Towards Managing Epistemic Complexity in Narrative Games

Jennifer Wellnitz mawellni@ncsu.edu
Chris Martens martens@csc.ncsu.edu
POEM Lab
North Carolina State University
Raleigh, North Carolina, USA

Abstract

Interactive narrative experiences can get complex: authors create combinatorial, generative play spaces that can grow to involve a number of states exponential in the number of story-world variables. When *character beliefs* are a key aspect of game state, authoring and debugging are especially onerous, leading to game bugs in production. We explore this problem in the context of the interactive narrative game *Elsinore*. We present an application of Dynamic Epistemic Logic to model character belief update in Elsinore with the goal of controlling epistemic complexity. In principle, an epistemic logic engine should offload some of the manual effort in managing the knowledge effects of information sharing between characters. In practice, we find that our encoding fails to significantly reduce authoring complexity and bug-prone complexity from character belief management. We discuss why this turns out to be the case and potential future directions.

Figure 1: Screenshot of Elsinore's primary interface for moving through the environments and observing conversations.

INTRODUCTION

Interactive narratives are a growing genre of published games and an interesting area for games and AI research. Many interactive narratives, as they become more complex, feature a vast number of possible result states for any action, based on a number of factors and decisions from elsewhere in the story. These states can become difficult to manage, requiring long chains of complicated preconditions and many versions of similar states that reflect the current state of the game in all of its detail. Manually authoring these states, their associated actions and transitions, and their preconditions can be onerous, and can create bugs in the code. These problems can be exacerbated in games where not all characters share the same information, vastly increasing the number of factors to keep track of.

In the long term, we want to minimize errors related to manual authoring of complex character belief changes. Specifically, we are interested in mitigating *narrative logic bugs*, in which the presence of behavior or the advancement to a game state is not permitted by the (often implicit) rules of the story world defined by the author, e.g. disabling a player action that should be possible, performing an action that should be impossible, or learning information from nothing. Because some surprising behaviors are desirable when creating emergent narrative experiences, we differentiate this kind of bug from unplanned event sequences that do not violate the rules of the world.

In this paper, we present our first steps towards this goal, a case study of the game *Elsinore*. Elsinore (Golden Glitch Studios 2019) is an interactive narrative game in which the player takes the role of Ophelia in the Shakespearean play *Hamlet*. The story begins with the plot of Hamlet as written, but then Ophelia is murdered instead of her canonical death by suicide. She then wakes up in her bed, realizing that she is in a time loop that resets on her death. The player is then tasked with manipulating the events of Hamlet to create a more favorable outcome for Ophelia and break the time loop. Ophelia's means of changing the story is an *information exchange* mechanic: as Ophelia moves through the world, she can observe conversations and events (see figure 1), which are logged in a record the player can access (see Figure 2). She can then participate in conversations and share certain elements of knowledge she has learned, called Hearsay, with other characters—who in turn may change their actions in response to new information (see Figure 3). A positive feedback loop is created by the fact that the more information the player learns, the more ways they can change the story, generating new events that reveal new information.

Figure 2: Screenshot of Elsinore showing how the game reports knowledge updates for the player and NPC characters.

Figure 3: Screenshot of the hearsay mechanic in Elsinore, where topics in white can be shared with the NPC, and greyed out topics are unavailable

This game meets our criteria as a large and intricate interactive narrative with a number of bugs arising from that complexity. Issues relating to the management of character beliefs are particularly prevalent within Elsinore. Searching Twitter for player bug reports reveals several, such as this tweet by Amanda Gentzel:

> @elsinoregame Loving the game so far, but I think I found a bug. It is Saturday night (11pm), and when I selected to ask Horatio about my father's murder, I told him I feared Hamlet 'may' kill my father, and Horatio said it was impossible. But my father is definitely dead

This bug deals with characters having inconsistent beliefs. Whether it was in the writing of the dialogue or a bug in what conditions were being checked for that scene to play out, the actions of the player's character do not reflect the player's mental model of that character's knowledge. This is not an isolated instance; another tweet by Connor Fallon reads:

> So, @elsinoregame bug of the day: Polonius would preemptively arrest himself under certain conditions, going to sulk in his jail cell before anyone actually locks him up. He'd still leave his cell to do errands, though.

