball | 525 | 81% | 31 ↓ | 90% ↑ |
| *Space Win* | *327* | *65%* | *445 ↑* | *59% ↓* |
| Secret Agent | 308 | 44% | 60 ↓ | 25% ↓ |

TABLE II
SEARCH RESULTS FOR LLM-UCS ON BENCHMARK PROBLEMS

*Nodes* is the number of nodes in the space for the given problem before the first solution (i.e. breadth-first search), and *Explained* is the percent of these nodes that are explained. *Visited* is the number of nodes LLM-UCS visited before finding the solution, and *Vis. Exp.* is the percent of the visited nodes that are explained. Arrows indicate the direction of change from the baseline.

Overall the results were positive: For 8 out of 10 problems, the LLM-guided cost function significantly reduced the num-

[2]Characters who have incorrect beliefs can make plans which are actually impossible, but which the planner still needs to find in order to explain their actions. In Section V we discuss how our method can be extended to the epistemic parts of the search as well, but this was not necessary to obtain useful results from the current evaluation.

ber of nodes visited by uniform-cost search; and for 7 out of 10 problems it visited explained nodes at a higher rate than their base frequency, although the extent of this increase varied.

For all but one problem, the directions of change in explained ratio and number of nodes visited were inversely correlated, as we expected. However this was not the case for Secret Agent, in which LLM-UCS visited explained nodes at a lower rate (25%) than they exist in the space (44%), and yet still reached the solution in fewer nodes (60 from 308). Secret Agent is a single-agent pathfinding problem with very few narrative properties; perhaps in this domain, visiting explained nodes is less important than advancing the goal, so the LLM's suggestions were still helpful despite causing many unexplained nodes to be visited.

Jailbreak Lose and Space Win were negative results in both comparisons. For these problems, the LLM cost function had the opposite effect from what we wanted. It caused the planner to waste effort visiting unexplained nodes, and ultimately found the solution later than the breadth-first search. We will focus on these domains for future testing, but for now can only speculate as to why the current methodology failed in these cases. We will return to this discussion in Section V.

Barring these exceptions, it appears that LLMs do possess the basic capabilities we are looking for—they can guide the planner toward solutions, prioritizing explained nodes, in a variety of story domains. This should not be taken too generally. More experiments are needed to show whether this methodology works similarly using different types of LLMs or with different prompts or model parameters. More work could also be done to improve the methodology in terms of prioritizing explained nodes to a greater degree. This may be achievable through simple prompt engineering. For example, if the context were organized by character, which could be done automatically, would this help the LLM frame their individual perspectives and produce more believable suggestions?

### B. Parameters

The method we defined in Section III takes two parameters, $n$ (the number of suggestions to ask for) and $z$ (the precision constant). The precision constant determines how many decimal places we preserve of the cosine distance values when rounding to assign action costs. Consider an example: The set of distance values between each possible action and each suggestion, for one step of an example search, range from $0.032$ to $0.108$. By default we preserve two decimal places ($z = 100$), so in this case we assign costs ranging from 3 to 11. Preserving only one decimal place would reduce precision unhelpfully (costs could only be 0 or 1 in this case). Preserving three decimal places ($z = 1000$) would yield a much wider range of costs (32...108) and thus a more precise ordering.

We compared $n = [2, 3, 4]$ and $z = [100, 1000]$ using 10 runs of each of three problems: Grandma Lose, Secret Agent, and Raiders. Neither of the parameters had a significant impact on the mean number of nodes visited or the mean percent of explained nodes for any of these problems (using T-Tests). We did however observe a greater variance between the 10 different runs (using F-Tests) in the Grandma Lose problem. The variance in the number of nodes visited was affected by both the number of suggestions ($p < .001$) and the precision ($p = .0089$); while the variance in the percent of explained nodes was affected by just the number of suggestions ($p = .0039$). No variances were affected in the other two problems.

### C. Limitations

The difference in performance across the various problems is likely due to how well the different types of stories are represented in the LLM's training data. An important distinction between symbolic story planners and language-based story generators is that planners treat words as arbitrary symbols and are agnostic to their meaning, but word choice has significant implications for language-based methods. As a stress-test we converted the Grandma Lose problem into an unintuitive and confusing domain with no clear genre or setting, by switching the names of all the characters and renaming the items and places. For example the protagonist's goal, instead of "The hero wants to have the medicine at the cottage", is now stated as "The bandit wants to have the dirt at the basketball court", while all underlying logic remains the same.

We ran the LLM-UCS search on both the original Grandma Lose problem and this modified version, for 10 runs each, and compared their results. (BFS solves both problems the exact same way.) As we expected, performance decreased drastically as a result of this modification: On average, LLM-UCS solved the original problem in 28 nodes of which 75% were explained; but on the modified problem it took an average of 72 nodes with only 33% explained—lower than the baseline of 38%. The LLM's suggestions were significantly less helpful for the version of the problem that subverts common knowledge and genre expectations. This suggests that the method may be of limited use for certain kinds of domains, and underscores the importance of word choice in the domain translation text.

The method also relies on a text-based similarity metric, which does not always agree with semantic similarity in the context of the story. For example, sentences A: "The guard attacks the bandit" and B: "The guard attacks the hero" have similar wording, but very different implications for the story. Conversely, sentences C: "The hero runs away from the merchant" and D: "The hero walks to the crossroads" may describe the same action in the story, even though they are worded very differently. This sometimes results in poor action-suggestion matching. Future improvements to the similarity measurements or matching algorithm may yield better results.

## V. Discussion

Symbolic story generation methods give creators fine-grained control over what types of content can be included in generated stories. Planning algorithms can reason over large story spaces and make stronger quality assurances than reactive methods which generate content based solely on the current state. However, planning is expensive; narrative planners either struggle to maintain character believability or they use costly methods that do not scale as well as reactive systems. In this work we seek to leverage the capabilities

of LLMs to make narrative planning more efficient while preserving the advantages of symbolic approaches.

Toward our ultimate aim of incorporating LLM guidance into efficient search heuristics, we present and evaluate a simple action cost function that uses suggestions from LLM queries to guide a narrative planner. For most of the benchmark problems, the LLM's suggestions were successful in helping the planner prioritize believable actions and reach the solution much earlier. This demonstrates that LLMs are capable of estimating simultaneously whether an action leads toward a goal and whether it is believable, two core capabilities needed for a narrative planning heuristic.

The method did not work in two domains, Jailbreak and Space. These will be useful test cases moving forward, and a deeper analysis of the transcripts from their searches may provide more insight into what happened. The failure may stem from something structural about the domains. For example, Jailbreak has a large branching factor compared to other problems, meaning the planner always has a lot more actions to choose from at any state. It is possible that this affected performance in some way, although initial review of the transcript does not indicate so. There are no similarly obvious structural differences in Space compared to the rest of the domains.

The failures could also be due to the wording used in our domain translations. This seems to be the more likely culprit in the case of Jailbreak. One of its actions has a much longer translation than the rest, and this happens to be the first step in the solution we are looking for.[3] Since the prior steps in a story are included in the prompt, they act as few-shot prompting, causing the LLM to match their format in its response. If one of the actions in the history has an extra long translation, the LLM may attempt to give lengthier suggestions, which is bad in two ways. Longer sentences add noise to all the similarity measurements and are harder to match accurately. Additionally, producing a long first suggestion can cause the LLM to run out of tokens mid-sentence on the second suggestion. The transcript confirms that this happened on at least several occasions. The offending action is required for the solution, so this issue may have systematically derailed the search.

At this stage we have applied the technique only in the "real" world (actions whose preconditions are satisfied), but it could be extended into the epistemic parts of the search space as well (actions whose preconditions are not satisfied, but some character believes that they are). Specifically, whenever the planner is considering a character's plan, we can reframe the prompt according to that character's perspective: Rather than describing the true initial state, the story goal, and all the events that have happened so far, describe the current state *according to that character's beliefs*, that character's goal, and only the events that the character observed when they happened. Given this context, we would expect the LLM's suggestions to be helpful in guiding the planner toward reasonable *character plans,* providing assistance in finding usable explanations in much the same way as it helps find solutions.