This bug also exemplifies problems in character beliefs — Polonius updates his behavior due to the beliefs and actions of other characters, but before the event that would logically change his character's beliefs. Finally, in a tweet by one of the developers of Elsinore, Katie Chironis, says

> it would be a slight exaggeration to say that elsinore is 9 billion if/else statements duct taped together, but only slightly. so... we'll just keep adding more.

We refer to the complexity that creates character belief update bugs such as these as *epistemic complexity*. Because of Elsinore's epistemic complexity, we decided to try constructing a model of the game in which the belief update mechanics are managed with *Dynamic Epistemic Logic* (DEL), a formalism for reasoning about how agent beliefs change over time in the presence of imperfect information and communicative actions. DEL seems like a good fit for reasoning about Elsinore, not only because of the game's reliance on belief update, but also because of how it conceptualizes time in terms of possible worlds, computing which events are possible or not in the current timeline. As such, the aim for this project is to investigate DEL as a tool for managing epistemic complexity in Elsinore, testing the hypothesis that common bugs in the Elsinore game could be avoided by using Dynamic Epistemic Logic to handle character beliefs.

RELATED WORK

Little work has been done to determine the efficacy of Dynamic Epistemic Logic as basis or component for computational and interactive narratives. However, other techniques have been employed to model narrative games, manage large sets of characters in interactive narrative, and to reduce authorial burden in these games. For example, Claire Wolf's The Teeny Tiny Mansion (TTTM) (Wolf 2017) is a simple model of an adventure game which is formally verified to be solvable. The aim of this project is not to create a playable experience per se, but to prove that adventure games are able to be formally verified to be solvable. Though questions of solvability are not the aim of this paper, the genres of adventure games and interactive narrative are closely related, and this project strives to do something similar to TTTM in creating a model of an engine for character beliefs rather than a fully playable experience.

Other work into managing complex stories may focus on different aspects of NPC interaction to improve upon, rather than belief and knowledge management. Ware et al. frame the problem of managing NPCs in complex narrative as a story graph pruning problem (Ware et al. 2019). The aim presented by Ware is similar to the aim of this paper, to manage complex narratives programmatically by using a system to simplify some aspect of the NPCs that would otherwise have to be manually authored. However, where this paper does not address NPC actions in emergent narrative, Ware's specifically simplifies the concept of NPC belief states, showing that these methods of addressing similar problems both have their limitations.

Much work has already been done to effectively manage character beliefs in complex story environments that doesn't leverage Dynamic Epistemic Logic. For example, Farrell proposed a system where character beliefs are handled abstractly as action histories in the domain of story graph pruning (Farrell 2018). Although the intended domain is different from that of this project, the underlying motivation is quite similar — to develop a method of managing epistemic complexity in interactive narratives. This project focuses on offsetting the problem of managing belief states through the use of Dynamic Epistemic Logic and triggering distinct events in a story without tracking a full history, while Farrell's project focuses on abstracting away from belief management through the use of action history as an analogous but more efficient metric.

Finally, one of the main goals of this research is to limit the number of bugs introduced into an interactive narrative game by managing NPC belief states. Thus, this project seeks to simplify the authoring task for these games. While we present one method to do so, many others have been employed. James Thomas presents a novel system for interactive narrative authoring which employs a collaborative approach to planning agents, giving some manual control to the author but otherwise streamlining the authoring task significantly (Thomas and Young 2006). While the system described in this paper is significantly different, both belong to a growing body of literature concerning assisted or programmatic authoring of narratives.

APPROACH

This research focuses on the potential use Dynamic Epistemic Logic for character belief management in the field of interactive narrative. Dynamic Epistemic Logic is a logical system which represents imperfect information. It incorporates actions that effect the genuine state of the world, as well as actions that reveal or obfuscate pieces of that world, changing the beliefs of agents about which worlds they view as possible.

To represent Dynamic Epistemic Logic programmatically, and in the context of a multiagent game, this project uses the Ostari language. Ostari is a language meant to facilitate multiagent interactions centering around incomplete knowledge and changing belief states (Eger and Martens 2017). Ostari is a computational implementation of the formal mathematical language Alexandru Baltag used to describe Dynamic Epistemic Logic (Baltag 2002). This language allows for agents to maintain a set of worlds they believe possible, and for actions to be defined that control the flow of information to those agents, revealing information to some and obfuscating it to others, causing those agents to consider more possible worlds from the uncertainty.