We did not fine-tune the LLM on the language of each domain mainly because we were interested in how well it would perform without doing so. Fine-tuning is certainly a possibility that could improve suggestion quality and allow more accurate similarity measurements, but care should be taken to make sure the LLM is not biased to repeat any specific actions provided to it in this training. Further improvements worth considering include other methods of ranking the possible actions based on similarity. For example, it may be helpful to set a similarity threshold and prune actions that are not sufficiently similar to any suggestion.

[3]The original Jailbreak story begins with a prisoner stealing a pack of cigarettes, which triggers the prison bully to threaten him and his friend, setting off the entire chain of events. This effect of the steal action is simplified in the planning domain: stealing the cigarettes directly results in the characters being threatened. We translated the action as: "Ernest steals the cigarettes from the cells. This angers the bully, who threatens to kill both Roy and Ernest." The side effect is important to the story and would not have been inferred from just "Ernest steals the cigarettes from the cells", so we thought it important to state in the translation.

## VI. Conclusion

In this paper we argue that a large pre-trained language model can improve the performance of a state space narrative planning algorithm. Planners are excellent tools for interactive narrative systems because they produce logically sound and coherent stories that adhere to a handwritten symbolic domain. Unfortunately planning does not scale well, and this is especially true for recent methods that model complex character beliefs. We have demonstrated that current large language models are capable of providing helpful suggestions to a planner—suggestions that guide the search both toward the goal and through paths of valid nodes. We have also highlighted two benchmark problems in which the current methodology fails and suggested possible avenues for improvement. We hope these results will inform future attempts to integrate LLM guidance into more complex search strategies.

## Acknowledgement

This research was funded in part by the U.S. Department of Defense.

## References

[1] J. R. Meehan, "TALE-SPIN, an interactive program that writes stories," in *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, 1977, pp. 91–98.

[2] M. Lebowitz, "Story-telling as planning and learning," *Poetics*, vol. 14, no. 6, pp. 483–502, 1985.

[3] M. Mateas and A. Stern, "Structuring content in the façade interactive drama architecture," in *Proceedings of the 1st AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005, pp. 93–98.

[4] J. McCoy, M. Treanor, B. Samuel, A. A. Reed, M. Mateas, and N. Wardrip-Fruin, "Social story worlds with Comme il Faut," *IEEE Transactions on Computational Intelligence and Artificial Intelligence in Games*, vol. 6, no. 2, pp. 97–112, 2014.

[5] R. Evans and E. Short, "Versu–a simulationist storytelling system," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 113–130, 2013.

[6] M. Kreminski, M. Dickinson, M. Mateas, and N. Wardrip-Fruin, "Why are we like this?: exploring writing mechanics for an AI-augmented storytelling game," in *Proceedings of the international conference on the Foundation of Digital Games*, 2020.

[7] R. M. Young, "Notes on the use of plan structures in the creation of interactive plot," in *Proceedings of the AAAI Fall Symposium on Narrative Intelligence*, 1999, pp. 164–167.

[8] M. Cavazza, F. Charles, and S. J. Mead, "Character-based interactive storytelling," *IEEE Intelligent Systems special issue on AI in Interactive Entertainment*, vol. 17, no. 4, pp. 17–24, 2002.

[9] J. Porteous, M. Cavazza, and F. Charles, "Applying planning to interactive storytelling: Narrative control using state constraints," *ACM Transactions on Intelligent Systems and Technology*, vol. 1, no. 2, pp. 1–21, 2010.

[10] J. Porteous, F. Charles, and M. Cavazza, "NetworkING: using character relationships for interactive narrative generation," in *Proceedings of the international conference on Autonomous Agents and Multi-Agent Systems*, 2013, pp. 595–602.

[11] B. Kartal, J. Koenig, and S. J. Guy, "User-driven narrative variation in large story domains using monte carlo tree search," in *Proceedings of the international conference on Autonomous Agents and Multi-Agent Systems*, 2014, pp. 69–76.

[12] R. M. Young, S. G. Ware, B. A. Cassell, and 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, Special Issue on Formal and Computational Models of Narrative*, vol. 37, no. 1-2, pp. 41–64, 2013.

[13] S. G. Ware, R. M. Young, C. Stith, and P. Wright, "The Best Laid Plans," 2014. [Online]. Available: https://nil.cs.uno.edu/projects/blp/

[14] S. G. Ware, E. T. Garcia, M. Fisher, A. Shirvani, and R. Farrell, "Multi-agent narrative experience management as story graph pruning," *IEEE Transactions on Games*, vol. 15, pp. 378–387, 2022.

[15] S. G. Ware and R. M. Young, "Glaive: a state-space narrative planner supporting intentionality and conflict," in *Proceedings of the 10th AAAI international conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014, pp. 80–86.

[16] J. Teutenberg and J. Porteous, "Incorporating global and local knowledge in intentional narrative planning," in *Proceedings of the 2015 international conference on Autonomous Agents and Multiagent Systems*, 2015, pp. 1539–1546.

[17] M. Hua and R. Raley, "Playing with unicorns: AI dungeon and citizen NLP," *Digital Humanities Quarterly*, vol. 14, no. 4, 2020.

[18] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," OpenAI Blog, 2019, accessed 25 May 2023.

[19] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, and S. Kambhampati, "Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change," in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[20] L. Martin, P. Ammanabrolu, X. Wang, W. Hancock, S. Singh, B. Harrison, and M. O. Riedl, "Event representations for automated story generation with deep neural nets," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 868–875.

[21] L. Yao, N. Peng, R. Weischedel, K. Knight, D. Zhao, and R. Yan, "Plan-and-write: towards better automatic storytelling," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 7378–7385.

[22] S. Goldfarb-Tarrant, T. Chakrabarty, R. Weischedel, and N. Peng, "Content planning for neural story generation with aristotelian rescoring," in *Proceedings of the conference on Empirical Methods in Natural Language Processing*, 2020, pp. 4319–4338.

[23] H. Rashkin, A. Celikyilmaz, Y. Choi, and J. Gao, "PlotMachines: outline-conditioned generation with dynamic plot state tracking," in *Proceedings of the conference on Empirical Methods in Natural Language Processing*, 2020, pp. 4274–4295.

[24] A. Ye, C. Cui, T. Shi, and M. O. Riedl, "Neural story planning," *arXiv preprint arXiv:2212.08718*, 2022.

[25] D. McAllester and D. Rosenblitt, "Systematic nonlinear planning," Massachusetts Institute of Technology Artificial Intelligence Laboratory, Tech. Rep., 1991.

[26] T. Trabasso and L. L. Sperry, "Causal relatedness and importance of story events," *Journal of Memory and language*, vol. 24, no. 5, pp. 595–611, 1985.

[27] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin, "Magic3d: high-resolution text-to-3D content creation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 300–309.

[28] M. Heikkilä, "The algorithm: AI-generated art raises tricky questions about ethics, copyright, and security," MIT Technology Review, 2022, accessed 25 May 2023. [Online]. Available: https://shorturl.at/iswxJ

[29] S. G. Ware and C. Siler, "Sabre: A narrative planner supporting intention and deep theory of mind," in *Proceedings of the 17th AAAI International Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2021, pp. 99–106.

[30] R. E. Fikes and N. J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[31] C. Siler and S. G. Ware, "Solution density and search strategy in narrative generation," *IEEE Transactions on Games*, vol. 14, no. 4, pp. 715–724, 2022.

[32] S. G. Ware and R. Farrell, "A collection of benchmark problems for the Sabre narrative planner," https://github.com/sgware/sabre-benchmarks/blob/main/report.pdf, Narrative Intelligence Lab, University of Kentucky, Tech. Rep., November 2023.