This language is perfect to model the dynamics surrounding information and belief in the Elsinore narrative. For example, Ophelia will continue to be murdered every loop until she finds a way to prevent it. When she discovers who the murderer is, she will still continue to die until she alerts the guard captain and has him apprehend the killer. This is because the knowledge the player (and Ophelia) have is not the same as the knowledge that each character has. If the guard captain doesn't know who the killer is, he can't act on that information, even if the player does. Thus, the game must keep track of varying levels of knowledge and beliefs for each of the characters, something Ostari was built to do.

Ostari alone, however, is primarily useful in simulating interactions between digital agents, not in interfacing with those agents directly, and having a player become one the characters. Thus this paper also makes extensive use of work done by Henry Mohr in using Ostari as an engine to facilitate player interaction within a narrative (Mohr, Eger, and Martens 2018). Mohr's work deals with using Ostari in tandem with Python to create mysteries that a player interacts with to solve. Thus, the work done to interface with Ostari was instrumental in the ability to have a player take part in the narrative for this project.

The bulk of this research was done by building a model of a small section of Elsinore using both Ostari and Python. Python functions as a wrapper for Ostari and is charged with both running the Ostari code and managing certain metadata and world state information for the game itself. Ostari, on the other hand, defines the actions of the game, and manages events — their effects and preconditions — as well as character beliefs.

The majority of the functionality of the game is handled by the Python layer, and the specific functionality of each component is detailed in Figure 4. This includes handling player input, scheduling, and running the Ostari code on each "turn" of the game. Essentially, time is stagnant except in relation to player action, so each turn is composed of