## APPENDIX

STEP 1 PROMPT: ⟨inital state, character goals, action names, story goal, history⟩
The hero is alive. The merchant is alive. The guard is alive. The bandit is alive. The bandit is a criminal. The hero is at the cottage. The merchant is at the market. The guard is at the market. The bandit is at the camp. The merchant has the medicine. The hero has the hero's coin. The bandit's coin is in the chest. The guard has the guard's sword. The bandit has the bandit's sword. The merchant has the merchant's sword. There is a path between the crossroads and the cottage. There is a path between the crossroads and the market. There is a path between the crossroads and the camp. The hero does not know where the bandit is. The merchant does not know where the hero's coin is. The merchant does not know where the bandit's coin is. The guard does not know where the bandit is. The bandit believes that the hero is at the crossroads. The hero wants to bring the medicine to the cottage. The bandit wants to collect valuable items in the chest. The merchant wants coins and is not willing to commit crimes. The guard wants to attack criminals. Characters can walk, buy, loot, attack, rob, report, and take. The story must end with the hero either being attacked or having the medicine at the cottage. List two different actions that could happen first. State each as a short sentence on its own line.

RESPONSE:
The hero walks to the market. [ 0.0193 -0.0196 -0.0155 0.0039 -0.0137 0.0044 -0.0111 ... ]
The merchant buys the medicine. [ 0.0209 -0.0029 0.0037 -0.0149 -0.0398 0.0168 -0.0118 ... ]

| Cost Assigned | Possible Action |
|---|---|
| 4 | `walk(Merchant,Market,Crossroads)`<br>The merchant walks from the market to the crossroads.<br>[ 0.0127 -0.0162 0.0027 0.0010 -0.0151 0.0071 -0.0062 ... ] |
| 4 | `walk(Hero,Cottage,Crossroads)`<br>The hero walks from the cottage to the crossroads.<br>[ 0.0180 -0.0119 -0.0148 0.0068 -0.0066 0.0250 -0.0010 ... ] |
| 5 | `walk(Guard,Market,Crossroads)`<br>The guard walks from the market to the crossroads.<br>[ 0.0212 -0.0166 0.0037 0.0006 0.0078 0.0180 -0.0087 ... ] |
| 6 | `walk(Bandit,Camp,Crossroads)`<br>The bandit walks from the camp to the crossroads.<br>[ 0.0097 -0.0156 0.0103 -0.0057 -0.0126 0.0345 -0.0131 ... ] |
| 7 | `attack(Merchant,Merchant,Market)`<br>The merchant attacks herself.<br>[ -0.0208 -0.0121 -0.0026 -0.0045 -0.0171 0.0047 -0.0037 ... ] |
| 7 | `attack(Merchant,Guard,Market)`<br>The merchant attacks the guard.<br>[ -0.0029 -0.0182 0.0027 -0.0065 -0.0088 0.0301 -0.0049 ... ] |
| 8 | `attack(Hero,Hero,Cottage)`<br>The hero attacks himself.<br>[ -0.0196 -0.0050 -0.0066 0.0003 -0.0166 0.0147 -0.0175 ... ] |
| 8 | `attack(Guard,Merchant,Market)`<br>The guard attacks the merchant.<br>[ 0.0018 -0.0228 0.0030 -0.0096 -0.0022 0.0345 -0.0107 ... ] |
| 9 | `report(Merchant,Camp,Market)`<br>The merchant reports seeing the bandit at the camp to the guard.<br>[ 0.0097 -0.0198 0.0214 -0.0077 0.0015 0.0517 -0.0261 ... ] |
| 10 | `take(Bandit,BanditCoin,Chest,Camp)`<br>The bandit takes the bandit's coin from the chest.<br>[ -0.0035 -0.0230 0.0241 -0.0225 -0.0286 0.0292 -0.0105 ... ] |
| 10 | `attack(Guard,Guard,Market)`<br>The guard attacks himself.<br>[ -0.0112 -0.0042 0.0143 -0.0048 0.0010 0.0363 -0.0211 ... ] |
| 10 | `attack(Bandit,Bandit,Camp)`<br>The bandit attacks himself.<br>[ -0.0229 0.0009 0.0227 0.0011 -0.0250 0.0339 -0.0185 ... ] |