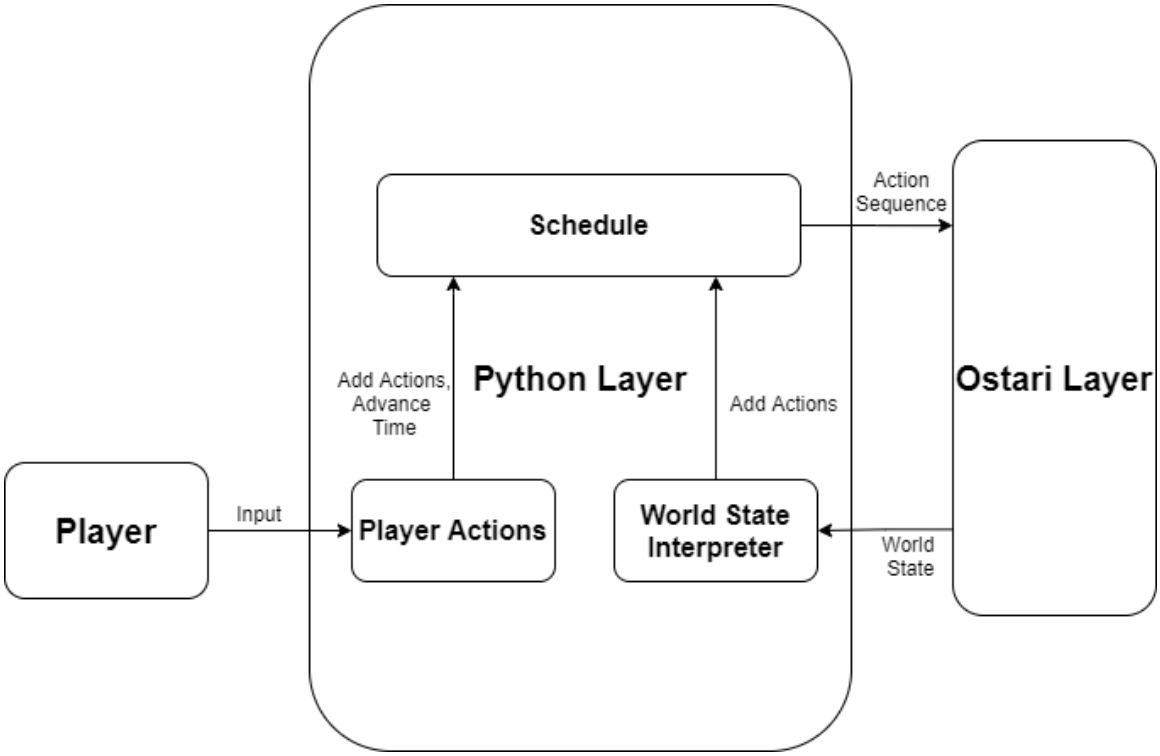

Figure 4: Diagram of the overall system

the player executing an action, and the game resolving the effects of that action. Depending on the action itself, new actions may be put on the schedule or time may be advanced. The Python code interfaces with Ostari by taking a template of the Ostari code for the game, and replacing the list of actions to execute with the list of actions that have occurred thus far in the game. The Python layer then runs the Ostari as a sub-process, and captures and interprets the output. Thus, the Python code maintains a list of actions that have occurred and reruns the Ostari on every player turn.

There are five gameplay actions available. The player may wait, tell hearsay to another character, view an event, go to another location, or query the status of another character (mostly for the purpose of experimentation rather than game play). These actions are described in more detail in Table 1.

For example, Figure 3 illustrates how player information and information sharing is handled in Elsinore. The image depicts the hearsay sharing mechanic in Elsinore, which is primarily how the player affects the game. Here the player can see all of the pieces of information they have learned throughout the game by observing events and talking with various NPCs. They can share certain pieces of this information with the character they're speaking with to influence the information that character knows, learn new information from the resulting conversation, and possibly change that character's actions further down the timeline. All of this happens in the interface shown in Figure 3 and in the resulting conversation, and is played out directly in the game.

By way of comparison, in the model of Elsinore built using our system, all of these actions — except for wait — are instantaneous. The only way to advance time is through the wait function. Further, no actual dialogue is displayed and hearsay is shared using specific codes to represent each piece of information. These choices were made primarily because, although this model is fully playable, playing the game through it does not facilitate the story in much detail. Rather, this model is a simplified version of the game meant to help evaluate it, rather than a playable potion of the game itself.

Further, the design of this project was split into two stages — skeleton and content. The skeleton makes up the mechanics of the game, which could potentially be translated to other games of a similar type. The content is an implementation of a subset of the content within the Elsinore game such that the mechanics of the skeleton can be tested in a genuine game and so the results can compared. While Elsinore contains many locations and characters, for the purpose of this paper some locations were collapsed into each other (resulting in the composite locations: Grounds/Docks, Lower Hall, Upper Hall, Main Hall, and Courtyard) and only the characters (Bernardo, Brit, Polonius, Laertes, and Hamlet) and events necessary such that the player can solve their murder were included. This was to give a small cross section of the game for comparison while still focusing mainly on the design of the mechanics.

Action	Ostari	Elsinore	Effect
Wait	User specifies an amount of time to wait, instantaneous shift in time	User holds a button while the world speeds up	Time fast forwards
Tell Hearsay	User specifies a character and a piece of hearsay	User approaches a character and chooses a piece of hearsay	Character reacts to hearsay and new events are scheduled/new hearsay is learned
Observe Event	User indicates that they want to observe an event, event ends immediately	User clicks on an icon above an event, witnesses it play out in real time	Player may learn hearsay
Move	Instantaneous travel	Real time travel	Player moves
Query	User specifies a character	Information about characters is seen in journal and map screens	Player learns about a character and sees their location

Table 1: Action Comparison from Elsinore to Ostari

Character Beliefs

The main purpose of Ostari in this project is to manage the beliefs of the game characters. In fact, the main purpose of this project is to see if managing character beliefs using Dynamic Epistemic Logic is a viable alternative to hand authoring states and transitions in narratives which make heavy use of character beliefs. The Ostari code, dynamically generated by the Python layer, executes a series of actions which represent all of the events that have happened from the start of the game to the present. Each of these actions are defined in the Ostari code, and executing them in the order that they occurred in the game results in a world states representing each character's beliefs at the current moment in the game. Python does pattern matching against the current state of the world to determine which actions have their preconditions met, but the tracking of agents beliefs about themselves and other agents is all internal to Ostari.

Character beliefs had to be specifically authored to fit the mechanics of Elsinore. Specifically, NPC characters have beliefs and goals, which represent their knowledge about the world and their current drive respectively. Ophelia, the main character, on the other hand, has pieces of information called hearsay which she can potentially share with NPCs to change their behavior and knowledge to advance the game. Thus character beliefs are represented as objects of those three categories: beliefs, goals, and hearsay. This was simpler than representing the ideas in each of these beliefs using complex relationships (for example, defining primitive actions within Ostari and expressing each belief as a conjunction of these primitives), as the original games treated them as simple Boolean variables in terms of triggering events.

As such, the character beliefs are primarily used as preconditions for scheduling events. Beliefs are formatted as Boolean variables and events use beliefs as preconditions to mimic the original design philosophy of Elsinore. The aim of this research is to determine if managing these beliefs with Dynamic Epistemic Logic cuts down on errors, and this section of functionality is the core addition of this research.

Events

The main building block of the narrative in Elsinore are events. Events are situations that arise due to the current state each of the characters' beliefs and the state of the world, and may or may not involve the player. As events are often both the cause and effect of changes in the world, and the product of character beliefs, it was important to show how events were affected by the changes to belief representation. Within this model, events are dealt with both on the Python and on the Ostari side. The Python layer contains metadata about the events like what time they take place and in what location. It also is responsible for deciding which events have occurred and adding them to the action queue to be run in Ostari. Ostari, on the other hand, has actions that represent the preconditions and effects of each event, and when the Python layer tells the Ostari to run an event, Ostari is what deals with the effects of that event occurring.

For each event, there are two actions in Ostari: one which contains only the preconditions for the event and sets a predicate indicating to the Python layer to schedule the event, and one that contains the effects of the event, and is scheduled by the Python layer to execute the event. These preconditions are typically a mix of character beliefs and world state information (who has died or left the castle and so on). The effects set character beliefs, change the world state, and can give the player, should they be observing the event or be otherwise involved, new hearsay. These functionalities are split into two actions such that the Python can schedule an event in advance of running it, rather than having to execute the full action when the preconditions for the event are met (but before the time it was supposed to take place) or having to check if each action was valid at every time step to avoid missing one.

Events in Python, on the other hand, contain all the information needed for the Python layer to schedule and eventually execute the event. This includes a start time, stop time, location, all of the characters involved, whether or not the player is inherently involved in the event, whether or not the event results in a reset (player death), and the name of the event in Ostari. This data must be contained in Python because timekeeping is handled in this layer and because all of that information is necessary to add the event dynamically to the Ostari code. However, as Python layer needs to pattern match against the current World state to determine which events to schedule, Python must also maintain a list

of preconditions for each event.

Scheduling

As stated earlier, the game runs off of a turn based model where the player takes some action and the world responds to it. Each response causes the Ostari layer to be rerun and a new world state to be generated. From this world state, the Python layer finds whichever events have not been scheduled, completed or cancelled, and finds among them those events who's preconditions have also been met and adds them to the schedule.

After each run of the Ostari layer, the Python layer iterates through each of the possible events. Each event that is not already scheduled, cancelled, or completed, and that has all of its preconditions met, is added to the schedule. Further, this is also the stage where events are cancelled. If an event on the schedule is deemed impossible (mostly by the presence of the $impossible(event)$ predicate set by some other event), that event is removed from the schedule. This is only one way that events may be cancelled, the other is by conflicts.

To deal with the scheduling of events, the Python layer keeps track of a current master schedule. Events are added and removed from this schedule as the game progresses and events are executed when observed by the player or when the current time passes their end time on a wait action. Further, when events conflict over resources, particularly overlapping in time and requiring some of the same characters present, the scheduling algorithm simply defaults to cancelling the older event in lieu of the new. This behavior is present in the original game as well and represents one's plans being changed with new information.

CONCLUSIONS AND FUTURE WORK

Overall, the results of this research were inconclusive, but this specific approach yielded little to no improvement over the original design. These results were partially caused by the limitations of the interactivity of Ostari, but, beyond technical limitations, this project uncovered a significant incompatibility between the structure of Elsinore and the ideas of Dynamic Epistemic Logic.

Although a similar approach to Henry Mohr's work was used to allow the player to interface with Ostari, the Ostari language is still not particularly suited to complex user interaction. This mostly is a result of the fact that Ostari cannot be interfaced with at runtime, but only have the results fetched after the program finishes. An early planned approach involved querying Ostari to determine which actions are currently possible. This would have replaced having Python track prerequisites and pattern match to schedule events. Would this functionality been possible, Ostari would have been able to be responsible for more of the functionality of the game as a whole.