STEP 2 PROMPT: ⟨inital state, character goals, action names, story goal, history⟩
The hero is alive. The merchant is alive. The guard is alive. The bandit is alive. The bandit is a criminal. The hero is at the cottage. The merchant is at the market. The guard is at the market. The bandit is at the camp. The merchant has the medicine. The hero has the hero's coin. The bandit's coin is in the chest. The guard has the guard's sword. The bandit has the bandit's sword. The merchant has the merchant's sword. There is a path between the crossroads and the cottage. There is a path between the crossroads and the market. There is a path between the crossroads and the camp. The hero does not know where the bandit is. The merchant does not know where the hero's coin is. The merchant does not know where the bandit's coin is. The guard does not know where the bandit is. The bandit believes that the hero is at the crossroads. The hero wants to bring the medicine to the cottage. The bandit wants to collect valuable items in the chest. The merchant wants coins and is not willing to commit crimes. The guard wants to attack criminals. Characters can walk, buy, loot, attack, rob, report, and take. The story must end with the hero either being attacked or having the medicine at the cottage. The story begins with the following steps:

The merchant walks from the market to the crossroads.

List two different actions that could happen next. State each as a short sentence on its own line.

RESPONSE:

| | |
|---|---|
| The guard follows the merchant to the crossroads. | [ 0.0169 -0.0303 -0.0009 -0.0072 0.0020 0.0341 0.0042 ... ] |
| The hero walks from the cottage to the crossroads. | [ 0.0180 -0.0119 -0.0148 0.0068 -0.0066 0.0250 -0.0010 ... ] |

| Cost Assigned | Possible Action |
|---|---|
| 0 | `walk(Hero,Cottage,Crossroads)`<br>The hero walks from the cottage to the crossroads.<br>[ 0.0180 -0.0119 -0.0148 0.0068 -0.0066 0.0250 -0.0010 ... ] |
| 2 | `walk(Guard,Market,Crossroads)`<br>The guard walks from the market to the crossroads.<br>[ 0.0212 -0.0166 0.0037 0.0006 0.0078 0.0180 -0.0087 ... ] |
| 3 | `walk(Merchant,Crossroads,Cottage)`<br>The merchant walks from the crossroads to the cottage.<br>[ 0.0216 -0.0129 -0.0001 0.0071 -0.0102 0.0241 -0.0007 ... ] |
| 4 | `walk(Merchant,Crossroads,Market)`<br>The merchant walks from the crossroads to the market.<br>[ 0.0153 -0.0176 0.0011 -0.0027 -0.0184 0.0035 -0.0092 ... ] |
| 4 | `walk(Merchant,Crossroads,Camp)`<br>The merchant walks from the crossroads to the camp.<br>[ 0.0187 -0.0165 0.0045 -0.0037 -0.0140 0.0274 -0.0143 ... ] |
| 4 | `walk(Bandit,Camp,Crossroads)`<br>The bandit walks from the camp to the crossroads.<br>[ 0.0097 -0.0156 0.0103 -0.0057 -0.0126 0.0345 -0.0131 ... ] |
| 8 | `take(Bandit,BanditCoin,Chest,Camp)`<br>The bandit takes the bandit's coin from the chest.<br>[ -0.0035 -0.0230 0.0241 -0.0225 -0.0286 0.0292 -0.0105 ... ] |
| 8 | `attack(Hero,Hero,Cottage)`<br>The hero attacks himself.<br>[ -0.0196 -0.0050 -0.0066 0.0003 -0.0166 0.0147 -0.0175 ... ] |
| 8 | `attack(Guard,Guard,Market)`<br>The guard attacks himself.<br>[ -0.0112 -0.0042 0.0143 -0.0048 0.0010 0.0363 -0.0211 ... ] |
| 9 | `attack(Merchant,Merchant,Crossroads)`<br>The merchant attacks herself.<br>[ -0.0208 -0.0121 -0.0026 -0.0045 -0.0171 0.0047 -0.0037 ... ] |
| 9 | `attack(Bandit,Bandit,Camp)`<br>The bandit attacks himself.<br>[ -0.0229 0.0009 0.0227 0.0011 -0.0250 0.0339 -0.0185 ... ] |