Further, the Python layer ended up being responsible for a large proportion of the functionality of the game. The purpose of this project was to model the dynamics of information amongst the characters solely through Ostari to determine the efficacy of Dynamic Epistemic Logic in representing those ideas in a narrative, but with the functionality of Ostari not being compatible with the method of interactivity required for a game of this style, even much of the work surrounding character beliefs was pushed to the Python layer. Determining which actions were now valid and scheduling should have, ideally, been in the domain of the Ostari layer, but the capability simply wasn't there.

On a different note, the narrative structure of Elsinore was not as compatible with the ideas of Dynamic Epistemic Logic as initially thought. Though the story and gameplay were largely dependent on the idea of incomplete information and possible worlds, the actual story itself was too strictly authored to make full use of actions in Ostari as general rules and not simply triggers for a structured sequence. Initial plans included expressing each belief of characters as some combination of generic predicates (kills(player, player), etc), but because the original game used these beliefs as more of boolean triggers, and because events are always structured and pre-authored rather than emergent, this approach was quickly scrapped — it would simply make preconditions needlessly laborious while not giving any real benefit. In a game that focuses more on a generic or emergent narrative, especially one that is procedurally generated, this approach is likely to be much more effective.

One of the key aims of this process was to determine if it helps to mitigate bugs. Unfortunately, it did not. Because events are so definite in this game, they essentially had to be authored in Ostari much as they are in the original code — with long chains of preconditions. As such, many of the potential errors come from improper preconditions and improper effects, both of which are still distinctly possible in this system. In fact, with two separate code bases and with preconditions authored in both, inconsistencies may be more likely.

While this system may not work well for Elsinore, that is partly the product of the type of game Elsinore is. Referring back to the bug found by Amanda Gentzel, where Horatio's dialogue doesn't accurately reflect the current state of the world or his current knowledge, that issue could be caused by not checking the right preconditions in deciding what scene plays when the player brings up that topic with him, or it may be that dialogue for that specific situation did not exist at the time. The latter option highlights an important limitation for our system — even if it handled knowledge and preconditions perfectly, in a game like Elsinore, where the scenes are heavily scripted and pre-authored, the authors themselves still have to write all of the dialogue for every conceivable scene. Our system would not be able to help prevent this bug. As such, our system would work best in less story rich environments, where characters react based on their beliefs but only from a pool of generic actions rather than scripted scenes. A prime example of these types of games would be The Sims series.

Though this project was ultimately not successful, elements of it have led to other potential avenues for research. One such avenue is an expansion of Ostari to incorporate runtime intervention. Specifically, adapting Ostari to both allow for user input (in the form of specifying actions to be executed) and querying of world state, possible actions, and so forth during runtime, between actions would suffice.

These alterations would give more fine tuned control in executing Ostari code, as well as open up a whole new type of game to be viable under the Ostari language.

Finally, though the content portion is specific to modelling Elsinore, the skeleton used to adapt the content is suitably generic to be useful in other similar, albeit a bit less structured, narratives. This skeleton could serve as a tool to both interface with Ostari and to design more emergent narratives which could make use of Ostari's unique system of character beliefs. Hopefully, this tool could be used to do further research into how Dynamic Epistemic Logic pairs with interactive narratives.

References

Baltag, A. 2002. A logic for suspicious players: Epistemic actions and belief–updates in games. *Bulletin of Economic Research* 54(1):1–45.

Eger, M., and Martens, C. 2017. Practical specification of belief manipulation in games. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Farrell, R. 2018. Experience management with beliefs, desires, and intentions for virtual agents. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Golden Glitch Studios. 2019. Elsinore. Published online.

Mohr, H.; Eger, M.; and Martens, C. 2018. Eliminating the impossible: A procedurally generated murder mystery. In *AIIDE Workshops*.

Thomas, J. M., and Young, R. M. 2006. Author in the loop: Using mixed-initiative planning to improve interactive narrative. In *Workshop on AI Planning for Computer Games and Synthetic Characters*.

Ware, S. G.; Garcia, E. T.; Shirvani, A.; and Farrell, R. 2019. Multi-agent narrative experience management as story graph pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, 87–93.

Wolf, C. 2017. The teeny tiny mansion (tttm). Technical report, University of Applied Arts Vienna.